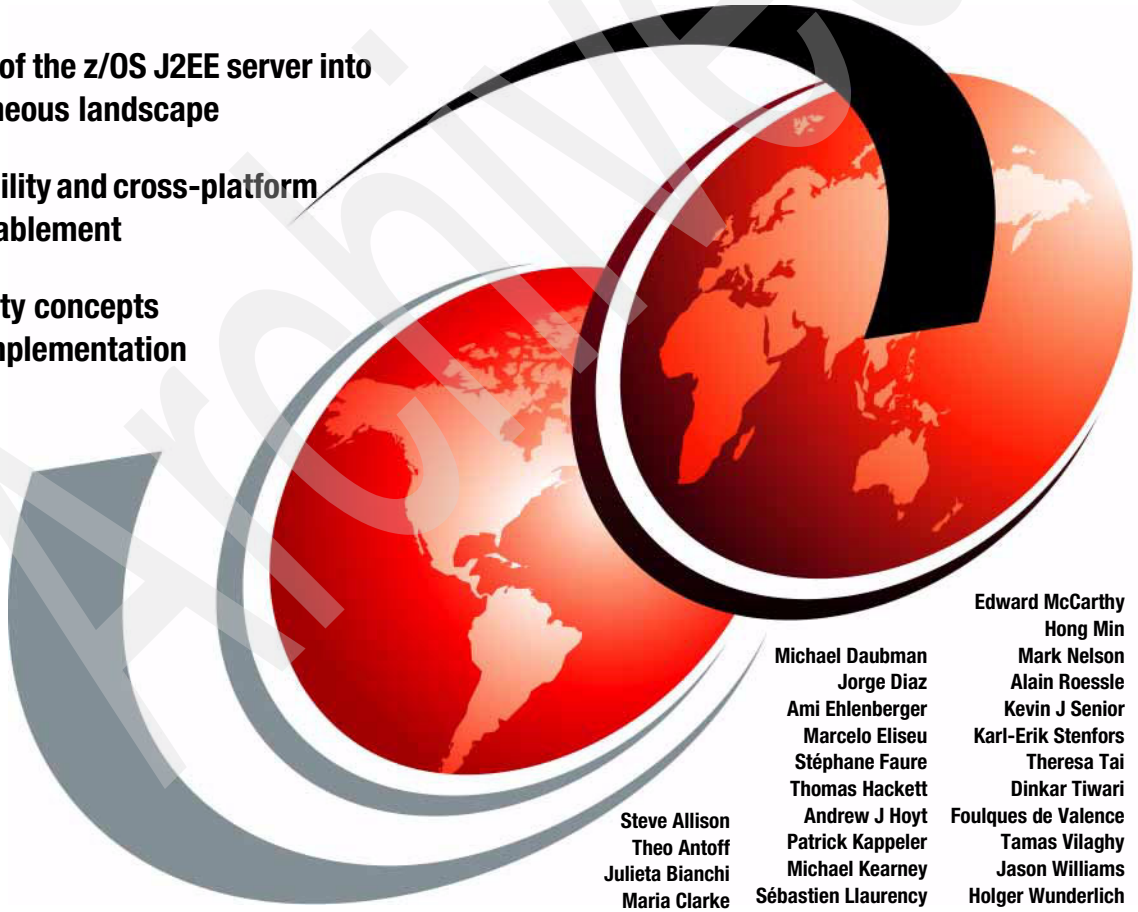
WebSphere® software

IBM

WebSphere Application Server for z/OS V5 and J2EE 1.3 Security Handbook

- Integration of the z/OS J2EE server into a heterogeneous landscape
- Interoperability and cross-platform security enablement
- J2EE security concepts and their implementation



Steve Allison
Theo Antoff
Julieta Bianchi
Maria Clarke
Michael Daubman
Jorge Diaz
Ami Ehlenberger
Marcelo Eliseu
Stéphane Faure
Thomas Hackett
Andrew J Hoyt
Patrick Kappeler
Michael Kearney
Sébastien Llaurency
Edward McCarthy
Hong Min
Mark Nelson
Alain Roessle
Kevin J Senior
Karl-Erik Stenfors
Theresa Tai
Dinkar Tiwari
Foulques de Valence
Tamas Vilaghy
Jason Williams
Holger Wunderlich

Redbooks

ibm.com/redbooks

International Technical Support Organization

WebSphere Application Server for z/OS V5 and J2EE 1.3 Security Handbook

June 2005

SG24-6086-01

Note: Before using this information and the product it supports, read the information in “Notices” on page xvii.

Second Edition (June 2005)

This edition applies to Versions 5 and 5.1 of IBM WebSphere Application Server for z/OS and OS/390.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
CICS®
DB2®
DFSORT™
Domino®
e-business on demand™
@server®
eServer™
HiperSockets™
ibm.com®
IBM®
IBMLink™
IMS™
Lotus®
MQSeries®
Multiprise®
MVS™
OS/390®
Parallel Sysplex®
RACF®
Redbooks (logo) ™
Redbooks™
RETAIN®
RMF™
S/390 Parallel Enterprise Server™
S/390®
SecureWay®
Tivoli®
VTAM®
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

What do you think of when someone mentions z/OS security? Probably of something that is trustworthy, or even impenetrable. Perhaps you also think of something that is a little complex and challenging to administer.

What comes to mind when someone mentions Internet security? Perhaps you think of prominent Web sites that have been maliciously “hacked” or credit card numbers that have been stolen.

Using working examples of code and configuration files, in this IBM Redbook, we explain how you can run your Web-enabled applications with as high a level of security as other z/OS applications and subsystems, even if those applications were written or originally deployed on another platform, by using the Java 2 Platform Enterprise Edition (J2EE) programming model and IBM WebSphere Application Server for z/OS and OS/390®.

This redbook will help architects, application programmers, WebSphere and security administrators, and application and network architects to understand and use these products.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Holger Wunderlich is a Project Leader at the IBM International Technical Support Organization, Poughkeepsie. He writes extensively and teaches IBM classes worldwide on all areas of e-business with IBM @server zSeries. Before joining the ITSO in 2001, he was a cross-server consulting IT Specialist in Germany. He worked for the IBM eServer™ Technical Support Team. He also worked in the OS/390 area as a systems programmer, systems engineer, and as technical support specialist in the z/OS UNIX® and WebSphere world for three years. His areas of expertise include cross-server consulting, z/OS UNIX System Services, e-business infrastructure, e-business security concepts, large scale ISP architectures, and Java/390. He has written several Redbooks™ and developed class material addressing large systems topics.

Theo Antoff is a Senior RACF Architect in Australia, as well as the director of his own companies in Australia (AG Glomar Pty Ltd) and the U.S. (Antoff IT, Inc.), consulting on all aspects of the IBM Security Server components. He has 15 years of experience in RACF and MVS™ Systems programming. His projects include SDSF, CICS®, and DB2® conversions to RACF, CA-Top Secret to RACF migrations, merging RACF databases, and security technical reviews. In his previous career as a physicist, Theo was involved in the research and development of the technology of non-volatile memories based on MIS structures. He has worked for IBM in Australia for three years.

Maria Clarke is an e-business Specialist in IBM Global Services Australia. She has 18 years of experience in supporting OS/390, primarily as a Systems Programmer. This includes four years of UNIX System Services and OS/390 e-business support. Most recently, she has worked in e-business enablement services supporting WebSphere across multiple platforms. She holds a Bachelor of Science degree in Computing/Mathematics from Monash University. Maria is an IBM Certified Systems Expert in “Administration for IBM WebSphere Application Server” and “IBM WebSphere Application Server V4.0.1. for z/OS and OS/390.”

Jorge Diaz has worked as a Senior Consultant, Architect, and Engineer, focusing on in the past decade the areas of distributed, object-oriented software architecture and development. At IBM, Jorge provides consulting for the WebSphere family of products throughout the Americas and Europe.

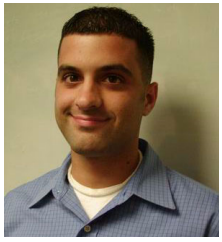
Ami Ehlenberger is a Staff Software Engineer and has been with IBM for the past four years. She started in OS/390 development in 1999, and has since worked in the areas of integration test, solution design, and services. She has a Bachelor of Science degree in Computer Science from Indiana University of Pennsylvania and an MBA in e-business from the University of Phoenix. Her areas of expertise include LDAP, WebSphere, and IBM Tivoli Access Manager. Ami currently works for the Custom Technology Center, an eServer services organization.

Marcelo Eliseu is an e-business Specialist in IBM Global Services Brazil. He has been working with the WebSphere family products on multiplatforms for three years and has 13 years of experience with OS/390. His areas of expertise include Java and WebSphere Application Server support, solutions, and integration. Marcelo is an IBM Certified Systems Expert in “Administration for IBM WebSphere Application Server.”

Stéphane Faure is a WebSphere Advisory IT Specialist with the Design Center for e-business on demand™ at the EMEA ATS Product and Solutions Support Center in Montpellier, France. He has 14 years of experience in the computing field. He holds a degree in Computing from Gustave Eiffel College, Bordeaux. His areas of expertise include S/390®, z/OS, VM, z/OS UNIX System Services, TCP/IP, IBM HTTP Server, WebSphere Distributed Platforms, and z/OS. He is IBM Certified as a WebSphere Specialist, Solutions Expert in CICS Web enablement, e-business Solution Designer, and e-business Solution Technologist.

Tom Hackett is a Senior Software Engineer in the United States. He has 23 years of experience in the software field. He holds a Bachelor of Arts degree in Philosophy from Drew University, a Master of Divinity degree from Wesley Theological Seminary, and a Master of Science degree in Computer and Information Science from Syracuse University. His areas of expertise include software testing, software patents, and WebSphere Application Server.

Andrew J. Hoyt is a Software Engineer in the United States. He holds a Bachelor of Science degree in Computer Science from Marist College in Poughkeepsie, New York. He has worked on the WebSphere Product Validation team during his two years at IBM.



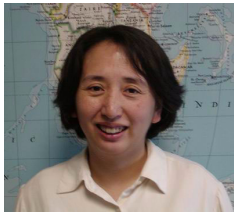
Patrick Kappeler was a computer specialist in the French Air Force and joined IBM in 1970 as a Diagnostic Programs Designer. He has held several specialist and management positions as well as international assignments, all dealing with S/390 technical support. He joined the EMEA S/390 New Technology Center in Montpellier in 1996, where he now provides consulting and presales technical support in e-business security.

Michael Kearney joined IBM in 1978 and has been working with security since 1990. During his time as a Security Specialist, he has worked with RACF, cryptographic services on zSeries, and WebSphere security. He earned his CISSP in 1997. Mike works at the Washington Systems Center in Gaithersburg, MD, where he provides technical support to IBMers and customers.

Sébastien Llaurency is a Java Developer and WebSphere Specialist in France at the Design Center for e-Transaction Processing. He has two years of experience in prototype development for EMEA customers. His areas of expertise include Java development tools and WebSphere Application Server.

Edward McCarthy currently works in the e-business Enablement Services team for IBM Global Services Australia. The team is responsible for covering all aspects of WebSphere Application Server and WebSphere MQ across all platforms. For the last two years, he has specialized in supporting the WebSphere family of products. Previously, he worked as a senior CICS and MQSeries® systems programmer for more than eight years with a large IBM customer. He also participated in writing the redbook *IBM WebSPhere Everyplace Server, Enterprise Wireless Applications*, SG24-6519.

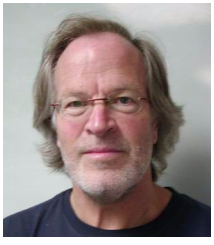
Hong Min is a Software Engineer at the IBM Design Center for e-business on demand, Poughkeepsie, U.S. She has more than six years of experience in e-business technical support, customer application design, and prototyping. She holds a Master of Science degree in Computer Science from Drexel University, Philadelphia, PA. Her areas of expertise include z/OS, Java, J2EE, and WebSphere.



Mark Nelson, CISSP, is a Senior Software Engineer with the z/OS Security Server Design and Development team. The past 15 of Mark's 21 years with IBM have been focused on RACF. Mark holds a Bachelor of Science degree in Computer Science from the Polytechnic University of New York and a Master's Degree in Information Systems from Marist College. He is a frequent speaker on RACF topics and has earned two "Top Gun" awards for the best session at the Vanguard Enterprise Security Expo, as well as several SHARE "Best Session" awards. Mark was also the recipient of the 1999 Vanguard "Chairman's Award" for his contributions to the information security industry.

Alain Roessle is an IT Architect for the IBM Design Center for e-business on demand, in Montpellier, France. Before joining IBM in 2000, he worked for a large distribution company for 20 years. His areas of expertise include CICS, DB2, and Parallel Sysplex®. For the last two years, he has worked on WebSphere security issues.

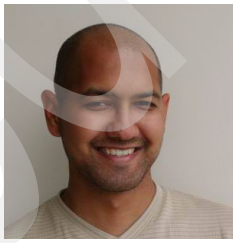
Kevin J. Senior currently works for IBM Software Group in Italy. He started working at IBM 23 years ago as an operator of the RETAIN® application and the IBM international network, and then moved into systems programming. He supported IMS™, DB2, and CICS for IBM internal and outsourced customer systems. He next worked as a technical specialist/architect for IBM Global Services, assigned to the IBM Insurance Business and working with a large CICS/COBOL/DB2 application called HUON. He later joined the AIM Services group in the IBM Hursley laboratory, providing worldwide support on CICS and specializing in connectivity with WebSphere for z/OS. He moved to Italy three years ago and now works for Technical Pre-Sales Support, specializing in CICS and WebSphere for z/OS. Last year, he created education material based on the *z/OS WebSphere and J2EE Security Handbook*, SG24-6846-01, redbook.



Karl-Erik Stenfors is a Senior IT Specialist in the Product and Solutions Support Centre (PSSC) in Montpellier, France. He has more than 30 years of experience in the large systems field, as a systems programmer and consultant with IBM customers, and for the last 17 years as an IBMer. His areas of expertise include zSeries hardware and operating systems. He teaches at numerous IBM user group and IBM internal conferences. Before joining the PSSC in 2000, he was an employee of IBM, Norway, where he held two

international assignments, both in the large systems field. He has participated in previous ITSO residencies.

Theresa Tai is in WebSphere on zSeries marketing technical support in Poughkeepsie, New York. She has over four years of experience in supporting WebSphere and Java on z/OS and OS/390. She holds a degree in Computer Science from Bernard Baruch college in New York City. She has a background in JES3 and JES2 Design, Development, and Service. She is a co-author of the Redbooks *WebSphere for z/OS V4 Problem Determination*, SG24-6880, in 2002 and *WebSphere Portal on z/OS*, SG24-6992 in 2003.



Dinkar Tiwari is an Advisory e-business Specialist in the U.S. He has more than six years of experience in the IT field and has worked at IBM for four years, focusing on e-business consulting. His areas of expertise include application and infrastructure architecture/design, Tivoli Access Manager for e-business, and WebSphere for z/OS. Previously, he was involved in system and storage sales. He holds a Bachelor's degree in Computer Science and is currently pursuing a Master's degree in Computer Information Systems.

Jason Williams is an Information Planner and Developer for the IBM Poughkeepsie eServer Information Solutions (eIS) organization. Since April 2003, Jason has worked on WebSphere Application Server security information, and his previous positions include z/OS Request for Announce (RFA) Information Developer and z/OS Security Server Information Planner. Jason also works on other assignments, ranging from serving as an eIS education coordinator to working with the Poughkeepsie Site Diversity Council (PSDC) career opportunity team on recruiting. He has a Bachelor of Arts degree in English and History from Binghamton University and is currently pursuing a Master of Science degree in Management from Rensselaer Polytechnic Institute (RPI).

Foulques de Valence is a WebSphere for z/OS Specialist with IBM Global Services. Currently based in Paris, France, he works at the French customer Technical Support Center with the WebSphere for z/OS team. In previous years, he provided customer support on Lotus® Domino® for OS/390 and UNIX System Services. Prior to IBM, Foulques served as the IT Manager of a small manufacturing company in the San Francisco bay area. He received his Master's degree in Computer Science and Engineering from Ensimag in France. He furthered his education at the State University of New York at Buffalo and at Stanford University in the U.S.

Julieta Bianchi is a Technical Advisor and Application's Architect for Citigroup. She has more than 10 years of experience developing and architecting large-scale applications. In the past few years, her concentration has been architecting and designing applications running on WebSphere on z/OS. Julieta has a Bachelor's degree in Electrical Engineering, and Master's and PhD degrees in Computer Engineering. She also teaches as an Engineering Adjunct Professor in Florida U.S.

Steve Allison is a Software Engineer in the zSeries WebSphere system test team in Poughkeepsie, where he has also had responsibilities for function testing and compliance testing for J2EE 1.2 and 1.3 during his three years with IBM. Previously, he worked on application development for two large international businesses. Steve holds Bachelor's degrees in Medical Technology and Electrical Engineering.

Michael Daubman is a Staff Software Engineer for the zSeries e-business Services team, where he specializes in enterprise security with a focus on RACF, PKI, and WebSphere security. Before his services role, Michael worked as a developer and function tester in the z/OS Security Server team. Michael holds a Bachelor's degree in Computer Science from RIT and an MBA from Marist College.

Tamas Vilaghy is a project leader at the International Technical Support Organization, Poughkeepsie Center. He leads redbook projects dealing with e-business on zSeries servers. Before joining the ITSO, he worked in the System Sales Unit and Global Services departments of IBM Hungary. Tamas spent two years in Poughkeepsie from 1998 to 2000 dealing with zSeries marketing and competitive analysis. From 1991 to 1998, he held technical, marketing, and sales positions dealing with zSeries. From 1979 to 1991, he dealt with system software installation, development, and education.

Many thanks to the following people for their contributions to this project:

Eve Berman, IBM Endicott

Karunakar Bojjireddy, IBM Raleigh

Box Boxall, IBM Hursley

Hyen V. Chung, IBM Austin

Richard Conway, IBM Poughkeepsie

Rajeev Gandhi, IBM Austin

Tim Hahn, IBM Raleigh

Thomas Hanicke, IBM Germany

Elizabeth Hatfield, IBM Endicott

John Hutchinson, IBM Washington

Chris Johnson, IBM Rochester

Kim Johnson, IBM Poughkeepsie

John C. (Jack) Jones, IBM Atlanta

Ivan Joslin, IBM Poughkeepsie

Jayanthi Krishnamurthy, Radiant Systems

Andreas Landenberger, IBM Germany

Sean Lee, IBM New York

Victoria Mara, IBM Poughkeepsie

Ivan Milman, IBM Austin

Ian Mitchell, IBM Hursley

Carla Molenda, IBM Endicott

Gino A. Piccolino, IBM Endicott

Gary Puchkoff, IBM Poughkeepsie

Birgit Roehm, ITSO Raleigh

Alfred Schwab, Editor, ITSO Poughkeepsie

James Sweeny, IBM Poughkeepsie

Matthew Sykes, IBM Poughkeepsie

Nancy Trent, IBM Poughkeepsie

Darryl Van Nort, IBM Chicago

Christopher Vignola, IBM Poughkeepsie

Phil Wakelin, IBM Hursley

Bruce Wells, IBM Poughkeepsie

Nigel Williams, IBM Montpellier

Young-Jun Yoon, IBM Poughkeepsie

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Who should read this book

Application programmers. This redbook will give you an understanding of the authentication and authorization facilities available through the J2EE programming model and the programmatic interfaces to those facilities that are available to application programmers. Understanding this information will help you to write application code that is platform neutral, while taking advantage of the highest levels of security available.

z/OS and WebSphere and security administrators. This redbook gives you the information you need to know in order to understand how J2EE applications fit within your security practices. It will also help you to choose guidelines and processes for communicating security requirements to application developers.

Application and network architects. This redbook discusses many options for performing security functions, particularly authentication, that will keep your z/OS system secure. It gives you the information that you will need in order to decide how and where to register users, what kind of credentials you should require, and how work should be identified within your z/OS system.

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

New and revised cryptographic information

Chapter 9, “Using cryptographic services” on page 239 has been updated with expanded SSL background information and new information specific to IBM @server® zSeries® 2084 and 2086 cryptography support. A new section has been added, 9.8.1, “Certificates in WebSphere and RACF” on page 268, explaining how IBM® WebSphere® Application Server can use certificates stored in RACF®.

Securing the file system

Three new sections, 10.11, “HFS security” on page 323, 10.12, “HFS ACLs” on page 324, and 10.13, “zFS security” on page 331 were added to assist you in protecting application and configuration files residing in an IBM z/OS® file system. In addition, 15.2.2, “Securing configuration files” on page 475 further explains configuration file security.

Security domains

Section 12.2.3, “SAF-based WebSphere authorization concepts” on page 351 now contains information about security domains, which can help separate the administration of cells when a local OS registry is used.

Java 2 security

The security investigation application, SWIPE, has been enhanced to support Java™ 2 security. We document this in 7.5.2, “SWIPE and Java 2 security” on page 159. This section will be useful to anyone deploying applications into a server protected with Java 2 security.

Enhanced support for Tivoli Access Manager

A new chapter, Chapter 14, “IBM Tivoli Access Manager and WebSphere Application Server integration” on page 417, details the use of IBM Tivoli® Access Manager for both authentication and authorization within WebSphere Application Server.

Other enhancements

Chapter 2, “Security design” on page 23, has a new section, 2.5, “Topological view of security” on page 32, giving a high-level view of security design choices. Cross references to other sections of the book have been updated and expanded.

Chapter 5, “WebSphere application migration security aspects” on page 91 has been updated with current information about the support of SyncToOSThread, removing migration actions that were necessary when this support was unavailable.

In 10.10.1, “Security Customization panel settings” on page 309, we clarify setting local OS security registry values for authorization and delegation.

Password management is discussed in a new section, 16.2.6, “Password management” on page 529.

We have also made minor additions and corrections throughout the book.

Information removed or relocated

Finally, in order to avoid duplication and reduce the size of this book, the chapters “Enterprise Information Systems security” (Chapter 18 in the previous edition) and “Java Messaging Services (JMS) security” (Chapter 19 in the previous edition) have been removed from this edition. The material is being migrated to *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064.

Part 1

Introduction to WebSphere and J2EE security

In this part, we introduce the security features of WebSphere Application Server for z/OS Version 5 and discuss how you can use this book to help you with your overall security design. This part also provides an overview of J2EE security concepts and security from the perspective of WebSphere applications, as well as application migration to IBM WebSphere Application Server for z/OS Version 5.

1

WebSphere Application Server V5 security overview

This chapter introduces you to new infrastructure concepts and terminology introduced with WebSphere Application Server V5.0. It also provides the high-level overview of the new security features and functions supported by IBM WebSphere Application Server 5.0 for z/OS and OS/390. The WebSphere 5.0 security is based on the J2EE 1.3 specification with the intend to promote consistency, commonality, and portability across all WebSphere Application Server platforms.

The security options for WebSphere Application Server for z/OS Version 5 have been enhanced to fully support J2EE 1.3 specification and requirements. There are some significant functional enhancements and new features. For those who have invested significantly in WebSphere Application Server for z/OS and OS/390 V4.0.1 would be pleased to know that WebSphere 5.0 maintains compatibility with 4.01. WebSphere Application Server for z/OS Version 5 will be welcomed by those who would like more consistency between WebSphere distributed and WebSphere on z/OS. We will be providing the same and more security features as the distributed platform from WebSphere Application Server for z/OS Version 5 and later.

1.1 WebSphere Application Server for z/OS Version 5 infrastructure overview and terminology

Implementing security in a WebSphere environment requires that your security administration, systems programming, and WebSphere application development staffs work closely together. Terminology can sometimes become a problem when these disparate groups work together. In this section, we present an overview of WebSphere infrastructure and terminology.

WebSphere Application Server

The server instance consists of a controller region and a servant region. In WebSphere Application Server V5.0, the Java virtual machines (JVM) reside in the controller and servant address spaces. A controller region and one or more servant regions are called a server. The controller and servant structure design makes it possible to start multiple servants by WLM, based on the workload queued by the controller region. Figure 1-1 depicts the infrastructure components, sampling a multiple cell Parallel Sysplex implementation.

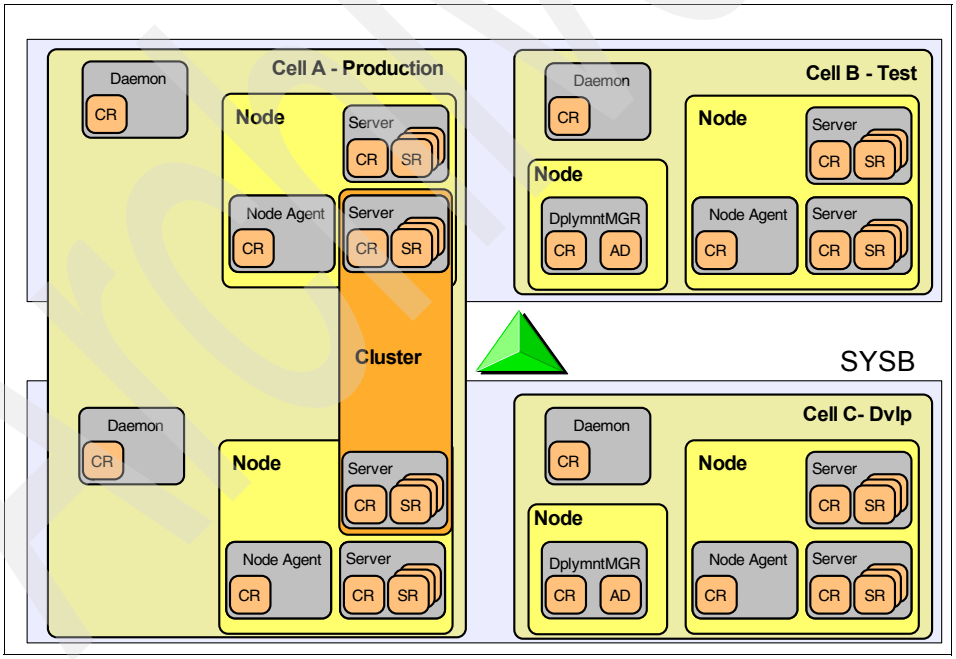


Figure 1-1 WebSphere infrastructure components in a Parallel Sysplex

Base Application Server

The base Application Server node is the easiest and simplest operating structure in WebSphere for z/OS V5. It consists of one or more server instances, a daemon server, one node and one cell. The definitions and configuration files are all kept in the HFS directory structure for the created base Application Server. The application server relies on a set of XML/XMI files for its configuration repository. There are also an MBean server, an administration application, and the name server in each application server. The application server administration is based on two tools: The Web browser-based graphical user interface called administrative console and the non-graphical command line scripting client.

Daemon server

The daemon server is a special server with only one controller region. It is a supporting daemon that is responsible for accepting and initiating application requests. For that, it has to know which servers are active and know all the applications in them. The daemon resolves an indirect IOR, which means it returns an IOR to the client that then can be used to access the application in the selected server, based on the z/OS Workload Manager (WLM) input for the best choice. There is only one daemon per cell per system needed in the architecture of WebSphere Application Server V5.0.

Node

A node is a collection of servers on a given system or LPAR. The node name in a cell has to be unique. The purpose of a node becomes clearer when the configuration includes several systems or LPARs, and the whole environment is managed from a central administrative console. This would assume that we also have a node agent, which is a specialized server that receives commands from the central administrator and issues those commands against the servers in its node. The base Application Server does not have such a node agent, because the administrative Web interface is running within one of its own servers.

Cell

A cell is a collection of nodes that makes up an administrative domain or boundary. A cell name must be unique and cannot be extended beyond the sysplex. The simplest example of this is the base application node. The administrative domain extends only to itself, which makes it very small. The cell can be extended to exist across multiple systems or LPARs in a sysplex and consist of many nodes, which makes the administrative domain very large. Even a base Application Server node can contain multiple servers, which also expands the administrative domain. Whether it is a very simple or a very large and more complicated configuration, an administrative domain is always required.

1.2 WebSphere Application Server V5 security features

This section provides a brief introduction to the new functions and features supported in WebSphere Application Server for z/OS Version 5 in three separate tables. First are the J2EE 1.3 compliance features, followed by WebSphere Network Deployment compliance, and WebSphere family security configuration items. The security functions and features are consistent between z/OS and the distributed platform, although there are some exceptions unique to the z/OS platform.

1.3 J2EE 1.3 compliance features

WebSphere Application Server V5 fully supports the J2EE specification and the security requirements under the specification.

Table 1-1 J2EE 1.3 compliance features

J2EE 1.3 compliance features	Comments
Java 2 security	Provides finer grain access control, and policy-based security that defines permissions support for enterprise applications. You can: ► Check if a piece of code has permission to access a system resource. ► Assign principal name (doAs). doPrivileged() provides limited checking in the call stack from check permission to the first doPrivileged(). Supports dynamic policy (was.policy in META-INF)
Servlet RunAs functionality	Provides the same RunAs functionality as EJB RunAs supported in V4.01. The same function applies to both servlets and JSPs.
Servlet filters	In addition to the existing form-based authentication processing, a new option of servlet filters can be used for additional authentication or processing. Both form-based login and servlet filters are supported by any servlet Version 2.3 specification compliant Web container. The form-based login servlet performs the authentication, and servlet filters perform additional authentication, auditing, or logging information. The servlet filters are invoked either before or after the login actions (CustomLoginServlet).
Common Security Interoperability Version 2 (CSIv2) security protocol	This provides RMI/IIOP interoperability between different application servers (bean to bean). Other security protocols are SAS for distributed and zSAS for z/OS. zSAS works with a SAF-based registry (RACF) on z/OS.

J2EE 1.3 compliance features	Comments
JAAS programming model	This is an extension to Java 2 security and an authentication mechanism (authorization is principal-based doAs). JAAS provides the flexibility in authentication and authorization programmatically within your application. WebSphere itself uses JAAS login modules. z/OS implementation adds support for principal-based security to authenticate SAF users. It provides code-based and user-based Java authorization on grants defined in the java.policy file. SAF-backed authentication is by Subject.doAs (authorization within doAs loop, and change the identity of the underlying z/OS thread within doAs loop). JAAS replaces CORBA programmatic login APIs, as WebSphere Application Server for z/OS and OS/390 does not support CORBA.
IBMJCE	IBMJCE (IBM Java Cryptography Extension for z/OS) replaces IBMJCA (Java Cryptography Architecture) and adds significant cryptographic capabilities to Java Cryptographic Architecture. Support is provided for digital signatures (RSA and DSA), hashing (SHAI MD2, MD5), keystore (symmetric and asymmetric keys protected by triple DES); and asymmetric algorithms use a public or private key pair.
JSSE	Java Security Socket Extension (JSSE) supports SSL and TSL algorithm types. It enables easier socket creation using encapsulated factories and extended capabilities for developers that need unique capabilities. Both JSSE and System SSL can use a RACF keyring or a keystore in the HFS file system.

1.4 WebSphere Network Deployment family compliance features at interface layer

The key to make WebSphere Network Deployment family-compliant is deploying applications from Network Deployment on z/OS seamlessly.

Table 1-2 WebSphere Network Deployment family compliance at interface layer

WebSphere Network Deployment compliance	Comments
wsadmin common scripting	Can be used to manage and deploy applications. Scripts are executed under the new tool wsadmin, which supports Java Command Language (JACL), which is a Java implementation of TCL. TCL (pronounced *tickle*) stands for “tool command language.”
WebSphere extensions to JAAS	This replaces CORBA. JAAS authentication extensions support Callback and the WSSubject.doAs class. The new getCallerSubject() and getRunAsSubject() APIs are provided.
Trust association interceptor	TAI support was delivered through the maintenance stream on WebSphere V4.01.
User registry interface, custom user registry	Pluggable User Registry Interface is used by both z/OS and the distributed platform.
JMX Admin Service	RMI/IIOP JMS connector and SOAP/HTTPs Java Management Extensions (JMX) connector.
J2EE Connector architecture (J2CA) connectors with user ID/password credentials and principal mapping	The enterprise information system (EIS) security for CICS, IMS, and DB2 for J2EE-based applications includes integration with EISs. An application server and an EIS collaborate to ensure the proper authentication of a resource principal, which establishes a connection to an underlying EIS. Enabled as part of global security, if global security is OFF, it will run with the default thread identity. It is not always appropriate to run with the default thread identity (server identity). J2CA supports user ID and password-based authentication and kerbv5-based (Kerberos V5) authentication.
WebSphere APIs	WebSphere APIs provide support for WSSubject, WSCredential, TAI, and other APIs.

1.5 Support of WebSphere family security configurations

The key here is to provide consistency and commonality between WebSphere on distributed platforms and the z/OS implementation.

Table 1-3 WebSphere family security configuration

WebSphere family security configuration	Comments
Common Configuration Model, using an administrative console	This is a common WebSphere Common Configuration Model (WCCM) with distributed platforms with z/OS extensions. System configuration is managed by the administrative console or security center. Configuration data is stored in the HFS. wsadmin scripting support is available.
Global security configuration	There is a global security switch; the default is global security off, and when it is switched on it automatically turns on Java 2 security as well. When global security is switched off or disabled, it disables all WebSphere-controlled securities. It does not disable Java 2 security and it does not disable all security definitions. There are still user IDs defined for the WebSphere process, and users and groups for administrators and STC profiles. Nothing can run without an identity on z/OS. security.xml stores the security definitions. The security administration applies to the cell level for WebSphere Network Deployment. Registration and authentication mechanism configuration is at the cell level.
security.xml file	The security.xml file stores most of the security definitions. The security administration applies to the cell level for WebSphere Network Deployment. Registration and authentication mechanisms are configured at the cell level, and many server specifics can be overridden.
Common application deployment metadata	WebSphere bindings for authorization and RunAs.
RoleBased authorizer for naming and administration	SAF EJBROLE authorization support is provided and the non-SAF registry support for RoleBased authorizer is disabled. NamingServer uses JMX Mbeans. Using IIOP for authentication from Java client or server as client is possible. Protecting CosNaming subsystem with security roles (CosNamingRead|Write|Create|Delete) is also possible.
LTPA authentication	LTPA signon domains can include distributed servers.
LDAP user registration implementation	LDAP registry can be non-z/OS and off-platform.

1.6 J2EE 1.3-compliant security enhancements

WebSphere Application Server V5 is J2EE-compliant, supporting all the Java 2 APIs that are common across all implementations under J2EE 1.3 specifications. EJB 2.0 and Servlet 2.3 specifications include new security enhancements. WebSphere 5.0 is J2EE 1.3 certified using the Compliance Test Suite (CTS).

1.6.1 Java 2 security

Java 2 security was introduced in SDK 1.3. It is fully supported in WebSphere Application Server V5. It provides policy-driven protections to system resources such as file I/O, sockets connection, program threads, and class loading on whether a piece of code can access the resource using policy files. The policy is declarative (not programmatic). It also provides fine-grained access control, and a policy defined by a set of permissions available for code from various signers or locations. It allows you to organize code into individual protection domains using code source location and a set of digital signatures. Each loaded class belongs to one protection domain with the same set of permissions.

What is a protection domain? First, a domain can be scoped by the set of objects that are currently directly accessible by a principal, where a principal is an entity to which permissions are granted. A domain conceptually encloses a set of classes whose instances are granted the same set of permissions. Protection domains are determined by the policy currently in effect. The Java application environment maintains a mapping from code (classes and instances) to their protection domains and permissions.

WebSphere extensions to Java 2 security

The security features that are unique to WebSphere are applicable to both distributed platforms and z/OS. The dynamic policy permissions are based on resource types rather than code base location. The default permissions are defined for resources that are specified in template policy files, and the filter files disable policy.

Enterprise application policy file

The default was.policy file created during application installation is the application policy file, if there is not one existing already, <your application>.ear/META-INF/was.policy.

Editing policy files

Consider the following items:

- Policy tool: Part of SDK, java/jre/bin/policytool, preferred method versus editing manually.
- WebSphere Studio Application Developer or Application Assembly Tool (AAT): Insert was.policy in the META-INF directory of the .ear file (also does syntax validation).
- Network Deployment environment: Use wsadmin to extract the file for modification.

1.6.2 J2EE role-based authorization enhancements

In WebSphere Application Server for z/OS and OS/390 V4.0.1 you could use a role-based security model programmatically or administratively in your applications.

In addition to this you now have the ability to perform a RunAs and change the identity of the user at runtime. The security roles can be defined to restrict access either to a specific method in your EJB or within the methods themselves. This security model offers flexibility for defining roles and does not have to match roles in your production environment. Security role references allow for bindings to be made at installation time to references in the application to the actual roles in the production server.

With the EJB 2.0 specification, you can specify a specific role to secure a method, avoid checking for security, or exclude the method from being called. For compliance with the specification you must specify at least one of these settings.

Roles are used to control access to J2EE resources like JSPs, servlets, and EJBs. The name of the roles authorized to access the J2EE resources are specified in the web.xml file of the application. In WebSphere Application Server for z/OS Version 5, the authorization information, such as which users or groups of users have which roles, can be specified in a number of ways.

There are two ways to perform role-based authorization checks in WebSphere on z/OS:

- Using SAF EJBROLE profiles: Using this mechanism, authorization is placed in the control of the security administrator. The security administrator defines an EJBROLE profile for each J2EE role defined in an application. A user is considered to be “in role” if the user is given READ permission to that EJBROLE profile. This support is only provided if the local OS registry is configured. This support was provided in WebSphere Application Server for z/OS V4.01. It is unique to WebSphere for z/OS.

- Using WebSphere bindings: The application deployer does user-to-role mappings by selecting users and groups in a registry and indicating that they are in that role for that application. The authorization is actually built into the .ear file. This can be done using WebSphere Studio Application Developer or AAT. The selection for how role-to-user mapping is done is at a cell level. It is dependent on the com.ibm.SAF.Authorization property. This can be set either to customization dialog or the administrative console.

Note: WebSphere Application Server for z/OS Version 5 SAF authorization does not support authorization subjects such as Everyone and AllAuthenticated, which are commonly used on WebSphere distributed platforms. This is because all applications on OS/390 and z/OS must run under a user ID. However, there are equivalent ways to provide this support: By giving UACC READ to an EJBROLE profile and having the unauthenticated user ID set to RESTRICTED.

1.6.3 WebSphere Application Server V5 and JAAS

Java Authentication and Authorization Service (JAAS) was introduced with SDK 1.3.0, and extends from Java 2 code source-based security model. z/OS implementation adds support for Principal-based (user ID) security. It is used to authenticate any WebSphere user. Grants are normally in the java.policy file but they can also be in a user-defined policy file. In WebSphere 5.0, we authenticate using whatever user registry is configured. An SAF-based registry is only used if you are configuring a Local OS registry. The SAF interfaces use an External Security Manager (ESM) such as IBM Resource Access Control Facility (RACF).

JAAS is a programmatic interface used to establish identity and perform authorization. It can be used for authentication and for authorization of users to ensure they have the access permissions required to do the actions performed.

With Java 2 security, you only have the ability to distinguish running code based on where it came from and who signed it. JAAS extends this support by providing the ability to provide access based on the identity of the user actually running the code.

The Pluggable Authentication Module (PAM) framework allows for authentication methods to be implemented without changing any of the login services, reducing the effort in using new authentication technologies. With PAM, the applications remain independent from underlying authentication services.

JAAS is the strategic programming model for WebSphere. It replaces the CORBA programmatic login APIs. The JAAS logon configuration can be configured through the administrative console or wsadmin.

Note: Do not remove or delete the predefined JAAS login configurations ClientContainer, WSLogin and DefaultPrincipalMapping. It would cause application failures.

The DefaultPrincipalMapping defines a special LoginModule that is typically used by J2EE Connector architecture (J2CA) to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back-end enterprise information system (EIS).

1.6.4 Java 2 security, J2EE security, and JAAS feature comparison

J2EE security and Java 2 security are relatively new to WebSphere on z/OS. Table 1-4 is intended to help readers to differentiate the various Java security features supported in the Java 2 framework. The main difference between Java 2 security, JAAS, and J2EE security is how security is enforced:

- Java 2 security is enforced by the JVM and server with the permissions specified in the policy files.
- JAAS security is enforced programmatically from within an application.
- J2EE security is enforced by the J2EE server or programmatically from an application.

Table 1-4 Java security feature comparison

Security feature	Purpose	Details
Java 2 security	Code-based access control	Enforce access control based on the location of the code and who signed it; not on the principal. Java 2 security is defined in policy files and enforced at runtime.
J2EE security	Identification, authorization, and identity propagation	Role-based security for JSPs, servlets and EJBs is supported. The identity can be changed based on method delegation attributes. J2EE security is defined in configuration settings or within application code and enforced by runtime or programmatically.
JAAS security	Authentication and authorization	Enforce access control based on the current principal/subject. JAAS security is defined in application code and enforced programmatically.

There are still some drawbacks to JAAS and you should be cautious in implementing it, as follows:

- JAAS still only covers authentication and authorization. It does not cover other aspects of security.
- The services that JAAS provides will only work if they are called. There is no method of ensuring that application programs are calling these services when required. To remain secure, applications should go through security code reviews.
- If not done from an enterprise perspective, each application can implement its own security policy. This would create an anarchy-based security model that will be difficult to clean up.

If you want services that the framework does not provide, you have to explicitly code them.

1.6.5 z/OS Java security components

The Java 2 security components are available through IBM Developer Kit for OS/390, and Java 2 Technology Edition at SDK 1.3.1. They are available using SMP/E or pax from IBM. We added a large set of security capabilities to the Java 2 framework on z/OS. The (see Table 1-5) are easy to use; they include the z/OS exclusive hardware assist cryptography capability in SDK 1.3.1, which enhances both security and performance.

Table 1-5 Java security components

Components	Function/features
JAAS	Java Authentication and Authorization Service
JCE	Java Cryptographic Extension using Common Cryptographic Architecture (CCA)
JCE4758	IBM hardware cryptographic device on z/OS
JSSE	Java Secure Sockets Extension
CertPath	Certificate generation and path validation
SAF Interface	z/OS security services in Java implement the SAF interface, JNI, to call SAF/RACF

1.6.6 JSSE security

Java Secure Sockets Extension (JSSE) is a 100% pure Java implementation common across all IBM platforms. IBMJSSE implements SSL 3.0 and TLS 1.0 algorithm types as Java 2 standard extensions. It provides the socket factories

that allow for ease of programming (if you want to simply take the defaults). JSSE can also help to avoid creating security holes for those who do not know all the in's and out's of security and cryptography. IBMJSSE is the preferred SSL and TLS provider for Java applications on z/OS and should be used in place of SystemSSL for Java applications to avoid overhead converting to the C-based service Java Native Interface (JNI).

1.6.7 CSIv2 security protocol

Common Secure Interoperability, CSIv2, is an IIOP-based security protocol, part of J2EE 1.3 requirements, that enable multivendor interoperability; Compliance Level 0 must be implemented. Defined by Object Management Group (OMG), this standard, open, secure interoperability framework is common across J2EE servers. CSIv2 defines the Security Attribute Service (SAS) that enables interoperable authentication, delegation, and privileges. CSIv2 is active only when global security is enabled.

The CSIv2 protocol is intended to be used in environments using transport layer security, such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

The CSIv2 Security Attribute Service is divided into two layers:

- The *authentication layer* is used to perform client authentication where sufficient authentication could not be accomplished in the transport.
- The *attribute layer* can be used by a client to deliver security attributes, such as identity (identity assertion) and privilege, to targets where they can be applied in access control decisions.

CSIv2 (Compliance Level 0) features

The CSIv2 features include:

- This is the successor to IBM Secure Association Service (SAS or zSAS), where CSIv2 is the strategic protocol. IIOP authentication protocols process inbound requests from Java clients or J2EE servers.
- Administration of CSIv2 is integrated with other aspects of WebSphere Application Server administration using the administrative console or the wsadmin command facility.

1.6.8 Pluggable authentication security

A *pluggable* is pretty much the same as the user exits provided with the z/OS and OS/390 base operating systems. Within the pluggable or exit we provide a set of standards that you can code to. In WebSphere Application Server for z/OS and OS/390 V4.0.1, SAF is required for authentication services. With

WebSphere Application Server for z/OS V5, you can still use SAF as the user registry, but now you can use LDAP or a custom user registry for authentication services. The Local OS registry is an SAF-based implementation provided by the product.

The z/OS-only pluggable authentication features include SAF and the Integrated Cryptographic Service Facility (ICSF) login module. The ICSF is an authentication mechanism. The user registry is a key plug point, because it enables you to write your own registry interfaces. So is trust association interceptor (TAI), which helps to extract authenticated identities if you have a front-end authentication server.

1.6.9 Security configuration in z/OS and OS/390

The security configuration is managed by the administrative console. The security data is stored in the HFS. The configuration model is consistent with WebSphere Common Configuration Model (WCCM) and has z/OS extensions. The wsadmin scripting is also available. Most of the security definitions are done under the administrative console, and key data is kept in security.xml. The Web container-specific security definitions are defined in the Web container definition, webcontainer.conf.

Security configuration rules

The security configuration scope for WebSphere Network Deployment is imposed at the cell level for registries and authentication mechanisms. All the properties can be set at either the server level or cell level, generally with server override.

All the security settings for your WebSphere servers are configured at the administrative console. Most changes in the global security setup require a restart of the application server.

Administrative console

The administrative console is a Web application and can start and stop servers as well as administer security. All security settings for the WebSphere server are configured in the administrative console.

WebSphere administrative security

As part of WebSphere Network Deployment compliance, WebSphere administrative security provides granularity of access control based on four security roles, which are:

- Monitor: Has read only access to observe system states and configuration data but cannot make changes.

- Configurator: Has monitor privilege plus the ability to change the WebSphere Application Server configuration.
- Operator: Is a monitor that can initiate runtime state changes (for example, start/stop).
- Administrator: Has a role as the operator plus configuration privilege and the permission required to access sensitive data including server password, LTPA password and keys, and other data. This is similar to the configurator role.

Note that these administrative security roles apply to the entire cell level.

For administrative security, you can assign individual users or groups to specific roles that might map existing groups in your enterprise to the corresponding level of administration in your WebSphere cell. The configuration of users and groups is done through the administrative console.

Most security operations such as turning on security, and selecting authentication mechanisms and user registries, require a restart of the application servers. After security is enabled, assigning users and groups roles does not require a restart.

1.6.10 Enabling global security

First, global security is disabled when WebSphere is installed in order to simplify configuration tasks. Java 2 security is enabled, by default, when global security is turned ON. Enabling Java 2 security is optional; you can manually disable Java 2 security while keeping Global security ON. The scope of security applies to the cell level, if any application server needs security. This should only apply to "trusted" applications that are installed on that application server. As mentioned earlier, the initial configuration is security disabled.

Note: Keeping Java 2 security disabled is probably a good idea when migrating from V4 to V5 to avoid application failures. Remember to enable and disable security in both Java 2 security and WebSphere Global security, and stay in synch to avoid unnecessary error conditions.

1.7 Comparisons between WebSphere Application Server for z/OS and OS/390 V4.0.1 and V5

The supported security features of WebSphere Application Server for z/OS and OS/390 V4.0.1 continue to be supported on WebSphere Application Server V5.0, with the exception of the DCE authentication security. This function is being

deprecated. IBM did not deliver SyncToOSThread on method descriptors in the initial V5.0 availability. This function was delivered through the service stream through PTF UQ88257 (APAR PQ81584) in the W502008 service level.

Table 1-6 shows a comparison of the security functions and features supported between IBM WebSphere Application Server for z/OS V5 and WebSphere Application Server for z/OS V4.01.

Table 1-6 Security functions and features comparison

Security functions and features	V4	V5	Comments
J2EE 1.3 compliant		X	J2EE 1.3 CTS (Sun's Compliance Test Suite).
zSAS IIOP authentication			
PassTicket within a sysplex	X	X	
SSL basic authentication	X	X	Requires z/OS SAF credentials.
Digital certs with SAF mapping	X	X	RACDCERT.
Kerberos with IBM protocol	X	X	Supports kerbv5 (Kerberos V5).
SAF identity assertion	X	X	Provides a way for one server to trust another server without authenticating again.
"Unauthenticated"	X	X	
Authorization and delegation	X	X	SAF authorization and delegation using EJBROLES used for servlets and EJB in V4, where WebSphere bindings for user-to-role and role-to-user mappings are V5 only.
z/OS Connector facilities using SAF Client or RunAs identity	X	X	In addition to common J2CA connectors, enable Sync-to-Thread function still supported. As of V5.02, this is now known as Connection Manager RunAs Identity Enabled.
J2EE 1.3 compliance features:			
Java 2 security		X	New on V5, can be manually disabled while keeping global security ON.
Servlet RunAs		X	Also JSP RunAs I.
Servlet filters		X	For additional authentication (form-based processing).
CSIv2 Compliance Level 0		X	Provide bean-to-bean interoperability.
JAAS		X	Fully supported on V5.

Security functions and features	V4	V5	Comments
WebSphere compliance API/SPI			
WebSphere extensions for JAAS		X	WSSubject class, Callback, getCallerSubject(,) and getRunAsSubject()
J2CA connectors with user ID/password credentials and principal mapping		X	New for V5.
Sync-to-Thread onMethod descriptor	X		Delivered in the service stream. See 1.7, “Comparisons between WebSphere Application Server for z/OS and OS/390 V4.0.1 and V5” on page 17.
Web container functions			
ICSF	X	X	
LTPA		X	
Custom user registry	X	X	
LDAP user registry		X	
TAI	X	X	Delivered in the maintenance stream in V4.01.
DCE authentication	X		Deprecated in V5.
Consistency and commonality between WebSphere Distributed and WebSphere Application Server for z/OS		X	Common functions and features with few exceptions that are unique to z/OS.

1.8 Key differences between WebSphere Application Server for z/OS and distributed platforms

From a security aspect, the WebSphere Application Server for z/OS and OS/390 V4.0.1 implementation is different from WebSphere for distributed platforms. The authentication and authorization protocols on z/OS are driven by SAF, which implies that WebSphere Application Server for z/OS V4 security functionality is influenced by the use of the SAF interface. The z/OS Security Server in protecting and securing business data and z/OS resources. Consequently, WebSphere Application Server for z/OS and OS/390 V4.0.1 might not easily interoperate with WebSphere on distributed platforms.

Additionally, security administration is different. This has completely changed as of WebSphere Application Server for z/OS V5, where there is a noticeable consistency in delivering security functions and features between WebSphere on z/OS and the distributed platforms.

1.8.1 Two types of SSL on z/OS

WebSphere security uses two different service providers for SSL. System SSL is a z/OS component and provides SSL services for application subsystems like the HTTP server, LDAP, and TN3270. System SSL is integrated with SAF and the cryptographic hardware on zSeries. Java Secure Sockets Extension (JSSE, also known as IBMJSSE) is a 100% Java SSL service new to WebSphere 5.0 and is similar to the distributed platform.

1.8.2 “Deprecated” V4 Advanced interfaces

“Deprecated” V4 Advanced interfaces are not documented in WebSphere Application Server for Distributed V5.0 documentation. These interfaces are still being supported in V5. However, they are not supported on z/OS, because z/OS does not support CORBA APIs.

These interfaces are:

- CORBA login helper
- Server Side Authenticator
- CORBA current usage

1.8.3 z/OS security properties

Like the distributed WebSphere V5 platforms, security properties in WebSphere for z/OS V5 can be managed and controlled from the administrative console or wsadmin command line utility. Some properties are set at the server level, while others are only set at the global cell. zSAS properties are stored in security.xml as general properties.

1.9 Summary

IBM WebSphere Application Server for z/OS Version 5 provides security infrastructure and mechanisms to secure sensitive resources for end-to-end enterprise security requirements. The new features of WebSphere Application Server for z/OS V5.0 security are based on J2EE 1.3 specifications for EJB 2.0 and Servlet 2.3 and include new security enhancements. Figure 1-2 on page 21 and Table 1-7 on page 21 are intended to serve as a reference index to the

descriptions of the new security functions and features, as well as implementation details and residency experiences.

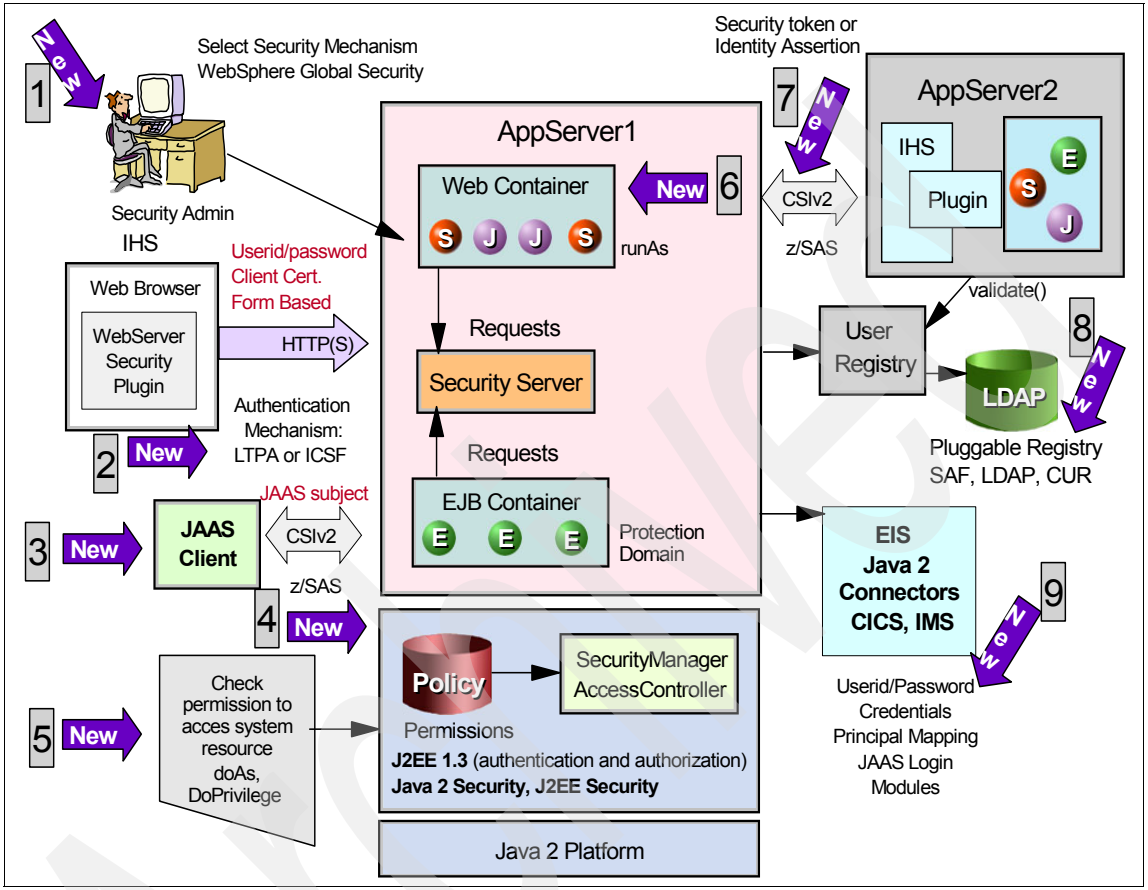


Figure 1-2 WebSphere Application Server for z/OS V5.0 new security features at a glance

Table 1-7 Reference index

Item	Security feature	Reference chapters
1	Administrative security	Chapter 15, “WebSphere administration and administrative security” on page 473
2	New authentication mechanism: LTPA or ICSF	Chapter 9, “Using cryptographic services” on page 239 “Lightweight Third Party Authentication (LTPA)” on page 62

Item	Security feature	Reference chapters
3	JAAS Client: CSIv2 and zSAS	Chapter 18, “EJB container security” on page 605 “Java Authentication and Authorization Service (JAAS)” on page 74 “Security Attribute Service (SAS)” on page 609 “CSIv2 authentication in WebSphere Application Server” on page 610 “Enabling CSIv2” on page 612 “CSIv2 solution scenarios” on page 621 “zSAS authentication methods” on page 630 “Enabling asserted identity security” on page 636 “zSAS asserted identity” on page 636 “Authentication in a sysplex” on page 638 “Client-to-server authentication” on page 639 “Server-to-server authentication” on page 640
4	Select security mechanism, and Java 2 Global security	“Security manager” on page 67 “Access controller” on page 68 “Java 2 security” on page 479 “Global security” on page 489
5	Java 2 security and J2EE security	Figure 4 on page 79 “Java 2 security” on page 66 “Java 2 security policy” on page 69 “JAAS for application security” on page 83
6	Web container security	“Role-based authorization” on page 47 “J2EE run-as” on page 53 Chapter 16, “Web container security” on page 501 “Web container authentication” on page 504 “Web container authorization” on page 541 “Web container identity delegation and propagation” on page 556
7	Server-to-server: CSIv2 Compliance Level 0	Chapter 18, “EJB container security” on page 605 “Identity assertion” on page 61 “zSAS asserted identity” on page 636 “Enabling asserted identity security” on page 636 “CSIv2 solution scenarios” on page 621 “Server-to-server authentication” on page 640
8	Pluggable registry Pluggable authorization security	“User registries” on page 46 Chapter 12, “Local operating system registries” on page 343 Chapter 13, “Remote registries” on page 361 “Custom user registry” on page 372 “Sample scenario: Implementing file-based custom user registry” on page 405
9	J2CA with user ID and password credentials and Principal Mapping	*WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064

2

Security design

This chapter describes the basic steps you should take to determine the security elements you need for your enterprise. These steps are important to consider when building an e-business enterprise using Java applications that cross platform boundaries. It is not meant to be a comprehensive list of things to do but a starting point from which you can build a security policy for your environment.

This chapter contains the following sections:

- Overview of security challenges
- Finding the right level of security for your enterprise
- Making some key decisions
- Finding the right balance for your application
- Topological view of security
- Summary

2.1 Overview of security challenges

Today's security needs have changed with the arrival of e-business technology. Not too long ago, the focus for security was only on operating systems and networks. With the growing demand for a secure Java-based e-business environment, the Java security model must be integrated with the security of the underlying operating system and its security services. This has further complicated corporate security challenges.

In today's business environment, some enterprises are aiming at bullet-proof security solutions, while some are settling for the minimal security that they think is good enough for their environment. This is based on their risk tolerance level and understanding of the cost of compromised security.

Now the question is, how much security do you really need, at what cost, and where is the balance? It is known that one of the biggest challenges with security is managing it. You need to think about monitoring the effectiveness of the current security policies and processes, managing and administering security, real time damage control, and resolving security exposures as part of the recovery process. You have to learn how to balance security with everything else, such as technical feasibility, cost effectiveness, performance, and protecting your environment and processes without sacrificing system availability or productivity.

2.1.1 Assessing and managing security risks

There is no one single generic security policy applicable to all enterprises that guarantees near flawless security. Each enterprise must:

- Determine the appropriate amount of risk for the information that it disseminates.
- Determine what is considered an appropriate cost for the risk it is willing to take: the expense of the security infrastructure and the cost to the enterprise if sensitive information is exposed.
- Consider maintaining levels of service to its customer base.

All of these factors are important when determining the right balance of security costs and their associated risks.

Security is the art of managing risk. A good security person must balance the cost of unauthorized use of equipment and data with the cost of securing that system and data.

2.1.2 Evolving with emerging technologies and trends

A typical business environment already has security measures in place and has implemented various security functions and features selected at some point. One of the major challenges for corporate security today is the integration of diverse processes and information. Consider how the new emerging technologies would work with, or be interoperable with, the existing security structure. On rare occasions you see a customer that has the luxury to architect a security environment from scratch and has the flexibility to implement any, if not all, security functions and features they could possibly need. But most must work to pick and choose what they believe to be the best security features to address the end-to-end security requirements not only for their enterprise but also their customers, business partners, and suppliers.

Because security is a continuous process, integrating and transforming your enterprise towards the new emerging technologies and trends becomes a continuous challenge (and is always an important element for all Java-based e-business environments).

WebSphere Application Server for z/OS Version 5 conformance to the new Java 2 and J2EE security features, such as protection domains, security policy, privileged code, and runtime access controls brings new security challenges to z/OS security professionals. Java is expected to run in a user environment but must be able to do things that have traditionally been restricted on the z/OS platform. The J2EE model starts to relegate some of those tasks to the container but still provides the application developer with the ability to add security business logic to prevent and mitigate security risks.

2.2 Finding the right level of security for your enterprise

Where do you start when choosing the right security and defining security policy guidelines for your environment? It is common practice that you first get the right people involved in the process. This includes Web masters, IT architects (networking, system, and security), application developers, deployers, and security administrators.

The rule of thumb is to start with your current infrastructure. Evaluate security elements from each of the layers (see Figure 2-1 on page 26) and determine what to update and where to introduce new security elements for your environment. You also need to determine what is critical, and to identify and prioritize plans to provide an effective and efficient security policy. At the same time, bear in mind the importance of maintaining the right balance between security and other competing priorities such as technical feasibility, performance, and the overall cost of maintaining security.

2.2.1 Evaluate security elements in each layer

Review and validate security elements in each layer.

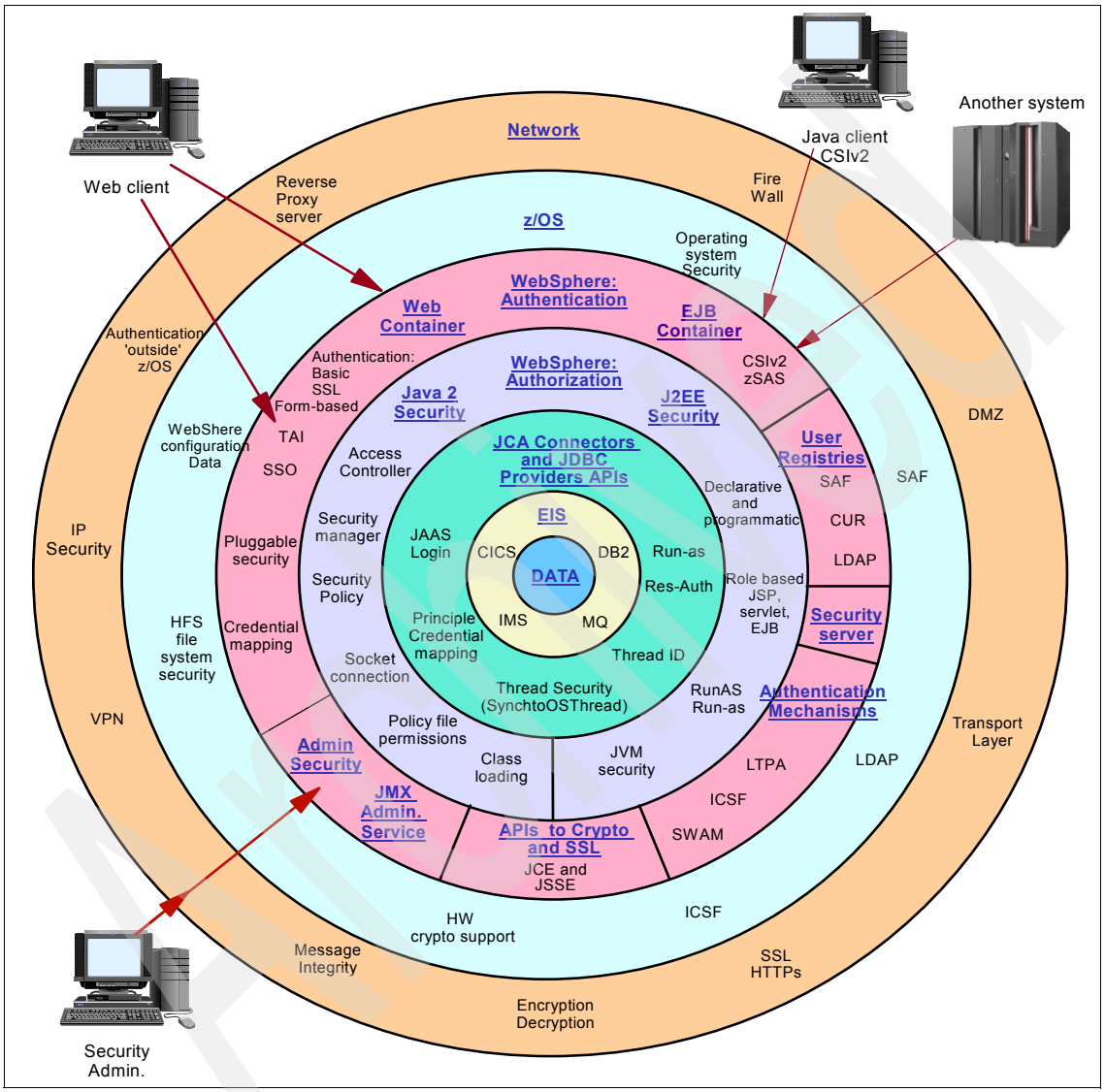


Figure 2-1 Security infrastructure layer chart

The following items provide a glimpse of how to proceed:

- Network infrastructure layer:
  Use firewall technology and proxy servers?
- Operating system platform layer:
  - Use and take advantage of operating system security and services, and hardware security features?
  - Is the operating system security protecting the WebSphere configuration properties and files?
- WebSphere runtime environment layer:
  - What does the Web and EJB container offer for authentication and authorization?
  - How does the WebSphere Application Server runtime allow you to integrate authentication and authorization support?
  - Is the Network Deployment security deployment configuration properly set and understood for node, cell and node agent?
- Java application and JVM layer:
  - Understand the new Java 2 and J2EE security functions and features and how they interact and integrate with security features provided by the WebSphere Application Server.
  - Need the fine grained access or need programmatic security?
  - Need J2EE-based security policies and support J2EE security APIs?
- Enterprise information system (EIS) layer:
  - Have you taken the proper actions to ensure that security is maintained from the application to the Java 2 connectors and then to the EIS and the back-end data source?
  - How will the new principal identity mapping work for your EIS transactions?
- Database layer:
  - How are security identities being passed and maintained?

2.2.2 Ask the key questions

The following list is a list of questions to keep in mind while determine the right level of security for your environment:

- What are the security requirements, goals and objectives?
  - What kind of security does this project require?

  - Identify how much security is appropriate
  - Note that near flawless security is neither achievable nor practical.
- What are the costs of compromised security?
  - Awareness of vulnerabilities and weaknesses
  - Risk tolerance level on malicious intent
    - Financial losses (warranty costs)
    - Reputation and creditability loss, which leads to further financial losses
- Exploiting emerging technologies and trends
  - Are new emerging trends and technologies necessary for success?
  - Make sure you understand the pros and cons of new functions and features.
  - Choose wisely from what is available
- Investing in industry standards
  - Avoid creating home-grown security rules and policies that do not conform to the J2EE programming model.
  - Long term cost savings
    - Extendability and maintainability
- Are people you do business with security-minded?
  - Do your business partners, vendors, and suppliers have proper security practices, processes, and measures to protect critical information that belongs to you?
  - Do customers demand that your enterprise security infrastructure be nearly flawless to protect their critical data or privacy?
- Simplify complexity
  - It is tempting to want to build a bullet-proof environment.
    - Do what is practical and necessary.
    - Do what is possible and manageable.
  - Avoid overkill, that is, do not spend a million dollars to protect a dime.
  - It is important to keep it simple.

2.3 Making some key decisions

It is possible that the security architecture of your enterprise is already known. Or, perhaps you are trying to determine what elements should be included in

your architecture and how those elements affect your deployment and configuration of WebSphere Application Server. In this section we explore some of the more important questions you might ask and, depending on the answers, where else in this book you might look for further information.

2.3.1 Intranet or Internet?

Are the applications that you currently want to deploy in WebSphere Application Server intended for use only by individuals within your enterprise (intranet), or by a wider population extending outside your enterprise (Internet)? In addition to influencing your network topology (for example, whether you will need a firewall between the Internet and your application server), the answer to this question might determine whether you want the users to authenticate with existing SAF identities or with identities from a different registry.

If you plan to use only SAF identities, you can use an OS registry (such as RACF), as described in Chapter 12, “Local operating system registries” on page 343.

If you are planning to use identities from another source, read Chapter 13, “Remote registries” on page 361.

2.3.2 Where will authentication take place?

The answer to this question can be determined by considerations discussed in 2.3.1, “Intranet or Internet?” on page 29. Often, if an SAF identity is used, authentication takes place within WebSphere Application Server itself. This might also be the case for other identities if an LDAP or a custom registry is used. For information about authentication in the Web container, see Chapter 16, “Web container security” on page 501. If your client is not a Web browser or other HTTP client but instead is a Java client using RMI/IIOP to invoke EJBs directly, you can read about authentication within the EJB container in Chapter 18, “EJB container security” on page 605.

For Internet applications especially, many enterprises use a DMZ between firewalls and perform authentication there. You can read about authenticating in a DMZ using a proxy server in Chapter 13, “Remote registries” on page 361, especially 13.8, “Sample scenario: Off-platform authentication with WebSEAL on Linux for zSeries using TAI” on page 409.

There will be cases where none of the authentication methods provided in WebSphere or Tivoli products meet the exact needs of your application. You might find it necessary to perform some authentication function in the application itself. In this case, see Figure 4 on page 79.

2.3.3 How will authorization to resources be determined?

You have several choices for authorization. As we discuss in 3.2, “J2EE container-based security” on page 46, J2EE supports role-based authentication within the Web and EJB containers. There are two methods for mapping user and group names to J2EE security roles. One is to permit users and groups to SAF EJBROLE or GEJBROLE profiles, as described in 12.2.3, “SAF-based WebSphere authorization concepts” on page 351. The other is to create XML security role mappings from roles to users during application installation. The process for performing this mapping is outside the scope of this book, because it is covered in detail in the *WebSphere Application Server Information Center* at:

```
http://www.ibm.com/software/webservers/appserv/infocenter.html
```

You will find an illustration, however, in Figure 7-26 on page 154.

Some enterprises might choose to centralize authentication and authorization in a server that is separate from where WebSphere Application Server runs. You might use Tivoli Access Manager, as discussed in “Using Tivoli Access Manager with a CDAS” on page 364.

There will be cases where none of the authorization methods provided in WebSphere or Tivoli products meet the exact needs of your application. You might find it necessary to perform some authorization function in the application itself. In this case, see 4.1, “Programmatic security” on page 80.

2.3.4 What other resources need to be accessed?

You might often deploy applications to WebSphere Application Server that are not entirely new and self-contained. They might require access to resources that fall outside the scope of WebSphere Application Server or the J2EE model itself. You might, for instance, need to access established applications running in a CICS or IMS environment. Or, you might need established data that is managed by DB2.

These products might require a different identity than those that are authenticated by WebSphere Application Server. This would be the case, for example, if WebSphere Application Server is using an LDAP or custom registry, or if authentication is being done in a DMZ against a registry other than SAF. In these cases, you must provide a way for the enterprise information system to be supplied with a SAF identity. This is explained in *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064.

However, you might need to access files or data sets or other resources protected by the operating system. In this case, Java 2 security could enhance

the security of your application. Read about Java 2 security in 3.5.2, “Java 2 security” on page 66.

2.4 Finding the right balance for your application

So far the discussion has revolved around how to evaluate the enterprise architecture to manage risk. Prior to J2EE, for Java-based enterprise applications we focused on byte-code verifier, jarsigner, and classloader security. With J2EE 1.3 Java security conformance, we move beyond the basics and take a closer look at the new security features that are available for Web applications.

2.4.1 Container-managed security

The container provides a number of services to an application. These services (including security) allow the components to take advantage of server functions without extensive coding.

Container-managed security (also called declarative security) is customizable and configurable so that application developers have the ability to take advantage of a security infrastructure without hard-coding security access rules. Container-managed security is the first attempt at removing application developers from security, allowing developers to concentrate on business logic, rather than its interaction with infrastructure. For many enterprises this is the perfect method for their security needs.

Container-managed security allows the developer to tie methods of an object to an abstract role. This makes it possible to code a program to a generic group of identities that can be managed by security people in charge of the infrastructure. With container-managed security the bean creators define what role is required in order to access each method. The infrastructure security implementers are responsible for creating the roles in the security product or WebSphere bindings and adding users to the roles. This allows the container to determine if the user can execute the code.

2.4.2 Application-managed security

Although it is best to have the infrastructure responsible for managing security, a totally container-managed security model might not be granular enough for many enterprises today. The new Java 2 security provides the ability to enforce fine-grained access control and policy-based security. Defining permission support at the enterprise application level is a step closer to end-to-end e-business security.

J2EE component security provides the role-based security for JSPs, servlets, and EJBs. This provides the capabilities that allow application developers to define or declare this role-based security in configuration settings or even business logic within the application code. Therefore, J2EE security can be enforced either in the runtime environment or programmatically, or both.

JAAS application program-controlled security

When declarative security alone does not provide enough flexibility for a business's need for permission and access control decisions, Java Authentication Authorization Service (JAAS) provides an extension to Java 2 security that allows the application developer to extend the authentication and authorization programmatically within an application.

The JAAS programming model allows the application developer to design application authentication as pluggable, which makes the application independent from the underlying authentication technology. JAAS is the strategic programming model for WebSphere; it replaces the CORBA programmatic login APIs.

Many security products are starting to provide implementations of the JAAS architecture, making it easier for applications to make use of existing security infrastructures to minimize the amount of code an application developer must write. In addition, many businesses see the value of creating enterprise-wide security services that all applications use. These allow application developers to make use of the security services without having to code them. It also allows a group of application developers to focus on security for all applications.

2.5 Topological view of security

A successful WebSphere Application Server for z/OS security implementation connects many instances and types of servers together so that they appear, to a client, to be a single secure system, with a single sign-on to a set of related, authorized business functions.

Security for WebSphere Application Server for z/OS is a large enough topic that the table of contents of this book might not organize the information in a way that best suits everybody's needs. A second option is Figure 2-1 on page 26, which organizes a very large set of terms into "layers," that might also help you to access terms related by layer by looking them up in the index of this book. A third option is presented here, a simple graphical organization based on a small set of diagrams:

- A base topology illustrating the most common system types and connections
- The base topology overlaid to represent specific aspects of security

Each diagram is followed by annotations and narrative text and references to other sections of this document and other IBM publications that provide further details.

2.5.1 Base topological view

The base topology presented here is generalized. It does not imply whether or not any of the system types are running on the same host system or even sysplex, and it does not imply how many instances there are for each type of system. For example, we can assume that the WebSphere application servers and the EIS systems are running on z/OS, but it is possible that the IBM HTTP Server is running on a non-z/OS platform with plug-in connectivity to the z/OS application server, and IBM WebSphere Edge Servers always run on a non-z/OS platform.

This is not an architectural design, nor is it a recommended, specific topology. Instead, it enables us to reference certain standard server types and products diagramatically and clarifies which products interoperate with which. This simplified model provides a useful navigation into the other sections of the book.

The diagram shown in Figure 2-2 also reflects the number of different administrator guides and references that are required to complete a security implementation. Each server type is documented separately, and resolution of interoperability issues between server types poses a special challenge.

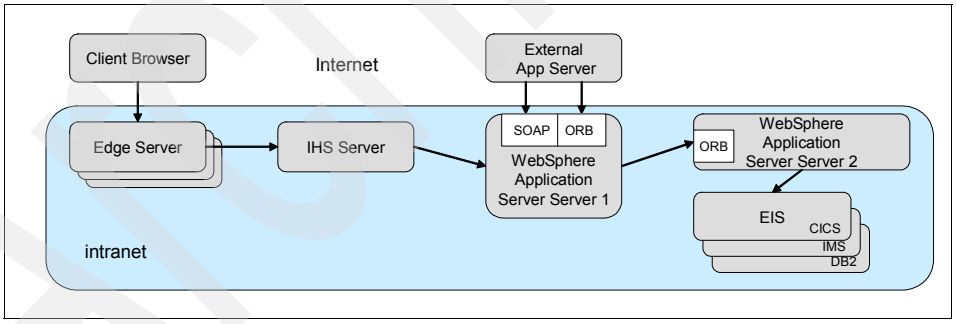


Figure 2-2 Base topology

The diagram illustrates that a WebSphere Application Server for z/OS implementation typically represents a “wrapper” to existing established databases and transactions, making the business functions available to clients through the Internet, without requiring extensive changes to the established systems.

Typical clients for the enterprise system include users accessing the system with Web browsers, but might also include customer or supplier application servers acting as clients to the enterprise's WebSphere application servers.

2.5.2 Encryption

An aspect of WebSphere security that cuts across all of the server types is encryption, which can be generalized as the use of SSL for all TCP/IP connections. Global use of encryption requires careful planning and configuration for each of the connections, as shown in Figure 2-3.

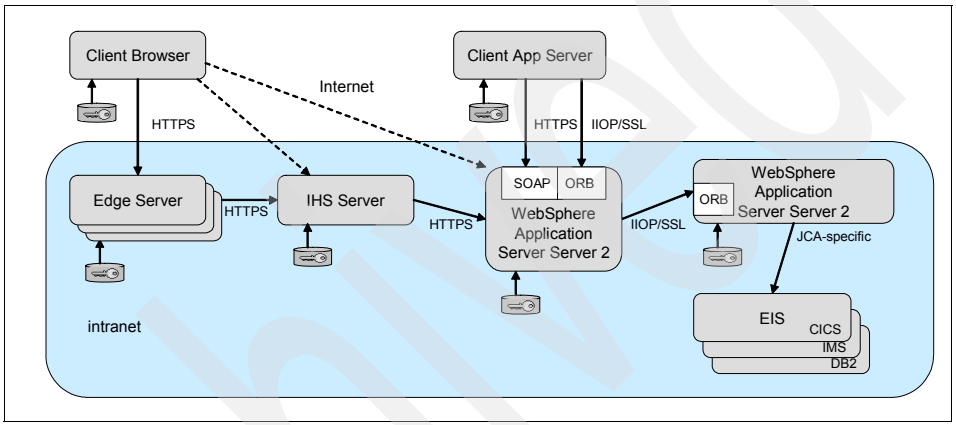


Figure 2-3 Encryption

We illustrate the fact that there are two protocols that are used to access transactions within the application server, including HTTP for Web applications and IIOP for EJBs; both must be configured to run over SSL for all transactions that include security constraints. The icons with keys simply represent keyfiles and certificate stores that have to be managed so that private encryption keys and trusted certificates are kept in synch between the different servers.

HTTPS connections

HTTPS is the HTTP protocol secured with SSL used to transport HTML requests and responses between servers and client browsers. HTTPS and SSL concepts in general are covered in 16.2.5, "Certificate-based authentication" on page 524.

For many implementations, there are no client browser configuration issues; Server certificates validate against CA certificates installed with the browser.

However, if the WebSphere SSL repertoire is configured to require client certificates, you will have setup issues that depend on whether the client connects directly to the WebSphere HTTP handler, or to a proxy server such as

IBM HTTP Server or Edge Server. For configuration information, refer to 16.2.5, “Certificate-based authentication” on page 524 and 17.3.7, “Protection setups in the IBM HTTP Server and security credential forwarding” on page 581.

If you are using cryptographic services, refer to 9.8, “Set up Secure Sockets Layer (SSL) for WebSphere for z/OS” on page 265.

The configuration of Edge Server and IBM HTTP Server is covered in Chapter 17, “Security integration with the WebSphere HTTP plug-in” on page 559.

IIOP/SSL connections

IIOP is the CORBA transport protocol for TCP/IP and is secured at the network level with SSL. It is used whenever remote EJB methods are invoked. The CORBA CSIv2 architecture defines IIOP/SSL requirements, which are presented in 1.6.7, “CSIv2 security protocol” on page 15 and 3.4, “Security interoperability using IIOP” on page 60.

The configuration of IIOP/SSL in WebSphere is covered in 18.2, “Common Secure Interoperability Version 2 (CSIv2)” on page 608 and 18.3, “CSIv2 solution scenarios” on page 621.

2.5.3 User registries and authorization databases

Another aspect of security that affects several of the server types is the use of user registries and authorization databases.

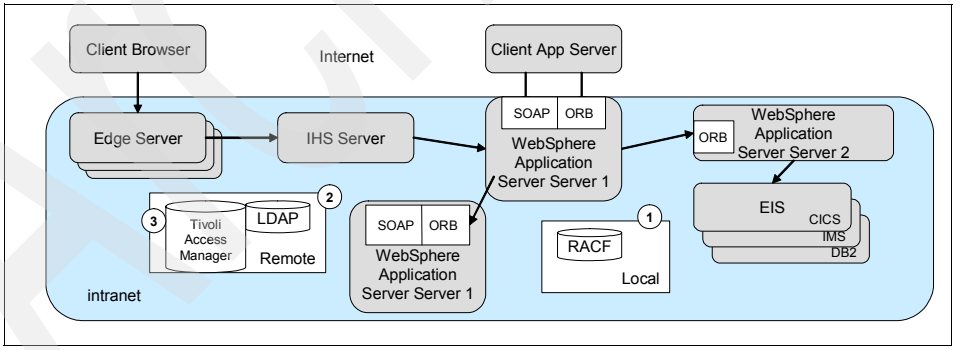


Figure 2-4 User registries

Figure 2-4 illustrates several types of user registries. Option 1 is RACF, a local OS registry (or another SAF-compliant registry). RACF is a good choice for a user registry provided that all of the WebSphere Application Server instances are running in the same sysplex. It can provide superior performance because it is

closely integrated with the operating system, and it provides the highest level of security. RACF is shown in the lower-right corner of the diagram because authentication for EIS systems is also based on RACF (or another SAF registry).

For more details about RACF user registries, refer to Chapter 12, “Local operating system registries” on page 343.

Options 2 and 3 are remote registries. A remote registry is required if single sign-on is required for users defined in a non-SAF registry. A third application server running on a non-z/OS platform is added to the basic topology to illustrate another scenario requiring a remote registry.

Option 2, LDAP, can be hosted on many different platforms, including z/OS. For more details about LDAP user registries, refer to 13.5, “Sample scenario: Using LDAP native authentication” on page 374.

Option 3, Tivoli Access Manager, can be hosted on several platforms other than z/OS. Tivoli Access Manager and WebSEAL are two Tivoli products that are designed specifically to enable single sign-on and to move the management of authorization out of WebSphere to Edge Servers. For more details about Tivoli Access Manager user registries, refer to 13.6, “Sample scenario: Authentication with Tivoli Access Manager for e-business WebSEAL” on page 398.

2.5.4 Identity flow

Another aspect of security, which simplifies authorization, is the automatic flow of identity from client through application servers to the enterprise information systems (EISs).

From the perspective of the client, all of the servers shown in Figure 2-5 on page 37 appear to be a single secure system, with a single sign-on to a set of related, authorized business functions. The system views the client as an identity, which is then used to determine the transactions (servlet, EJB, or EIS) that the client is authorized to perform.

Frequently, an identity might be changed to a more general identity as it flows through the system, with fine-grained authentication and authorization occurring closest to the client, and more generalized authentication and authorization occurring closer to the EIS.

Documenting identify flow requirements can start at the enterprise information systems, where the identity requirement of the system is a function of how it was designed (user versus group versus system identity, and so on). Redbooks are currently being written that explain each of the IBM EIS connectors and how they are configured in WebSphere. Each of these books contains a flowchart illustrating how WebSphere connector factory settings and underlying EIS

connector settings control identity flow. The enterprise information system's identity requirements will identify an end node of the flowchart that should be used to select connector and connector factory settings so that the proper identity will flow to the EIS system.

Working backward from the EIS connection factory settings into the J2EE application design will require choices that, at a high level, define:

- The mix of declarative versus programmatic techniques
- The degree of identify specificity required for the J2EE components

Figure 2-5 illustrates some of the choices that control identity flow through the system, which, in turn, affect the configuration of user registries and authorization services. The sample flow at the bottom of the figure suggests a small subset of the possible choices and does not represent a typical or recommended flow.

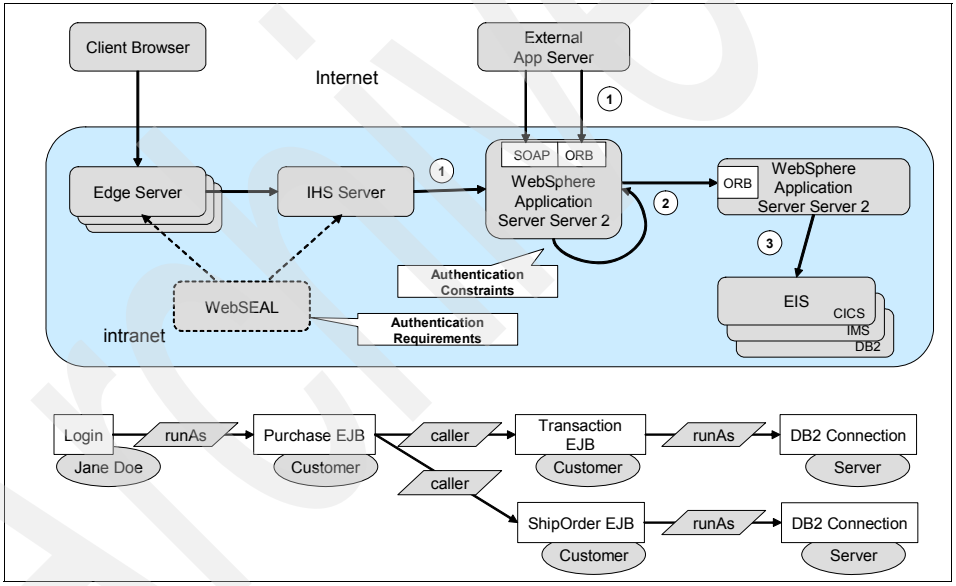


Figure 2-5 Identity flow

The following steps describe the sample flow:

1. An initial browser session might access a non-secured function on the enterprise system, with no authentication. However, the first time that the session accesses a secured resource, the client generally authenticates using a user ID and password.

   An external application server generally accesses a secured connection and authenticates to the system to establish an identity.

Determination of whether or not user authentication is required and the method for obtaining the user ID are typically defined in Web applications, using the authentication methods described in 16.2, “Web container authentication” on page 504, and the security constraints described in 16.3, “Web container authorization” on page 541.

A product, such as WebSEAL, used with Edge Server or IBM HTTP Server, can be used to move authentication requirements and method decisions out from the WebSphere Application Server, as described in 13.3, “Trust association interceptor” on page 366, 17.2.2, “WebSphere HTTP plug-in” on page 569, and Chapter 14, “IBM Tivoli Access Manager and WebSphere Application Server integration” on page 417.

2. The WebSphere application server is where the Java Authentication and Authorization Service (JAAS) is used to create a subject, as explained in 3.5.3, “Java Authentication and Authorization Service (JAAS)” on page 74. The subject is a representation of identity that can easily be changed as transactions flow from servlets to session EJBs, and then to entity EJBs and EIS transactions.

   There are several important techniques that can be used in the application server flow to change the identity to a more generalized identity, or to define authorizations for groups of users rather than individual users.

   Techniques to change identities include:

   – Run-as definitions for EJBs and servlets. This provides declarative control of identity using any one of the J2EE 1.3 or WebSphere Version 4.0 options; refer to 3.2.4, “RunAs versus run-as: Identity propagation” on page 52.

   – The javax.security.auth.Subject doAs() method. This provides J2EE programmatic control of identity; refer to 4.2, “JAAS for application security” on page 83.

   Techniques to define authorizations for groups of users include:

   – User ID to group ID mappings in user registries. All user registries support the association of multiple users to a single functional group, which can then be mapped by authorization facilities to authorization constraints in servlets and EJBs. This aspect of user registries is described in 11.1.2, “User registry authorization data” on page 335 and 13.1.2, “Authorization with a remote registry” on page 363.

   – Role-based servlet and EJB authorization. This provides J2EE declarative authorization for groups of users to servlets and EJBs; refer to 16.3, “Web container authorization” on page 541 for servlet roles, 18.5.2, “Resource authorization at the application level” on page 648 for EJB roles, and 12.2.3, “SAF-based WebSphere authorization concepts” on page 351 for a description of how to use RACF to map EJB roles to users and groups.

3. Identity flow from WebSphere Application Server to an EIS using J2EE Connector architecture connectors is illustrated here. Techniques to change identities within the connector include:
   - Connection Manager RunAs Identity Enabled.
   - The javax.resource.spi.ManagedConnection getConnection() method. This provides J2EE programmatic control of identity.
   - Res-auth definitions for J2CA connectors. This provides J2EE declarative control of identity.
   - Thread identity and thread security. Closely related declarative techniques for determining whether the thread identity is passed to the underlying EIS when using a local connection.

   For information about connector identity flow, refer to *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064.

   Additional issues with identity flow are specific to the underlying EIS connector. Flowcharts showing the relationship between connection factories, connectors, and identify flow are planned for an upcoming series of Redbooks documenting WebSphere connectors.

Note: J2CA is the J2EE architected mechanism for invoking an EIS from an application server. Depending on the capabilities of the EIS, there might be other means of communicating with it. CICS, for example, supports JMS, SOAP, and IIOP clients. These communication methods each have unique security characteristics, many of which are discussed in *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064 and *WebSphere for z/OS Connectivity Architectural Choices*, SG24-6365.

2.6 Summary

Security is about risk management. First and foremost, identify the risks the enterprise is facing and determine how much exposure your environment can tolerate based on business requirements and objectives. Make your key decisions on security elements you need by examining your current security architecture end-to-end.

The security infrastructure should be examined from an enterprise-wide point of view. It should be flexible enough to encompass the needs of all of the platform requirements and should be built with an eye on exploiting new technologies as they emerge.

There are many evolving technologies and trends in the security arena. New functions and techniques are creating an increasingly sophisticated infrastructure

to implement security appropriately. The right balance of this technology and a good overall security policy will ensure that the right investment is made to create a secure environment.

One thing to keep in mind about security is that at the very least you should turn on WebSphere global security to prevent unauthorized persons from connecting to a server and making unwanted changes to your production environment. Make sure you secure the file system that holds your configuration files and has passwords and log files with confidential system information. Continue to rely on security functions and features provided by WebSphere container-managed security, operating system security, and network security, at a minimum.

For further background information, including security concepts from WebSphere Application Server V4, many of which continue to apply to V5, see *z/OS WebSphere and J2EE Security Handbook*, SG24-6846.

3

J2EE 1.3 and WebSphere Application Server V5 security concepts

This chapter provides a conceptual overview of the security elements included in the Java 2 Platform Enterprise Edition (J2EE) 1.3 specification and implemented in WebSphere Application Server for z/OS and OS/390 Version 5.

This chapter contains the following sections:

- Overview
- J2EE container-based security
- Resource authentication
- Security interoperability
- Additional security capabilities

3.1 Overview

The notion of security includes several aspects.

Identification is the process by which users or programs that communicate with each other provide identifying information.

Authentication is the process used to validate the identity information provided by the communicating partner.

Authorization is the process by which a program determines whether a given identity is permitted to access a resource, such as a file or application component.

The J2EE specifications describe the concepts used for these processes. This is described in 3.2, "J2EE container-based security" on page 46.

J2EE specifies a component programming model for security that provides both application programming interfaces (APIs) and declared properties. The security APIs used in the logic of application programs are referred to as *programmatic security*. The declared security properties, called *declarative security*, are found in components' deployment descriptors and in policy files. One objective of the J2EE programming model is to encourage the use of declarative security, which is enforced by the container. This removes much of the responsibility for security from the application developer.

J2EE does not provide a security policy. It provides APIs and security properties. J2EE products, however, such as WebSphere Application Server for z/OS and OS/390, can implement the APIs and security properties in a way that defines a security policy.

IBM WebSphere Application Server Version 5 provides the security infrastructure and mechanisms to take the following steps:

- Protect sensitive J2EE resources and administrative resources
- Address enterprise end-to-end security requirements for:
  - Authentication
  - Resource access control (authorization)
  - Data integrity
  - Confidentiality
  - Privacy
  - Secure interoperability.

IBM WebSphere Application Server security is based on industry standards. Version 5 has an open architecture that allows secure connectivity and interoperability with the following:

- Enterprise information systems (including DB2, CICS, WebSphere MQ and EIS products from other vendors)
- Security providers (including Tivoli Access Manager and WebSEAL secure proxy server)
- Products from other vendors

WebSphere Application Server provides a unified, policy-based, and permission-based model for securing Web and Enterprise Java resources according to the J2EE specifications. Specifically, Version 5 complies with J2EE specification Version 1.3 and has passed the J2EE Compatibility Test Suite. Product security is a layered architecture built on top of the operating system platform, JVM, and Java 2 security, and employs a rich set of security technology including:

- The Java 2 security model, which provides policy-based, fine-grained, and permission-based access control to system resources. This is described further in 3.5.2, “Java 2 security” on page 66.
- The CSIv2 security protocol. CSIv2 is essential for interoperability among application servers from different vendors and with enterprise CORBA services. See 3.4, “Security interoperability using IIOP” on page 60 for more information about CSIv2.
- Java Authentication and Authorization Service (JAAS) programming model for Java applications, servlets, and EJBs. This is described more fully in 3.5.3, “Java Authentication and Authorization Service (JAAS)” on page 74.

The security model and interfaces supported include Java Secure Socket Extension (JSSE) and Java Cryptographic Extension (JCE), providing for secure socket communication and for message and data encryption. These are described in 3.5.5, “Cryptographic application and data security” on page 77.

WebSphere Application Server security sits on top of the operating system security and the security features provided by other components, including the Java language, as shown in Figure 3-1 on page 44.

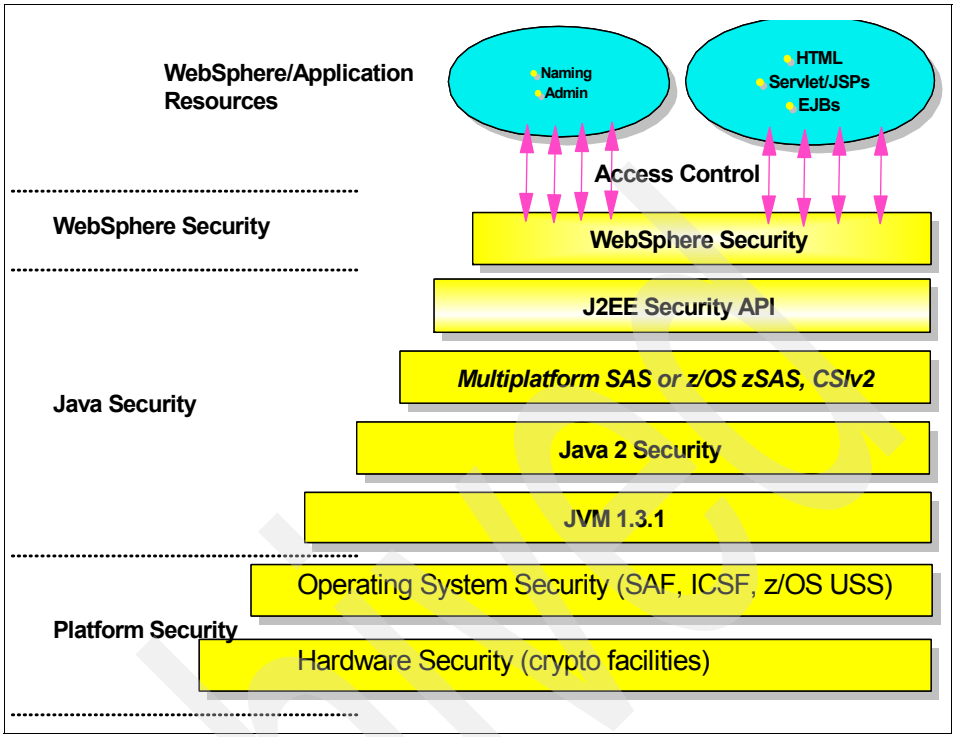


Figure 3-1 WebSphere environment security layers

3.1.1 Security server topology

Each application server, node agent, deployment manager, and daemon address space has a local security server that interacts with the platform security facilities. Figure 3-2 on page 45 shows a typical node in a WebSphere Application Server for z/OS configuration.

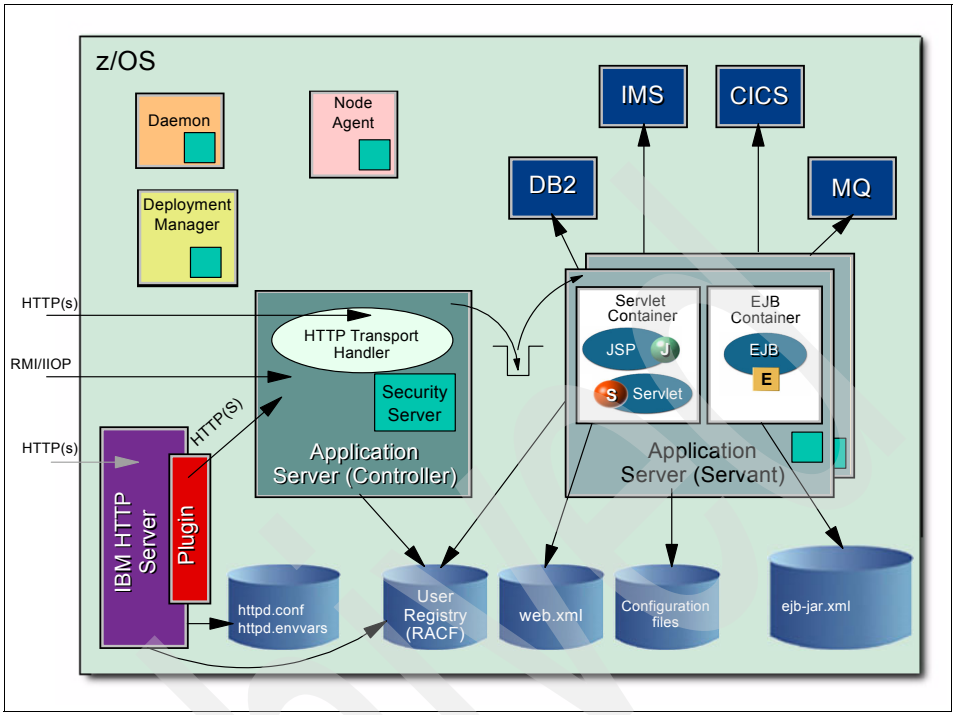


Figure 3-2 Security server in WebSphere environment

3.1.2 Terminology used for J2EE security

These terms have a particular meaning in J2EE:

Principal An entity that can be identified and authenticated (for example, the initiator of a request, such as a user or requestor).

Principal name An identity of a principal (a user ID, for example).

Credential Reference information, returned by the authentication mechanism, that can be used to authenticate a principal.

Subject A set of principals and their credentials associated with an execution context; an abstraction for a user.

Security role A logical grouping of users that share a level of access permissions.

Security domain This is the scope that defines where a set of security policies are maintained and enforced (also referred to as a “security policy domain” or “realm”). J2EE does not define the scope of a security domain.

3.1.3 User registries

The *user registry* stores user and group names for authentication and authorization purposes. Authentication mechanisms configured for WebSphere Application Server consult the user registry to collect user-related information when creating credentials that are then used to represent the user for authorization.

3.1.4 Global security

Global security is the security configuration effective for a cell and applies to all applications running on any application server in that cell.

It does the following:

- Determines whether security will be applied at all.
- Sets up the user registry against which the authentication will take place.
- Defines WebSphere Application Server authentication mechanisms.

Global security is managed from the administrative console and can be completely enabled and disabled by selecting a single switch. If global security is enabled, and a security constraint is set for a particular resource, the resource is secured. In a Network Deployment environment you can disable security on individual application servers while global security is enabled. However, you cannot enable security on an individual application server while global security is disabled.

3.2 J2EE container-based security

J2EE containers are responsible for enforcing access control on component objects and methods. Containers provide two types of security:

- Declarative security
- Programmatic security

Declarative security

Declarative security is used when an application can be *security unaware*. With declarative security the security policies of an application are defined in the deployment descriptor, external to the application code. The deployment descriptor includes the application's security requirements, security roles, access control, and authentication requirements. At runtime, the container uses the security policy defined in the XML descriptor files to enforce data constraints and access control.

Programmatic security

Programmatic security is used when an application must be *security aware*. It can be used by the application code to make access decisions when declarative security alone is not sufficient to express the security model of an application.

The J2EE Specification API provides methods for determining the caller's identity and role.

The EJB methods are:

- isCallerInRole()
- getCallerPrincipal()

The HttpServlet methods are:

- isUserInRole()
- getUserPrincipal()

3.2.1 Role-based authorization

The J2EE specification defines a security role as "A logical grouping of users that are defined by an Application Component Provider or Assembler."

The WebSphere security administration approach provides flexibility and portability. Security roles determine the security policies for an application by creating named sets of users (for example Manager and Clerk) that will have access to secure resources and methods. The deployer also has control over which mechanisms are used to determine role membership. Security roles can be configured for the application, Web components and the EJB components during deployment using a development environment such as WebSphere Studio Application Developer or the Application Assembly Tool (AAT).

Security roles can be mapped to users, groups of users, or *special subjects* during the process called *security role mapping*. Security role mappings can be performed from the WebSphere administrative console during application installation, or at any time after the application has been installed. When SAF is the user registry, security role mapping can be performed using profiles in the EJBROLE class or GEJBROLE grouping class.

There are two special subjects in WebSphere V5.0:

- *AllAuthenticatedUsers,* which permits all authenticated users to access protected methods
- *Everyone*, which permits unrestricted access to a protected resource

Note: These special subjects are not available out of the box with the SAF user registry. They can be emulated when given universal access rights to roles.

Security constraints protecting J2EE applications allow considerably more flexibility than those protecting traditional applications. Typically, applications as a whole are protected, if at all, by an ACL that is global with respect to the program. A user or group of users might or might not have permission to execute the program as a whole, but there is no means of restricting parts of the program short of including authorization logic within the application code itself. J2EE applications have a declarative means of protecting each individual method of each component by specifying, outside the program logic, which security roles can execute it.

In terms of programmatic security, security role names are used in the APIs discussed in 3.2.2, “Web container authentication and authorization” on page 50 and 3.2.3, “EJB container authentication and authorization” on page 51.

To implement both programmatic and declarative security, application developers (including what J2EE refers to as “component providers” and “assemblers”) declare role names in the component’s deployment descriptor. In addition, for declarative security the application assembler associates security roles with “method-permission” elements in the deployment descriptor. A user can invoke a method only if a role to which the user’s identity is mapped is listed on that method’s method-permission element.

Deployers make J2EE components and applications ready for particular operational environments. Part of this deployment process includes editing the deployment descriptor (usually through a graphical tool) to map security roles to users or user groups that are understood by the security implementation of the environment.

As depicted in Figure 3-3 on page 50, application developers (component providers) can use J2EE roles both programmatically and declaratively. The application assembler can link the role names used by application components from multiple sources. The security administrator or deployer provides role names that meet the requirements of the system where the application is being deployed. The security administrator links or permits individual users or groups to the role names.

The application developer, using a tool such as IBM WebSphere Studio Application Developer, can declare the security roles used in the application. Because multiple components might use different security role names for similar capabilities (for example, “manager” and “boss” in Example 3-1), the application assembler can link the role names used by the application components. In the example, the application assembler has linked the role names “manager” and “boss” to the role name “Supervisor”. The role name declarations and links appear in the deployment descriptor. It can be edited using WebSphere Studio Application Developer, the Application Assembly Tool (AAT), or tools from other vendors.

Example 3-1 Deployment descriptor: Security-role-ref example

```
Web deployment descriptor
. . .
<security-role-ref id="SecurityRoleRef_1058814855423">
   <description>Boss of the workers, used in isUserInRole()</description>
   <role-name>manager</role-name>
   <role-link>Supervisor</role-link>
</security-role-ref>
. . .
EJB deployment descriptor
. . .
<security-role-ref id="SecurityRoleRef_1058814855698">
   <description>Boss of the workers, used in
isCallerInRoleisCallerInRole()</description>
   <role-name>boss</role-name>
   <role-link>Supervisor</role-link>
</security-role-ref>
. . .
<security-role id="SecurityRole_1058814855818">
   <description>Boss of the workers, mapped to users & groups</description>
   <role-name>Supervisor</role-name>
</security-role-ref>
```

The security administrator or deployer provides role names that meet the requirements of the system where the application is being deployed. The security administrator links or permits individual users or groups to the role names. The

procedures used by the security administrator and deployer will vary depending on the deployment platform.

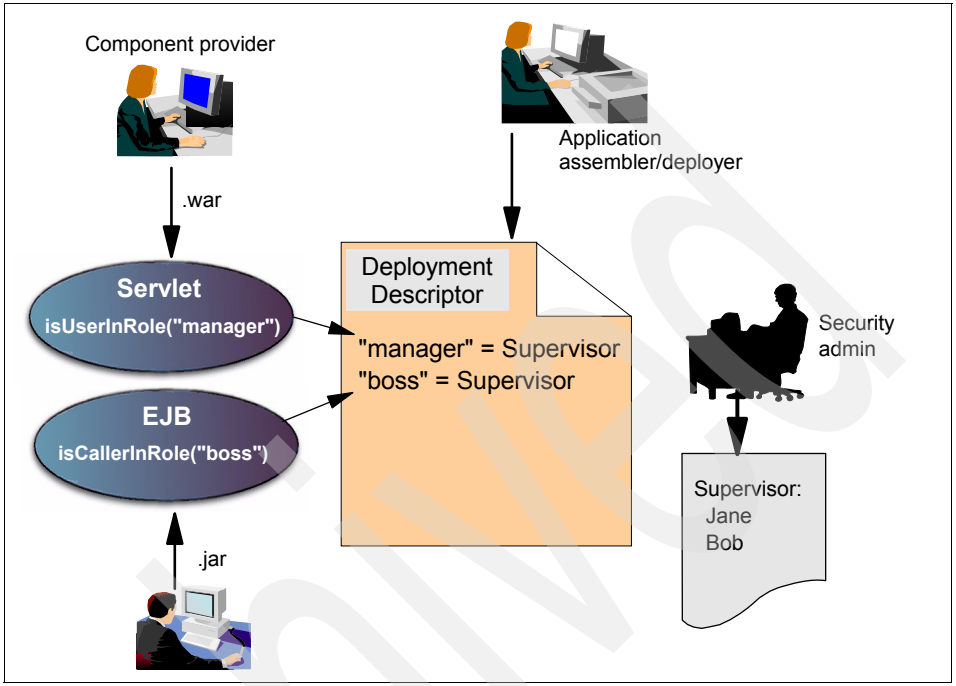


Figure 3-3 J2EE security roles

3.2.2 Web container authentication and authorization

Web containers implement J2EE-specified methods of authentication and authorization.

Web container authentication

The Web container follows the J2EE model. Requests that arrive through the IBM HTTP Server and one of the WebSphere HTTP plug-ins or through the HTTP transport handler can be authenticated using basic authentication (HTTP or HTTPS), form-based authentication, or client certificates.

WebSphere Application Server for z/OS and OS/390 does not support HTTP digest authentication.

Specifying the authentication method to be used

An application developer specifies which authentication method is to be used for an application by providing a *login-config* element in the application's

deployment descriptor. The login-config element specifies the type of authentication to use and any associated data, such as login and error pages for form-based authentication. Example 3-2 shows a login-config element.

Example 3-2 Sample login-config element

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>WASWebContainer</realm-name>
</login-config>
```

The deployment descriptor is maintained by tools such as WebSphere Studio Application Developer or the Application Assembly Tool.

Web container authorization

The Web container supports programmatic and declarative authentication models. Depending on the configured registry implementation, authorization takes place by checking SAF profiles (see 12.2.3, "SAF-based WebSphere authorization concepts" on page 351) or by checking the bindings file created by the deployment tools (see 13.1.2, "Authorization with a remote registry" on page 363). The Web container authorizes URL requests (resource collections).

Servlet filters

Although not intended primarily for security, servlet filters can be written to perform authentication and authorization functions programmatically. They can be configured to intercept HTTP requests from the Internet before any servlet is given control. Likewise, they can intercept HTTP responses after the servlet has finished its processing and before the response is returned. A servlet filter that is configured to intercept a request can even create a response and bypass execution of the servlet altogether.

Servlet filters can be combined, for example, with form-based login. Form-based login does not provide a way, if the login fails, to return a specific reason to the requestor. A servlet filter can be installed for the j_security_check URL to validate (as a request filter) the j_username and j_password values before the product-supplied j_security_check receives control. It could immediately return detailed failure information if the values are not valid. Alternatively, a response filter could be installed that would consult the security product for more detailed information when an error page is returned.

3.2.3 EJB container authentication and authorization

The Enterprise JavaBean (EJB) container supports different authentication and authorization methods from the Web container.

EJB container authentication

The EJB container supports programmatic and declarative authentication models.

The EJB container performs authentication when it receives a request for an EJB method through RMI/IIOP. You can configure the authentication methods the EJB container supports. This is specified as part of the server definition. Of course, the capabilities of the client sending the RMI/IIOP request and the location of the client with respect to the J2EE server control which of the specified methods is used.

The EJB container supports the following means of authentication:

- Basic authentication
- Certificate-based authentication
- Kerberos authentication
- Asserted identity
- Unauthenticated

These are described in Chapter 18, “EJB container security” on page 605. Asserted identity is elaborated for Version 5 in 3.4, “Security interoperability using IIOP” on page 60.

EJB container authorization

EJB container authorization follows the J2EE model and works basically the same as in the Web container. The EJB container authorizes EJB method calls.

3.2.4 RunAs versus run-as: Identity propagation

In enterprise applications, it is not always desirable to use the caller’s identity for authorization in all application components or situations. For example, an enterprise might want to make public information in a DB2 database available (on a read-only basis) to a large number of clients. Instead of granting read access on the DB2 table to each client user ID, it might be more efficient for the application (perhaps after authenticating the client) to switch to a generic identity for database access. This was not easy to accomplish with J2EE 1.2.

WebSphere Application Server for z/OS and OS/390 V4.0.1 provided an additional declarative setting, called runAs, for this purpose. Using runAs does not require any additional application code, because it is specified in an extended deployment descriptor (XDD) when the application is assembled.

WebSphere Application Server V5 for z/OS supports both the runAs concept introduced in V4 and the new run-as capability of J2EE 1.3.

J2EE run-as

The run-as capability defined in the J2EE 1.3 specification exhibits some differences from the IBM runAs introduced in WebSphere Application Server V4. While the IBM runAs is specified at a method level, the J2EE run-as applies to the entire component (servlet or EJB, for example). The IBM runAs applied only to EJBs, but the J2EE run-as can be specified for Web components (for example, servlets and JSPs) as well.

The J2EE run-as can have only a role-name as a value. If the caller's identity is desired for the component, the empty use-caller-identity attribute is used in the security-identity element of the deployment descriptor. Example 3-3 shows how J2EE run-as is specified in the deployment descriptor.

Example 3-3 J2EE run-as in the EJB deployment descriptor

```
<ejb-jar>
. . .
   <enterprise-beans>
      . . .
      <session>
         . . .
         <security-identity>
            <description>Identity to run the bean under</description>
            <run-as>
               <description>Boss of the workers</description>
               <role-name>Supervisor</role-name>
            </run-as>
         </security-identity>
      </session>
   </enterprise-beans>
   . . .
</ejb-jar>
```

RunAs (introduced in Version 4)

You can specify one of three values for runAs, as shown in Figure 3-4 on page 54. Caller specifies to use the caller's identity for the method selected and to propagate it to any subsequent methods invoked or J2EE resources accessed. This is the default behavior.

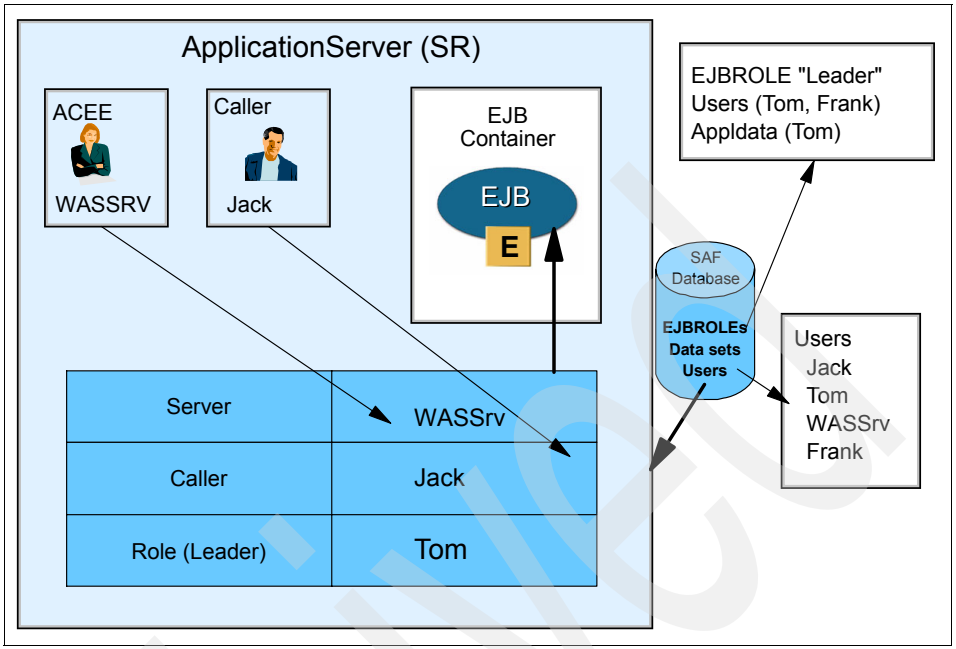


Figure 3-4 RunAs values

runAs Server indicates that the method should assume the identity of the application server region as represented in the Access Control Environment Element (ACEE). This identity will then be used and propagated to any subsequent methods invoked or J2EE resources accessed.

The third choice for runAs is Role. When this option is used, the application assembler selects the name of a security role that is defined in the application.

Example 3-4 shows an example of the extended deployment descriptor XML for specifying runAs.

Example 3-4 RunAs specification in ibm-ejb-jar-ext.xml

```
. . .
<ejbext: EJBJarExtension xmi:type=. . .>
   <ejbExtensions xmi:type=. . .>
      <runAsSettings xmi:id=. . .
         <methodElements xmi:id="MethodElement_1059145989166"
               name="getPrincipal" parms="" type="Remote">
            <enterpriseBean xmi:type="ejb:Session"
               href="META-INF/ejb-jar.xml#EJBSample"/>
         </methodElements>
         <runAsMode xmi:type="ejbext:UseSystemIdentity" . . ./>
      </runAsSettings>
```

```
      <enterpriseBean xmi:type="ejb:Session"
         href="META-INF/ejb-jar.xml#EJBSample"/>
   </ejbExtensions>
   <ejbJar href="META-INF/ejb-jar.xml#ejb-jar_ID"/>
</ejbext:EJBJarExtension>
```

Tip: In WebSphere Application Server V4, the runAs attribute had one of three values: The default Caller, Server, or Role. In contrast, while the Server attribute is not supported by the J2EE run-as, it can be simulated by assigning a role-name that is mapped to the servant region's SAF identity.

Using run-as with an SAF registry

With an SAF user registry, the security role name must be the same as that of an SAF EJBROLE profile. Security role names, however, are used to specify constraints and are not used to check authorization. Authorization is performed by checking that the principal name has been assigned to one of the required security roles. The principal name must be an SAF identity if an SAF registry is used.

In order to make this possible for a run-as role processing in an SAF registry, the parameter APPLDATA is included in the EJBROLE profile. Each EJBROLE profile whose name is to be used as a role name for runAs, must include an APPLDATA parameter that specifies an SAF user ID. This user ID must have at least READ access permission to the EJBROLE profile.

When Role is specified on the run-as mode, the SAF identity defined in the APPLDATA field (not the role name itself) is used and propagated to any subsequent methods invoked or J2EE resources accessed.

Tip: You can code APPLDATA in your EJBROLE and GEJBROLE profiles. In this case the runAs Role identity is picked from the APPLDATA field of the EJBROLE profile. This could lead to an unnecessarily complex security implementation. In order to keep the implementation clear and simple, we strongly recommend not duplicating APPLDATA in the EJBROLE and GEJBROLE classes.

Caller identity versus RunAs identity

It is important to understand the difference between caller identity and runAs identity.

- The caller identity is the identity being used for:
  - Checking methodPermissions of the current EJB
  - isCallerInRole(x)

  - getCallerPrincipal()
- The RunAs identity is the identity being used for:
  - Downstream authorizations (which means that the runAs identity will be used as the caller identity in EJBs called by the current EJB)
  - Outbound identity (which means that the runAs identity will be used as the user ID for EIS systems called from the current EJB when container-level authorization is specified)

Note: If the current EJB makes a call to a remote EJB hosted by a different server, the caller identity used by the remote server depends on the capabilities of the two servers to pass security information, for example, when a WebSphere Application Server running on z/OS or OS/390 can pass the runAs identity to another z/OS or OS/390 application server using a trust-based protocol such as asserted identity, Kerberos or SSL. If such a protocol is not used, the application server assigns a default value as the caller identity. In the case of z/OS and OS/390, this is the remote identity specified in the server definition.

Figure 3-5 on page 57 illustrates how the identity changes during the flow of an application. Here user Jack invokes the EJB1 method. Assume that Jack has been authenticated with an identity of Jack. A principal with the principal name of Jack is established. EJB1 runs with a caller identity of Jack. Because runAs is specified as Caller, when EJB1 invokes a method of EJB2 it passes the identity of Jack.

The EJB2 method, however, specifies runAs Role Leader. In this case, the method interrogates the EJBROLE profile for Leader and finds that the APPLDATA parameter specifies a user ID of Tom. Now EJB2 runs with a principal representing Tom. EJB2 runs with a caller identity of Jack and a runAs identity of Tom and passes the identity Tom when invoking the EJB3 method. The EJB3 method specifies runAs Server**,** so it uses the SAF identity from the application server region's ACEE when making any subsequent calls. Therefore, EJB3 has a caller identity of Tom and a runAs identity of the server region's SAF identity.

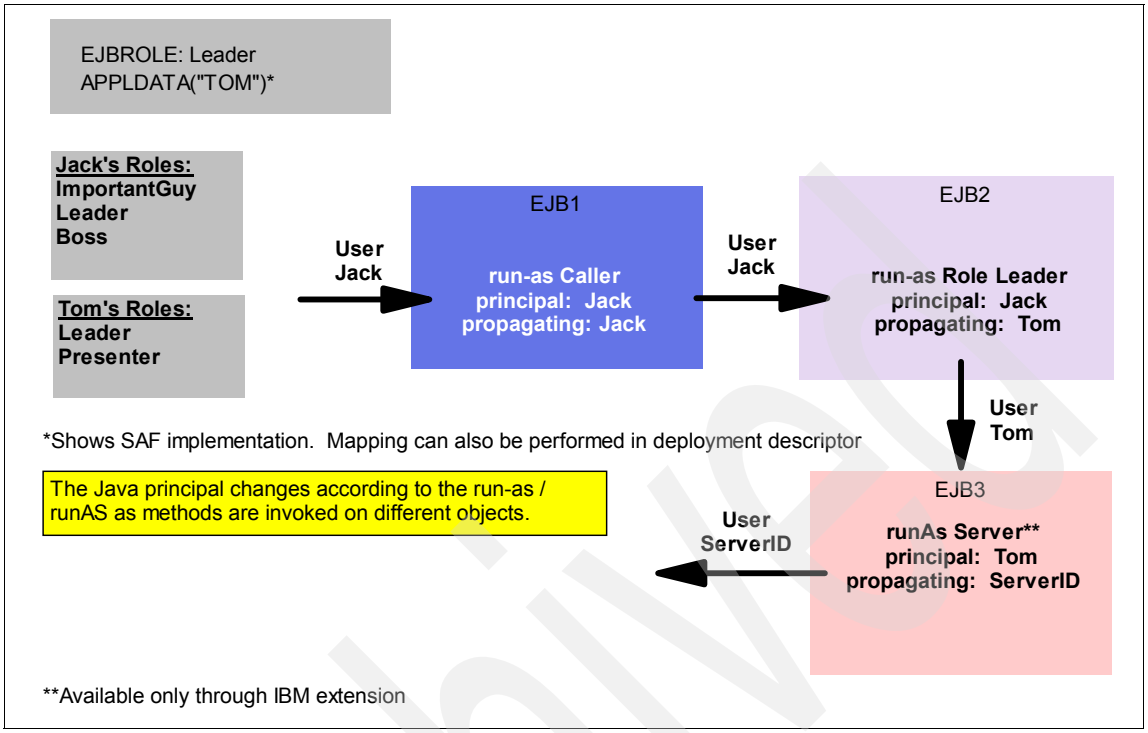


Figure 3-5 RunAs and identity propagation

Figure 3-6 on page 58 shows another way of depicting the relationship between the caller identity and the runAs identity.

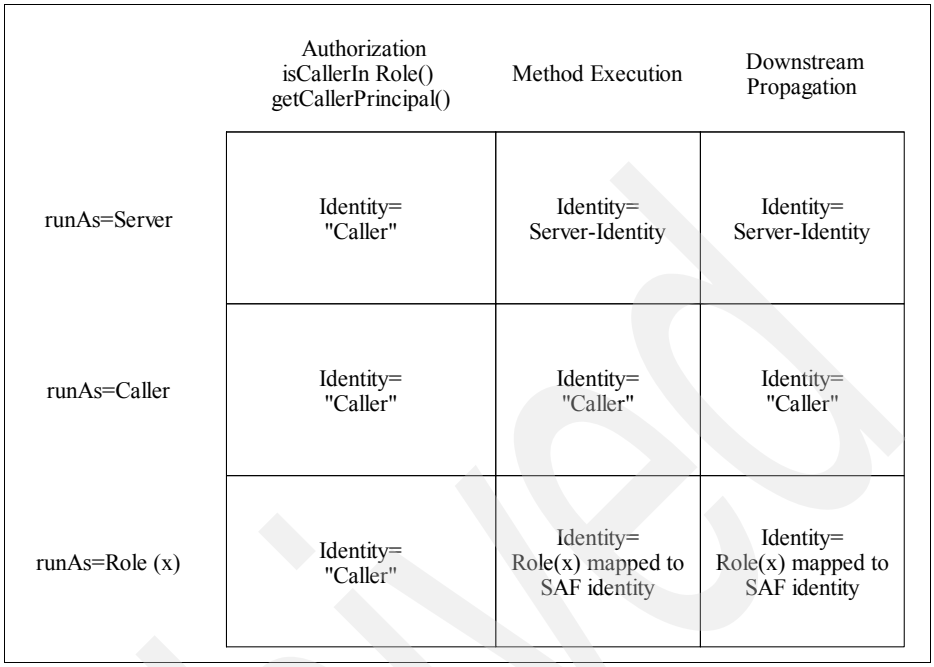


Figure 3-6 Use of caller identity versus RunAs identity

Caller, in this case, is the identity (principal) associated with the request for the method. The request might be coming from a Web application, in which case the caller is the principal associated with a servlet or JSP. It might also be from a remote RMI/IIOP client, in which case the caller identity is determined by the authentication protocol executed between the client and the server. If the request comes from another EJB, the caller identity is determined by the runAs setting of the calling EJB.

3.3 Resource authentication

In addition to authentication by Web and EJB containers, we must consider how authentication is to be accomplished for something called *resource factories*. A resource factory is an object that is used to create *resources*. Resources are means of accessing facilities (databases, for example) outside the realm of J2EE objects. The most common resources in use today are:

- Java Database Connectivity (JDBC) connections to relational databases
- J2EE connectors to enterprise information systems such as transaction managers

- Java Messaging Service (JMS) connections to messaging and queuing systems

When an application uses a resource factory, there usually must be a means of specifying to the container how authentication information is to be passed to the resource factory and to the resources that it creates. J2EE handles this through the *res-auth* deployment descriptor element.

The res-auth element can have two values. The first possible value, “Application” for EJB containers or “servlet” for Web containers, indicates that the application itself provides authenticating information. This can be done, for example, by coding a `getConnection(userid,password)` call, where the authenticating information consists of a user ID and password that the resource will be able to authenticate.

The other possible value, “Container,” indicates that the container provides authenticating information. In this case, it is the responsibility of the container to sign on to the resource using information provided by the deployer. How this information is provided is not within the scope of the J2EE specification, because it differs for each container’s implementation.

Example 3-5 shows an example of the XML for setting the res-auth value in the deployment descriptor.

Example 3-5 Example of res-auth specification in the deployment descriptor

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <resource-ref>
        <res-ref-name>jdbc/MyResource</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
      </resource-ref>
  </entity>
</enterprise-beans>
```

There is one exception to this logic. For JDBC connectors to DB2, the identity provided when the res-auth container is set will always be the application server region’s identity, unless Connection Manager RunAs Identity Enabled has been specified for the server into which the application is installed (see *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064). In this case, the JDBC connection behaves in the same way as the other resources and uses the identity derived from runAs.

3.4 Security interoperability using IIOP

The Common Secure Interoperability (CSI) security specification is defined by the Object Management Group (OMG), see:

http://www.omg.org

Currently in its second version (CSIv2), the specification defines the Security Attribute Service (SAS) protocol to address the requirements of Common Object Request Broker Architecture (CORBA) security for interoperable authentication, delegation, and privileges. The J2EE 1.3 specification requires application servers to support CSIv2 protocols. The SAS protocol is designed to exchange its protocol elements in the service context of a General Inter-ORB Protocol (GIOP) request, and reply messages that are communicated over a connection-based transport.

The protocol is intended to be used in environments where transport layer security, such as that available through Secure Sockets Layer (SSL) / Transport Layer Security (TLS) or Secure InterORB Protocol is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality that can be applied to supplement the features an underlying transport. The SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports can be unified.

Tip: The acronym "SAS" is used here to stand for the CSI Security Attribute Service. The same acronym is commonly used in WebSphere Application Server multiplatform publications to refer to the Secure Association Service. The meaning of SAS can usually be determined from the context, but the reader is cautioned to remain aware of this usage.

The SAS protocol is divided into two layers:

- The *authentication layer* is used to perform client authentication where sufficient authentication could not be accomplished in the transport layer.
- The *attribute layer* can be used by a client to deliver security attributes, such as identity and privilege, to a target where they might be applied in access control decisions.

The attribute layer also provides the means for a client to assert identity attributes that differ from the client's authentication identity (as established in the transport or SAS authentication layers). This identity assertion capability is the basis of a general-purpose impersonation mechanism that makes it possible for

an intermediate to act on behalf of some identity other than itself. This can improve the performance of a system, because the authentication of a client is relatively expensive. The server can validate the request by checking its trust rules.

In order for a component to invoke an EJB method that has been secured, there must be a protocol to determine the level of security and type of authentication to be used by the client and server. During the method invocation, the protocol must join the server's authentication requirements with the client's and select the appropriate policy. The server's requirements are determined by the object's Interoperable Object Reference (IOR). The client's authentication requirements are determined by the client's configuration.

WebSphere Application Server can be configured to support both CSIv2 and the set of protocols supported by WebSphere Application Server for z/OS and OS/390 V4.0.1 known as zSAS. In fact, both protocols can be supported *simultaneously*. The application server can receive a request using one protocol and then receive another request using the other protocol. zSAS is provided in Version 5 for interoperability with older clients and servers. CSIv2 allows vendors to securely inter operate and provides a greater number of features than zSAS.

Identity assertion

Identity assertion is the process by which the invocation credential is asserted to the downstream server during a call.

When a client authenticates to a server, the *received credential* is created. When authorization checks the credential to see whether access is allowed, it also sets the *invocation credential* so that if the component calls an EJB method located on another server. The invocation credential can be used as the identity with which to invoke the downstream method. Asserted identity for CSIv2 is explained in detail in "Identity assertion" on page 610.

Note: CSIv2 is intended to support interoperation not only with WebSphere Application Server, but with other application servers supporting J2EE. For example, since the end of 2004, CICS Transaction Server V2.3 is able to accept asserted identities from and assert identities to WebSphere Application Server when RMI/IIOP is used to communicate between EJBs in each product.

3.5 Additional security capabilities

In this section, we describe additional security capabilities of WebSphere Application Server for z/OS Version 5.

3.5.1 Authentication mechanism and single sign-on (SSO)

Authentication is the process of establishing whether a client is valid in a particular context. In WebSphere Application Server for z/OS Version 5, authentication mechanisms are based on JAAS implementations. The selection of an authentication method, how the client provides an identity and authenticating information (such as user ID and password), is determined by the application through its deployment descriptor and the capabilities of the client and server.

An authentication mechanism, as distinct from an authentication method, is configured globally for a cell. It is responsible for creating a credential that represents a successfully authenticated client user. Not all credentials have equal capabilities. The abilities of the credential are determined by the configured authentication mechanism. Each of the available authentication mechanisms (SWAM, LTPA and ICSF) provides credentials with slightly different formats and uses.

Lightweight Third Party Authentication (LTPA)

LTPA is intended for distributed, multiple application server and machine environments, including z/OS and other platforms. LTPA generates a security token for authenticated users that can be used with multiple servers. Considerations for LTPA are:

- LTPA supports single sign-on (SSO).
- LTPA requires that the configured user registry is a centrally shared repository such as RACF or LDAP or that the involved registries are synchronized.
- LTPA requires a symmetric cryptographic key that is shared among all of the servers in a domain.
- LTPA does not work with WebSphere Application Server for z/OS Version 4 servers.

Integrated Cryptographic Service Facility (ICSF)

ICSF as an authentication mechanism refers to the use of Integrated Cryptographic Service Facility by WebSphere Application Server for z/OS to generate a security token (for authenticated users) that can be used with multiple WebSphere Application Server for z/OS servers. It is intended for use with SSO with high security where multiple WebSphere Application Server for z/OS servers are accessed, including those running WebSphere Application Server for z/OS Version 4. Considerations for ICSF include:

- ICSF supports forwardable credentials and SSO.

- ICSF authentication requires that the configured user registry is a central shared repository such as z/OS SecureWay Security Server RACF (RACF) or LDAP.
- ICSF authentication requires the creation of ICSF keys that must be available to each server in the domain.
- ICSF authentication does not work with WebSphere Application Server on any platform other than z/OS.

Tip: The ICSF authentication mechanism is supported even if you use a registry, such as LDAP or CUR, that is not SAF based. The ICSF authentication mechanism will work only when the cryptographic hardware is enabled, as described in 9.4, “Activation of hardware cryptography support for zSeries 2084, 2086, 9672, 2064, 2066, or 7060 engines” on page 247, and if ICSF is configured properly, as described in 9.5.2, “Configure WebSphere to use hardware cryptography in support of the ICSF authentication mechanism” on page 259.

Important: The hardware cryptography support required by ICSF is not contained automatically in the zSeries 990 or 890 engines. If you use ICSF, you must install the PCIXCC feature. See 9.3, “Hardware cryptography support for zSeries 2084 or 2086 engines” on page 245for more information about this topic.

Figure 3-7 on page 64 illustrates the effect on SSO of which authentication mechanism is chosen.

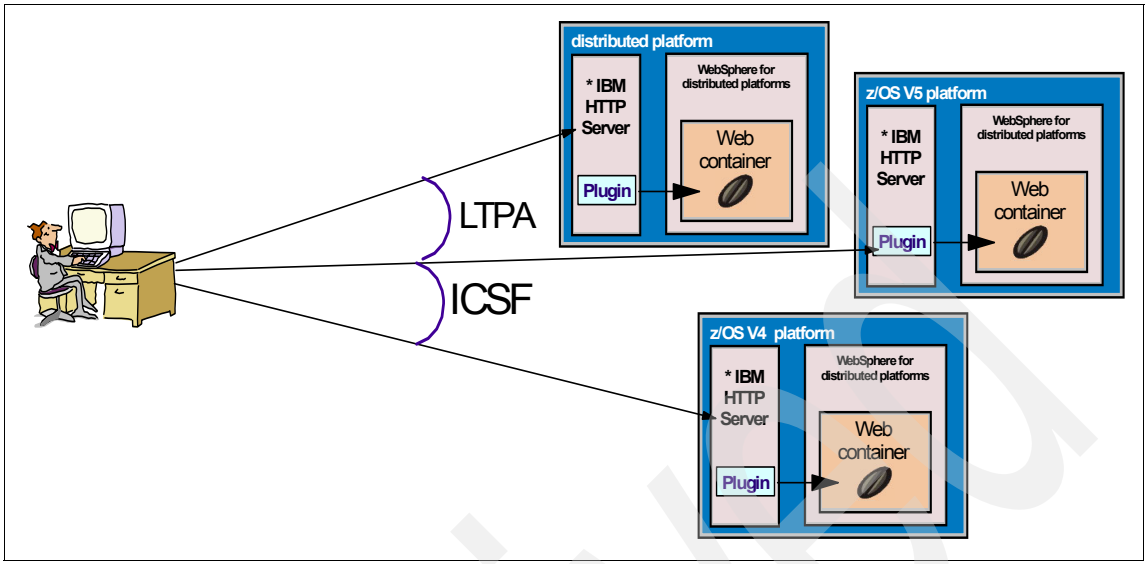


Figure 3-7 Effect of authentication mechanism on single sign-on

For SSO using a distributed server and a WebSphere Application Server for z/OS Version 5 server, choose LTPA for the WebSphere Application Server for z/OS Version 5.

For SSO using a WebSphere Application Server for z/OS and OS/390 V4.0.1 server and a WebSphere Application Server for z/OS and OS/390 V4.0.1 server, choose ICSF.

SSO between a distributed server and WebSphere Application Server for z/OS and OS/390 V4.0.1 is not supported, nor is it possible for one user to have SSO with a distributed server and a WebSphere Application Server for z/OS Version 5 server while another user has SSO with a WebSphere Application Server for z/OS and OS/390 V4.0.1 server and the same WebSphere Application Server for z/OS Version 5 server.

Single sign-on processing

Single sign-on is the process where users provide their user identity and password or client certificate once within a session. This information is then made available to all enterprise applications.

The goal is for an enterprise to be able to have one network identity per user, allowing the centralized management of the various roles the user might have in different applications. Thus, correct rules can be applied without either duplicating user data or requiring multiple identities for one user. In practice,

network identity management is not yet mature enough within most enterprises to achieve a single user registry, particularly when established applications are exposed to the Web. Different application servers have typically implemented their own security or used the operating environment security of the platform on which they have been deployed.

A token is generated by the authenticating server to the Web user. This token is the transient cookie called a *LTPA token*. Transient means that the cookie resides in the browser memory, is not stored on the user's computer system, and expires when the user closes the browser.

The cookie is encrypted using either LTPA keys or ICSF keys that must be shared among all SSO participating servers. It contains user authentication information, the expiration time, and other information.

Tivoli Access Manager, with its reverse proxy security sever WebSEAL, provides a robust mechanism for single sign-on that can be used in conjunction with ICSF or LTPA and a Trust Association Interceptor (TAI). Tivoli Access Manager (Tivoli Access Manager) can integrate most back-end applications servers using the Global Sign-On mechanism to third party user registries or extensions to the Tivoli Access Manager schema to include legacy user identities. When used in conjunction with a TAI on z/OS, the schema need not include the user's SAF password.

The requirements for enabling single sign-on using LTPA or ICSF are:

- All single sign-on participating servers have to use the same user registry (for example, the same LDAP server).
- All SSO participating servers must be in the same DNS domain (cookies are issued with a domain name and will not work in another domain than the one for which it was issued).
- All URL requests must use domain names. No IP addresses or host names are allowed, because this will cause the cookie to not work properly.
- Browsers must be configured to accept cookies.
- Servers' time and time zone must be correct. Single sign-on token expiration time is absolute.
- All servers participating in the SSO scenario must be configured to share LTPA or ICSF keys.

Simple WebSphere Authentication Mechanism (SWAM)

The Simple WebSphere Authentication Mechanism is intended for simple, single application server runtime environments that do not require a distributed security solution. Considerations for SWAM are:

- SSO is not supported.
- SWAM relies on the session ID for form based authentication. It is not as secure as LTPA or ICSF. Therefore, using Secure Sockets Layer (SSL) in combination with SWAM is highly recommended.
- SWAM cannot be used in a Network Deployment configuration.

3.5.2 Java 2 security

This section describes Java 2 security.

Introduction to Java 2 security

For those of us who are familiar with an SAF approach to resource protection, the concepts of Java 2 security might seem foreign. This is because Java 2 security was designed to address a different problem from SAF security. SAF security is intended to protect resources in an environment shared among multiple users with different levels of privilege. Therefore, resources are defined in profiles and users (or groups of users) are permitted to these resources with different levels of authority (such as READ, UPDATE, EXECUTE).

Java 2 security is intended to protect resources in an environment where the executable code might not be trustworthy. A classic example would be where the user of a personal workstation has downloaded code from the Internet.

All Java programs are written as source code and compiled into executable classes that are stored as Java byte code. In order to understand how Java 2 security works, we need to explain what happens when a Java class is loaded and runs.

All Java classes run in an environment called the Java virtual machine(JVM). We can think of the JVM as a “mini” operating system within the context of a z/OS address space. The JVM includes a classloader that brings the classes into memory. The JVM interprets the byte code and translates it into the native machine language of the processor on which it is running. The JVM and certain core classes that it depends on are inherently trusted[1], just as in the z/OS operating system the nucleus and LPALIB modules are trusted.

[1] By saying classes or modules are trusted, we mean that we rely on the source (for example, a respected vendor) or some kind of inspection and control process to provide assurance that the code will not compromise the integrity of the system.

The JVM uses separate class loaders for trusted and entrusted classes. This might be thought of as similar to the way in which z/OS distinguishes between APF-authorized modules and those that are not authorized. When the JVM loads an entrusted class, it first invokes a function called the *Verifier,* which determines whether the class to be loaded adheres to the rules of the Java programming language. No similar function exists natively in the z/OS operating system, as virtually any machine language is legal, at least until it actually executes. In this way, the JVM protects itself from modifications that might have been made after syntactically correct source code was compiled, possibly during transmission over a network or by malicious corruption.

With this as background, we are now able to understand Java 2 security in terms of its three components: security manager, access controller and policy.

Security manager

The security manager is invoked whenever an untrusted class tries to access a defined system resource. It checks the permissions in the policy. Some of the resources that can be protected in a policy are shown in Example 3-6.

Example 3-6 Resources protected by Java security

```
File Access
   canRead(), FileInputStream(), RandomAccessFile(), isDirectory(),
   isFile(), length(), canWrite(), FileOutputStream(), mkdir(), renameTo(),
   createTempFile(), delete(), deleteOnExit()

Network Access
   send(), receive(), getLocalAddress(), getHostName(), getLocalHost(),
   getAllByName()

Java VM
   ClassLoader(), loadLibrary(), checkPermission(), checkLink(), checkExit()

Program Threads
   stop(), resume(), suspend(), interrupt(), setPriority(), setName(),
   setDaemon()

System Resources
   getPrintJob(), setProperty(), getProperty(), setDefault(), getFont(),
   getEventQueue()

Security Aspects
   getFields(), getMethods(), getConstructors(), setPublicKey(),
   addCertificate()
```

Access controller

In order to perform its checking, the security manager invokes another function called the access controller. The access controller examines information about the code making the request and determines whether the policy allows the particular code to access the resource.

During execution, the JVM maintains information about each class in a stack. For each invocation of a method (the Java term for a subroutine call), the JVM pushes an activation record onto the stack. This activation record includes information necessary to determine where the class came from.

When invoked, the access controller traverses the stack, examining each current activation record. All of the currently active classes (those that have been called and have not returned) must satisfy the requirements of the policy in order for the resource access to be granted. Figure 3-8 illustrates the process.

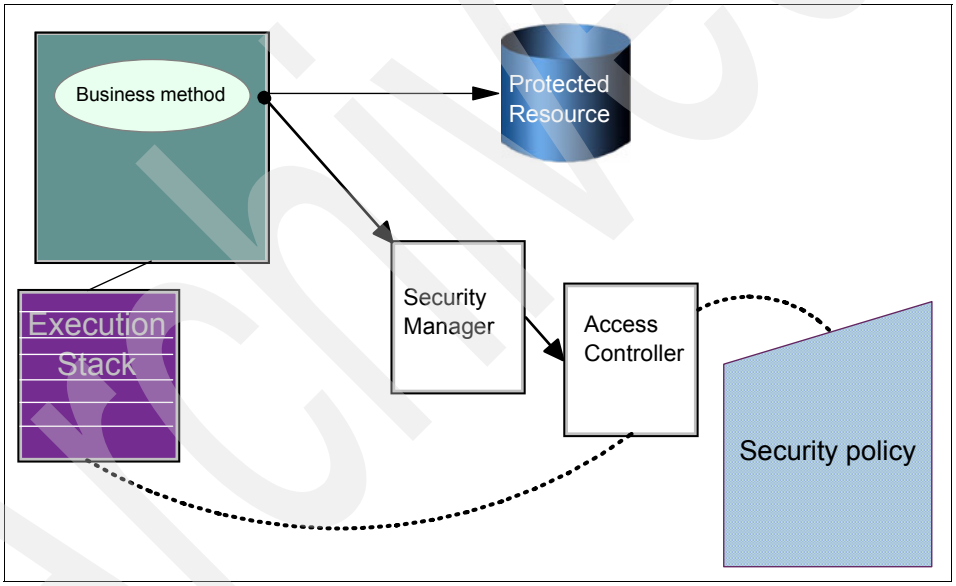


Figure 3-8 Checking access to resources

Why should I enable this function? SAF security already provides classes and profiles that can be used to protect most of the resources protected by Java 2 security. Remember, however, that in many cases the ACEE that your application is running under will carry the identity of the server and not the caller. If you want only certain callers to be able to access a resource, you can protect it with Java 2 security policies. This will require JAAS, described in 3.5.3, “Java Authentication and Authorization Service (JAAS)” on page 74.

Why might I not enable this function? SAF security already provides classes and profiles that can be used to protect many of the resources protected by Java 2 security. Moreover, if an application such as one being migrated from a previous level of WebSphere Application Server is not prepared for Java 2 security, and the application provider has not included a was.policy file or communicated the required permissions, the application is very likely to experience security access control exceptions at runtime. It might be that the application developer is not even aware of what permissions might be necessary. In this case it might be acceptable just to disable Java 2 security at the cell or server level.

Java 2 security policy

For WebSphere Application Server, the Java 2 security policy actually comes from several sources and is controlled by filters or masks that limit what resource access can be granted under the policy. By default, untrusted classes have no access to any resources. Components of the policy grant access to resources based on the origin (or source) of the class. The origin of the class can be determined by the URL designating where the class was loaded from (which in most cases will be the file system path and file name), a digital signature indicating where the code originated, or both.

A class can also be granted permission to access a resource on the basis of a digital signature if it is digitally signed by a private key. The public key of the signer, which is used to verify the signature, is found in a digital certificate, which must be in the keystore specified in the Java 2 security policy.

The Java 2 security policy in effect at any point is determined by the grant entries in the policy files and the dynamic policy (if any), limited by the policy filters.

Policy files

The default permissions granted to applications running in WebSphere Application Server are determined by several policy files. These policy files exist at a cell (for a Network Deployment installation) and server level and would normally be maintained by a WebSphere Application Server administrator. Figure 3-9 on page 71 illustrates these and other policy files and their scope.

Because an erroneous policy file can prevent a server from starting, we suggest maintaining policy files with *policytool*. This tool is not shipped as part of WebSphere Application Server, but as part of the Java Software Development Kit (SDK). It is normally found under <install_root>/bin.

Tip: Policytool requires the Java abstract windowing toolkit (AWT), which requires an X11 window server. This is usually not enabled for z/OS systems unless a remote X11 server has been established on another platform. We maintained all our policy files from a workstation using the SDK installed on the workstation, or by hand-editing where necessary.

Example 3-7 shows a portion of a policy file. This one is from the default app.policy file supplied with WebSphere Application Server.

Example 3-7 Grant statements from app.policy

```
grant codeBase "file:${application}" {
  // The following are required by Java mail
  permission java.io.FilePermission
   "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
  permission java.io.FilePermission
   "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar",
      "read";
};

grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};
```

For a Network Deployment (ND) cell, the cell-level policy file is in <install_root>/<Cell>/DepMgr/java/jre/lib/security/java.policy. (Base Application Servers do not have a cell-level policy file.)

The server level policy file is in <install_root>/<Server>/appserver/java/jre/lib/security/java.policy.

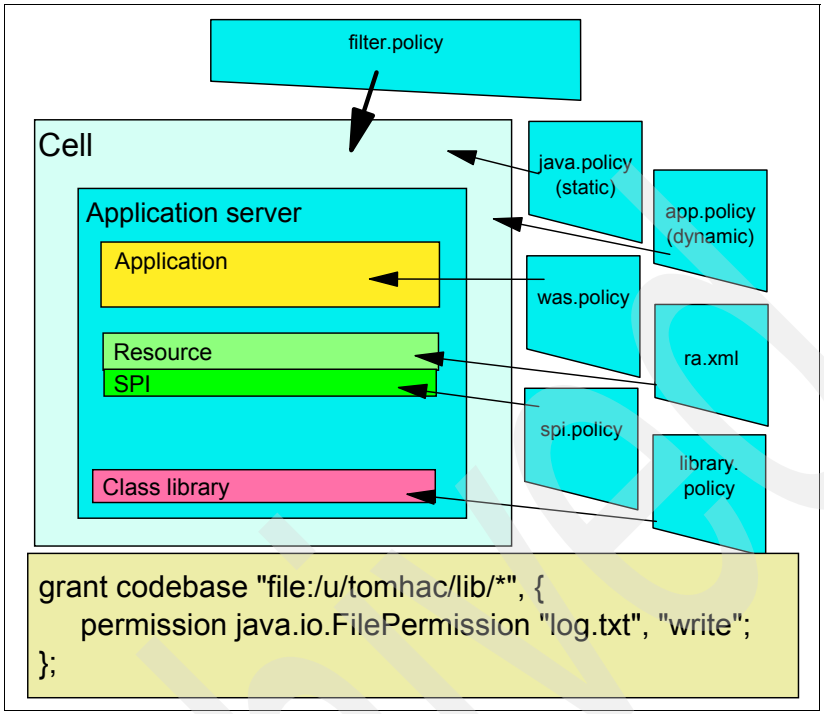


Figure 3-9 Policy files used by WebSphere Application Server for z/OS Version 5

Dynamic policy

In addition to the static policy files, which are in the default policy format provided by the JDK, WebSphere Application Server provides dynamic policy support. In the static policy format, the code base is defined by a URL (which can be a file name) in the policy file. However, in the dynamic format, the code base is calculated at run time and can refer to specific types of resources represented in product-reserved symbols. Table 3-1 on page 72 explains these symbols are explained.

Table 3-1 Reserved symbols in dynamic policy files

Symbol	Meaning
`file:${application}`	Permissions apply to all resources within the application
`file:${jars}`	Permissions apply to all utility .jar files within the application
`file:${ejbComponent}`	Permissions apply to enterprise bean resources within the application
`file:${webComponent}`	Permissions apply to Web resources within the application
`file:${connectorComponent}`	Permissions apply to connector resources both within the application and stand-alone connector resources

Several dynamic policy files are available. The file app.policy contains the default permissions that apply to all enterprise applications in a node and is located at <install_root>/<Server>/appserver/config/cells/<Cell>/nodes/<node>/app.policy. As with cell-level and server-level policies, this policy file would be maintained by a WebSphere Application Server administrator.

The file ra.xml contains connector application-specific permissions for a WebSphere Application Server enterprise connection. It is packaged in a resource adapter archive (RAR) file and is at <rar_file_name>/META-INF/was.policy.RAR. It should be supplied by the connector vendor and should not require modification by the installation.

The spi.policy file defines permissions for Service Provider Interface (SPI) or third-party resources embedded in the product, such as Java Messaging Service (JMS) or Java Database Connectivity (JDBC) drivers. This file is found at <install_root>/config/cells/<cell_name>/nodes/<node_name>/spi.policy. It too should be supplied by the connector vendor and should not require modification by the installation.

The library.policy file defines permissions for Java library classes shared by applications within a node. It is located at <install_root>/config/cells/<cell_name>/nodes/<node_name>/library.policy. Because the library.policy file defines permissions for shared application classes, it is normally be maintained by the applications group.

Applications might have requirements for specific permissions (for example, to read or write an application-specific file or data set) that are not granted in any of the default static or dynamic policies. These permissions could be added to one

of the default policy files, but might not be appropriate for every application. In fact, you probably do not want all applications to have access to files that are meant to be used only by specific applications. So, it is possible for an application developer or assembler to add permissions to a particular application by including a policy file, called was.policy, in the application .ear file. Such a policy file is included in <ear_file_name>/META-INF/was.policy.

It is important to remember that the permissions are cumulative for multiple policy files that apply in a given situation, including was.policy. Permissions are always expressed as grants, never “revoke” or “prevent.” Because the application programmer can add a was.policy file, you might be wondering what prevents an application programmer from erroneously or maliciously giving a program permissions that it should not have. Part of the answer entails vigilance on the part of the installation, but this can be made less onerous by policy filters.

Policy filters

The filter.policy file, which applies to an entire cell, is located at <install_root>/DepMgr/config/cells/<cell_name>/filter.policy (for a Network Deployment cell) or <install_root>/appserver/config/cells/<cell_name>/filter.policy (for a base Application Server). The filter.policy file would normally be maintained by a WebSphere Application Server administrator. The syntax differs from the other policy files in that instead of grant statements, it contains filterMask and runtimeFilterMask statements, which have the effect of removing permissions specified in the app.policy file or was.policy file.

filterMask statements do not apply to compound permissions such as java.security.AllPermission.runtimeFilterMask statements remove even compound permissions from the application.

Policy filtering, with either filterMask or runtimeFilterMask, works only for permissions package names beginning with java or javax.

doPrivileged()

In the section “Access controller” on page 68, we noted that whenever a resource is accessed, the access controller checks all of the activation records in the stack to ensure that the class accessing the resource (and all of its predecessors) have the necessary permissions to access the resource. This creates an insurmountable obstacle to writing utility services, such as logging events to a file. The logging service would need to have permission to write to the file, but you would not want to give this permission to all its callers.

In this case, the logging service can use the doPrivileged() method to virtually mark the stack such that the access controller does not look at any activation

records before the logging service. In other words, only the logging service itself needs to have permission to write to the file.

The effect on the execution stack of doPrivileged() is illustrated in Figure 3-10.

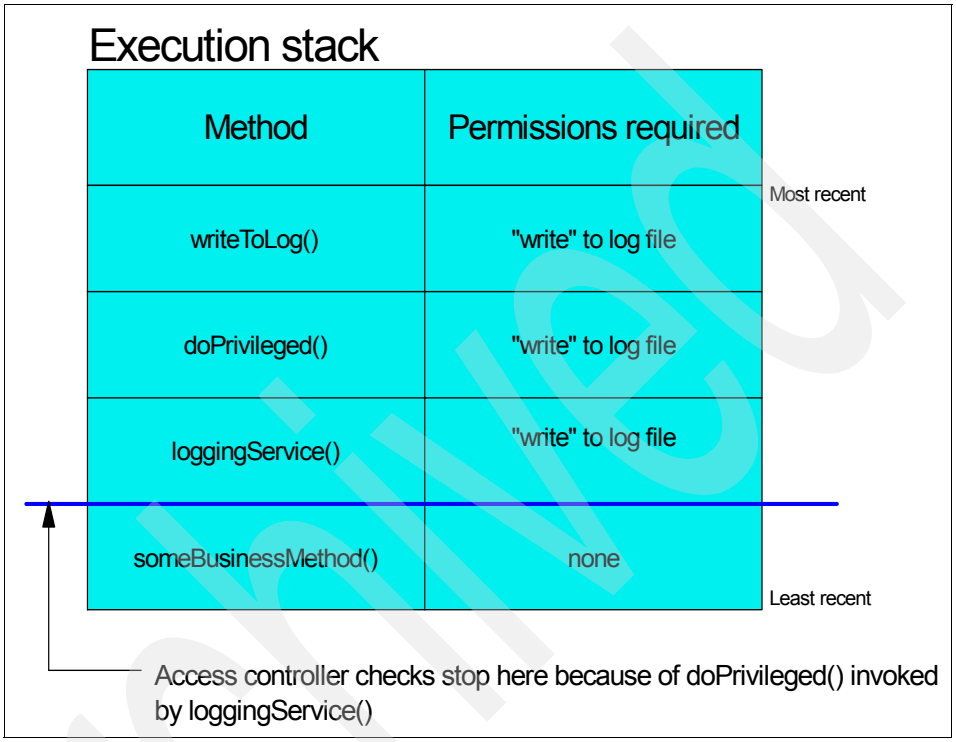


Figure 3-10 Stack after doPrivileged()

Why should I enable this function? It is good programming practice to limit access to shared or critical resources to the smallest unit of code that requires access. By encapsulating access to resources in a doPrivileged() block, only the invoker of doPrivileged() and the methods invoked from the doPrivileged() block require permission to access the resource. Otherwise, it would be necessary to expose the resource to access from any code that might call the service that requires access to the resource.

3.5.3 Java Authentication and Authorization Service (JAAS)

This section introduces Java Authentication and Authorization Service (JAAS) at a conceptual level. JAAS provides Java APIs to Pluggable Authentication Modules (PAM) and provides the strategic APIs for programmatic authentication,

replacing the SSOAuthenticator classes that are now deprecated (although still supported) in WebSphere Application Server Version 5. In addition to supporting PAM and providing a programming interface to applications, JAAS underlies the authentication mechanisms of the WebSphere Application Server container.

The key functions of the JAAS APIs are to perform programmatic login and to invoke methods using the identity from the login. Programmatic login with JAAS APIs results in a Java subject which is populated with principals and credentials resulting the user ID that logged in. The doAs() and doAsPrivileged() methods allow work to run under the resulting subjects.

JAAS also extends Java 2 security by allowing privileges to be associated with a particular subject. The doAS() method of the JAAS Subject performs work as a particular subject, that is, with the permissions that have been granted to that subject.

If the logic executed within the scope of the doAs() were to invoke an EJB, the subject passed in the Subject.doAs() would not be available to the EJB. This surprising result is due to an oversight in JAAS 1.0, the details of which are beyond the scope of this discussion. In order to overcome this problem, WebSphere Application Server has developed a wrapper of the Subject class called WSSubject. Instead of using the doAs() method of the Subject class, the doAs() method of the com.ibm.websphere.security.auth.WSSubject class is used instead. In this case, if the doAs() method invokes an EJB, the subject created by the JAAS login is retrieved and methods such as getCallerPrincipal() return the principal representing the user ID passed to the JAAS login.

doAs() versus runAs

Because both doAs() and runAs change the identity associated with the context, we illustrate the key differences between them in Table 3-2.

Table 3-2 run-as() versus doAs()

	run-as	**doAs()**
Type	declarative	programmatic
Scope	method (runAs) or component (run-as)	run() method
Values	caller (default) role server (runAs only)	arbitrary Subject (or WSSubject)
Propagated?	yes	WSSubject only

Login configuration

The processing that takes place when a JAAS login is encountered depends on settings, called JAAS login configurations, that can be altered and maintained by a WebSphere Application Server administrator using the administrative console. Any number of configurations can be defined, and each *loginContext* is associated with a particular configuration. Login processing associated with WebSphere Application Server authentication mechanisms use the default WSLogin configuration.

Figure 3-11 illustrates a loginContext being created with an installation-defined login configuration called MyLogin.

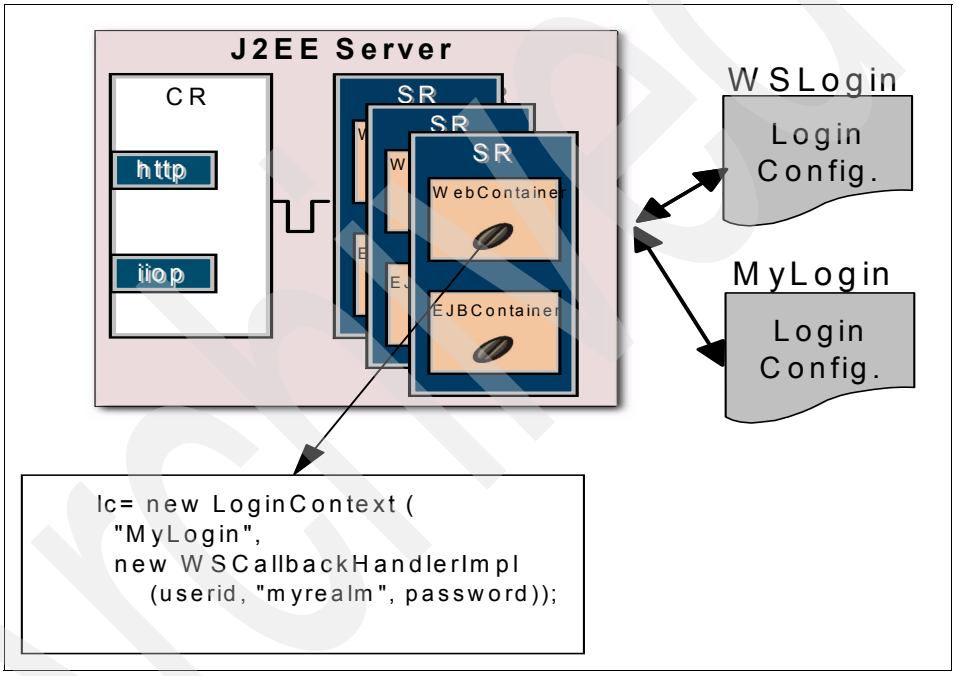


Figure 3-11 Multiple login configurations

Each login configuration consists of one or more login modules and an authentication strategy for each.

Note: An *authentication strategy* refers to how the results of a login module are interpreted, as given in the authentication strategy values below. This is completely independent from an authentication mechanism (discussed in 3.5.1, “Authentication mechanism and single sign-on (SSO)” on page 62) or an authentication method (discussed in 3.2, “J2EE container-based security” on page 46).

Why should I enable this function? The most common use of JAAS is programmatic authentication. While the J2EE programming model encourages declarative security, there are times when the needs of the application go beyond what the programming model is able to offer. Some applications, for example, might include functions that require authorization by more than one individual (for example, to override the usual currency maximum for a transaction). JAAS provides a standard programming model for authentication whose implementation (user registries, authentication mechanisms) can be configured in detail by the installation.

3.5.4 Additional programmatic login/logout capabilities

In addition to form-based authentication (see “Web container authentication” on page 50), there are additional programmatic authentication capabilities available in WebSphere Application Server, based on J2EE specifications and on IBM extensions to J2EE.

Custom login

There are situations when the login facility, using the j_security_check servlet provided by WebSphere Application Server, does not fulfill all requirements for the application. In this case, developers can extend the login facility and develop an extension to the existing code. We recommend using servlet filters as needed for pre-login or post-login processes. The servlet filters can perform some pre-login functions.

Form-based logout

One of the IBM extensions to the J2EE specification is the *form-based logout*. After logging out, the user is required to re-authenticate in order to have access to protected resources again. This logout form can be on any page, calling a POST action on the ibm_security_logout servlet.

3.5.5 Cryptographic application and data security

While identification, authentication, and authorization are sufficient to protect applications from being accessed by unauthorized callers (and some of the techniques, such as SSL, can also provide privacy), non-repudiation and message integrity for data flowing between the client and the J2EE server or between two J2EE servers, there are situations where these approaches are either insufficient or inappropriate.

In some cases, for example, the cost of using SSL (which encrypts entire messages in both directions) might be undesirable where only isolated fields of

data need to be protected. In other cases, an application might need to communicate with an external resource that does not reside in a J2EE server and is not available through a J2EE connector.

For these situations, the J2EE 1.3 specification requires products to support two additional specifications: Java Security Socket Extension (JSSE) and Java Cryptography Extension (JCE).

See 9.2, “How WebSphere fits in z/OS and zSeries cryptographic infrastructure” on page 242 for more details about this matter.

4

WebSphere Application Server application security

One of the goals of the Java 2 Platform Enterprise Edition (J2EE) security architecture is isolating the application developer from issues of security. “Application component providers should not have to know anything about security to write an application.”[1] In 3.2, “J2EE container-based security” on page 46, we discuss ways in which the Web container and the EJB container can provide security without application code having to be written.

Not all applications, however, can be designed in such a way that the application programmer is completely isolated from issues of security. This chapter describes J2EE and WebSphere Application Server capabilities available to application programs that require function that declarative security alone cannot provide.

This chapter discusses the following:

- Programmatic security
- JAAS for authentication and authorization

[1] *Java 2 Platform Enterprise Edition Specification, v1.3*, section J2EE3.3.1.

4.1 Programmatic security

Programmatic security includes any actions taken by application program code to authenticate users, test for authorization to resources, or change the effective user in the current execution context.

WebSphere Application Server offers opportunities for applications to perform programmatic security in all of these areas.

4.1.1 J2EE APIs

J2EE provides an API with two methods for the Web container and two methods for the EJB container.

Web applications use these methods:

- getUserPrincipal
- isUserInRole

EJB applications use these methods:

- getCallerPrincipal
- isCallerInRole

Obtaining the user ID

Methods getUserPrincipal() and getCallerPrincipal() (respectively) are used to retrieve the user ID associated with the current Web application (servlet or JavaServer Page) or EJB:

getUserPrincipal() For applications running in a Web container, this method, defined on HTTPServletRequest, can be used to retrieve a principal object for the user identity. The user ID can be retrieved from this principal object.

getCallerPrincipal() For applications running in an EJB container, this method, defined on EJBContext, can be used to retrieve a principal object for the caller's identity. The user ID can be retrieved from this principal object.

The application can use the returned user ID or principal object for decisions in its program flow. For example, a program might have a private authorization protocol and use the principal name to search an authorization table.

Another possibility would be if data is being retrieved from a relational database, and the principal name forms part of a key field. In a bean-managed persistence (BMP) EJB, we might see code like the fragments shown in Example 4-1 on page 81.

Example 4-1 Using getCallerPrincipal() to retrieve data from a relational database

```
. . .
String ejbUser = getEntityContext().getCallerPrincipal().getName();
. . .
ps = jdbcConn.prepareStatement("insert into myTable (name)" + "values(?)");
ps.setString(1,ejbUser);
if(ps.executeUpdate() !=1 ){
   throw new CreateException . . . .
   }
```

Testing for role access

Methods isUserInRole() and isCallerInRole() (respectively) can be used to check whether the current user has been given access to a certain role. Based on the result, the program can make decisions in its processing path.

isUserInRole() For applications running in a Web container, this method, defined on HTTPServletRequest, can be used to test if the authenticated user ID is authorized to the role specified in the argument.

isCallerInRole() For applications running in an EJB container, this method, defined on EJBContext, can be used to test if the user ID that called the method is authorized to the role specified in the argument.

An example of how isUserInRole() might be helpful is as follows. You might have an application that has versions in various stages of test (such as function, user acceptance, pre-production). Each version has a unique URI (such as /functionTest/myApp, /acceptanceTest/myApp) and each is accessed by a different set of users. The users should not have to know the URIs.

One approach is to write a routing servlet that sends each user to an appropriate version of the application (assuming each user should access one and only one version). An example of this type of code is shown in Example 4-2.

Example 4-2 Using roles to drive program logic

```
String redirector = null;
if (req.isUserInRole("functionTest")) {
   redirector = "functionTest/";
} else if (req.isUserInRole("acceptanceTest")) {
   redirector = "acceptanceTest/";
}
if (redirector != null) {
   resp.sendRedirect(redirector + "myApp");
   }
```

Important! Although Example 4-2 illustrates one potential use of isUserInRole(), the example does not actually provide meaningful application security. Instead, it provides convenience and avoids unintentional errors for a group of trustworthy users. The use of redirection merely obscures the actual URL from the end user temporarily. The end user will ultimately be able to see the actual URL, and nothing in the example prevents the end user from substituting the URL of a different version from the one he or she is supposed to be authorized to.

A more secure solution would be to have the routing servlet call session EJBs over a local interface, thereby preventing direct access from a remote client.

4.1.2 Programmatic authentication to resources

As discussed in 3.3, “Resource authentication” on page 58, a J2EE component’s deployment descriptor specifies access to back-end resources through a resource-ref element. Within the resource-ref element, the source of authenticating information used by the resource is specified in a res-auth subelement.

A res-auth value of *container* indicates declarative authentication, while *application* indicates programmatic authentication. With programmatic authentication, the application provides a user ID and password on the connection specification or on the getConnection() call, for example, `getConnection(userid, password)`.

The challenge in using programmatic authentication for resources is obtaining the password. If programmatic authentication is used for other components (for example, a Web application uses a form to prompt the end user for a user ID and password and then performs a JAAS login), the password can be passed as an argument from one method to another, so it is available for the method that finally calls the resource.

Alternatively, the getUserPrincipal() or getCallerPrincipal() methods can be used to obtain the user ID, and a table can be maintained to find the password.

Why might I not enable this function? We do not recommend using programmatic authentication for resources. Doing so requires a user ID and password to be available to the application code. In the case of common IBM subsystems (such as DB2 or CICS), this would currently be a SAF user ID and password. An important best practice for SAF security is never to store a clear text password in a file or application-accessible memory.

We recommend using declarative security techniques, such as run-as, in conjunction with a container res-auth value to allow the container to securely sign on to the resource without a clear text password.

When your security infrastructure does not support an integrated security solution, you might be forced to investigate programmatic authentication.

4.2 JAAS for application security

This section expands on the concepts introduced in 3.5.3, “Java Authentication and Authorization Service (JAAS)” on page 74. JAAS provides Java APIs to Pluggable Authentication Modules (PAMs) and provides the strategic APIs for programmatic authentication, replacing the SSOAuthenticator classes that are now deprecated (although still supported) in WebSphere Application Server Version 5. In addition to supporting PAM and providing a programming interface to applications, JAAS underlies the authentication mechanisms of the WebSphere Application Server containers.

The key functions of the JAAS APIs are to perform programmatic login and to invoke methods using the identity from the login. Programmatic login with JAAS APIs results in a Java subject which is populated with principals and credentials resulting in the user ID that logged in. The doAs() and doAsPrivileged() methods allow work to run under the resulting subjects.

doAs()

The function of the doAs() method is to perform some work as a particular subject, that is, with the permissions that have been granted to principals that are part of that subject.In Example 4-4 on page 84 we see a code fragment that performs a JAAS login (in the method doJAASLogin()) and uses the resulting LoginContext (in the method tryLogin()) to retrieve the subject and use it to invoke a Subject.doAs() method.

In order to perform the JAAS login and invoke the doAs() method, we had to grant permissions to our code beyond those provided in the default setup. We added the grant statements shown in Example 4-3 on page 84 to our app.policy file.

Example 4-3 Grant statements for JAAS login and doAS()

```
grant codeBase "file:/WebSphere/BSO6/appserver/installedApps/-" {
  permission javax.security.auth.AuthPermission "createLoginContext";
  permission javax.security.auth.AuthPermission "doAs";
};
```

Note: Although, with the necessary permissions (createLoginContext and doAS), this kind of logic could be performed within an application, we recommend that it be included in a separate utility library and be carefully monitored by security aware WebSphere Application Server administrators. Applications that require this type of function should call a utility method rather than imbed security functions directly.

Example 4-4 doAs() example

```
import javax.security.auth.Subject;
import javax.security.auth.login.*;
import com.ibm.websphere.security.auth.*;
import com.ibm.websphere.security.auth.callback.*;
. . .
   private LoginContext doJAASLogin(String userid, String password) {
      /* The following JAAS login code is copied, with modifications, from the
         InfoCenter. */
      LoginContext lc = null;

      try {
         lc =
            new LoginContext(
               "WSLogin",
               new WSCallbackHandlerImpl(userid, "realm", password));
      } catch (LoginException le) {
         System.out.println(
            "Cannot create LoginContext. " + le.getMessage());
         // insert error processing code
         le.printStackTrace();
      } catch (SecurityException se) {
         System.out.println("Cannot create LoginContext." + se.getMessage());
         // Insert error processing
         se.printStackTrace();
      }
      try {
         lc.login();
      } catch (LoginException le) {
         System.out.println("Fails to create Subject. " + le.getMessage());
         // Insert error processing code
         le.printStackTrace();
      }
```

```
      return lc;
   }
. . .
   public String tryLogin(String userid, String password) {
      /**
       * This method performs a JAAS Login, creating a Subject.  It then calls
       * EJBSample within a doAs() block to discover what principal was used.
       */
      LoginContext lc = doJAASLogin(userid, password);
      javax.security.auth.Subject mySubject = lc.getSubject();
      String result = (String)
         Subject.doAs(mySubject, new java.security.PrivilegedAction() {
         public Object run() {
            return readFile();
         }
      });
      return result;
   }
```

The code that is to be run is passed as an argument to the doAs() method. In this case it is declared as part of the invocation. The doAs() method returns the object returned by the PrivilegedAction's run() method, which in this case will be the result of the readFile() invocation.

WSSubject

If the logic executed within the scope of the doAs() were to invoke an EJB, the subject passed in the Subject.doAs() would not be available to the EJB. This surprising result is due to an oversight in JAAS 1.0, the details of which are beyond the scope of this discussion. In order to overcome this problem, WebSphere Application Server has developed a wrapper of the Subject class called WSSubject. Example 4-5 provides a code fragment showing how the WSSubject can be used.

Example 4-5 Use of WSSubject.doAs()

```
   public String WSlogin(String userid, String password) {
      /**
       * This method performs a JAAS Login, creating a WSSubject.  It then
       * calls readFile() within a doAS() block to read a file that only
       * the logged in user can access. */
      LoginContext lc = doJAASLogin(userid, password);
      javax.security.auth.Subject myWSSubject = lc.getSubject();
      String result = (String)
         com.ibm.websphere.security.auth.WSSubject
            .doAs(myWSSubject, new java.security.PrivilegedAction() {
         public Object run() {
            return readFile();
```

```
            }
        });
        return result;
    }
```

Note that the code is almost identical to that in Example 4-4 on page 84 except that instead of using the doAs() method of the Subject class, the doAs() method of the com.ibm.websphere.security.auth.WSSubject class is used. In this case, if the getResult() method invokes an EJB, the subject created by the doJAASLogin() method are retrieved and methods such as getCallerPrincipal() return the principal representing the user ID passed to the doJAASLogin() method.

Figure 4-1shows how the Subjects are handled when using WSSubject.doAs().

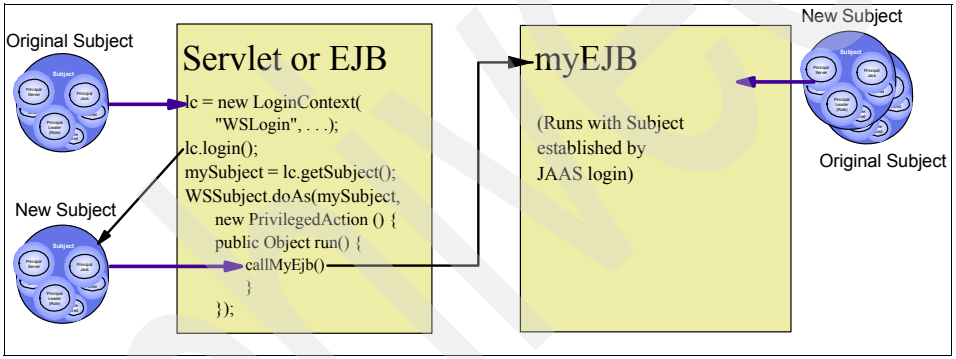


Figure 4-1 Effect of WSSubject.doAs()

The lc.login() method invocation creates a new Subject, but it is not the effective Subject in the execution context. This is accomplished by WSSubject.doAs(), which makes the new Subject the current Subject for the scope of the run() method, which includes the call to myEJB. During that period, the new Subject is effective in the execution context. The original Subject still exists, but is not accessible through the available APIs.

Login configuration

The processing that takes place when a JAAS login is encountered depends on settings, called JAAS login configurations, that can be altered and maintained by a WebSphere Application Server administrator using the administrative console. Any number of configurations can be defined, and each *loginContext* is associated with a particular configuration. Login processing associated with WebSphere Application Server authentication mechanisms use the default WSLogin configuration.

Figure 3-11 on page 76 illustrates a loginContext being created with an installation-defined login configuration called MyLogin. Each login configuration consists of one or more login modules and an authentication strategy for each.

Note: An *authentication strategy* refers to how the results of a login module are interpreted, as given in the authentication strategy values below. This is completely independent from an authentication mechanism (discussed in 3.5.1, “Authentication mechanism and single sign-on (SSO)” on page 62) or an authentication method (discussed in 3.2, “J2EE container-based security” on page 46).

The login modules are called in the sequence listed until success or failure is determined. The login module contains the logic for authenticating the user ID and password values in the loginContext. The authentication strategy is a string consisting of one of the following values:

Required The login module must authenticate the user ID and password or the request fails, regardless of what other login modules do.

Requisite The login module must authenticate the user ID and password or the request fails immediately, without calling any other configured login modules.

Sufficient If no Required or Requisite login module has failed, and this login module successfully authenticates the user ID and password, the login is successful without trying any further login modules.

Optional If there are no Required or Requisite login modules, this login module might or might not authenticate the user ID and password. Processing continues with the next configured login module. At least one login module must successfully authenticate the user ID and password for the request to succeed.

4.2.1 JAAS login verification using SWIPE

The SWIPE example demonstrates the use of JAAS. 7.6.3, “SWIPE: EJBCaller - Input Part C, JAAS” on page 179 describes how to use the SWIPE JAAS example.

Using SWIPE you can select to do a JAAS login, and can supply the four parameters required for a JAAS login, as shown in Table 7-11 on page 180.

The SWIPE example does a JAAS login using the values you supply, and then invokes a method on the EJBSample EJB using both the

javax.security.auth.Subject.doAs and the com.ibm.websphere.security.auth.WSSubject.doAs approaches.

WebSphere supplies an application login configuration for use with JAAS called WSLogin.

This supplied mechanism validates a user ID/password when a JAAS login is done against the user registry that global security in WebSphere has been configured to use.

To verify a programmatic JAAS login, we invoked the SWIPE secure EJBCaller servlet and logged on as user USRMGR.

We then filled in the JAAS input fields as shown in Table 4-1.

Table 4-1 JAAS login input fields

Field	Value
Logon module	WSLogin
Realm	realm
Userid	USRCEO
Password	USRCEO

We then selected **Yes** to have SWIPE do a JAAS login and pressed **Submit**.

The browser output shows what happened with regard to the JAAS login under the heading JAAS Report.

The display in Figure 4-2 on page 89 shows the result after the EJBCaller servlet has done the JAAS login and called a method in the EJBSample EJB using the javax.security.auth.Subject.doAs approach.

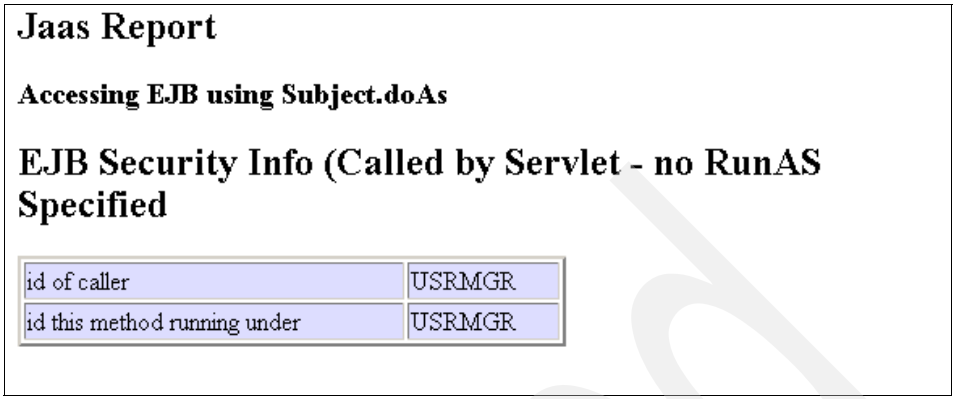


Figure 4-2 SWIPE: JAAS login, Subject.doAs result

Figure 4-2 shows that the user ID of the caller is still USRMGR, the user ID that was used for the initial authentication.

The display in Figure 4-3 shows the result after the EJBCaller servlet has done the JAAS login and called a method in the EJBSample EJB using the com.ibm.websphere.security.auth.WSSubject.doAs approach.

4.2.2 Your own JAAS application login configuration

You do not have to use the supplied WSLogin application login configuration, you can develop and use your own. You could use the SWIPE example to verify the behavior of a login configuration. After defining the new login configuration entry to WebSphere Application server, you can enter the name of it in the JAAS Login field called Logon Module.

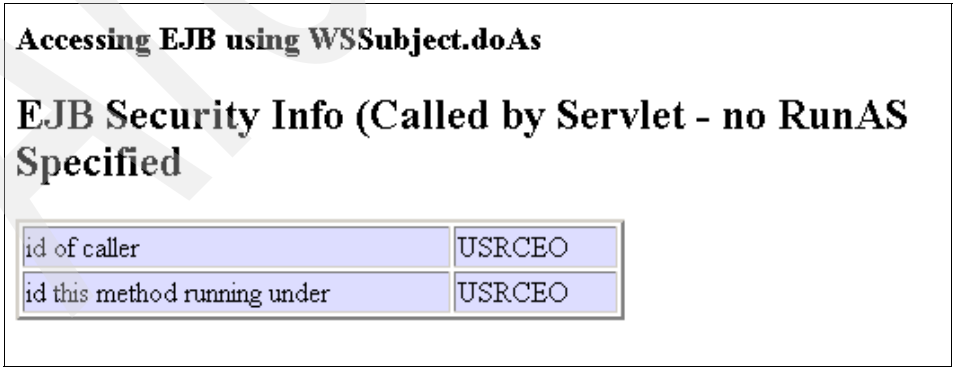


Figure 4-3 SWIPE: JAAS login, WSSubject.doAs result

Figure 4-3 on page 89 shows that the user ID of the caller is now USRCEO, the user ID with which the JAAS login was done.

It also shows that if you go through the effort of creating a JAAS login, but then just use the javax.security.auth.Subject.doAs method to invoke some process, and then the user ID used will still be the user ID that was originally derived by the authentication process.

If your intent of doing a JAAS login is to have the user ID you signed on with used when invoking further processes, the use of the javax.security.auth.Subject.doAs method will not achieve this. You must use com.ibm.websphere.security.auth.WSSubject.doAs.

5

WebSphere application migration security aspects

This chapter describes the security aspects that might be present when you migrate WebSphere applications to WebSphere Application Server for z/OS Version 5. Refer to *Migrating WebSphere Applications to z/OS*, SG24-6521for a more detailed discussion of all migrating aspects. We have identified the most likely problems one could encounter and describe potential solutions.

This chapter discusses the following topics:

- Thread level based security
- Security aspects when migrating Common Connector Framework (CCF) connectors
- Migrating SOMDOBJS to EJBROLE

5.1 Application migration security aspect checklist

Applications that might show security-related problems when migrated to WebSphere Application Server for z/OS Version 5 come typically from the following environments:

- Applications that are called by code running in the WebSphere Application Server V4 local redirector plug-in and using Web server protect rules to control access to operating system resources or to provide authentication. The local redirector plug-in is no longer available; the new WebSphere HTTP plug-in does not propagate identities authenticated by the Web server.
- Applications running in the WebSphere Application Server for z/OS and OS/390 V4.0.1 local redirector plug-in directly (also known as the “simple configuration option”) when security is based on IBM HTTP Server thread-level security capabilities. The WebSphere Application Server V4 local redirector plug-in is no longer available as a runtime for Web applications.
- Applications running in WebSphere Application Server 3.5 using IBM HTTP Server’s protect rules to control access to Operating System resources. Security is now defined in deployment descriptors, operating system resources are accessed with the servers identity.
- Applications running in WebSphere Application Server for z/OS and OS/390 Web container secured by RACF SOMDOBJS profiles. The WebSphere Application Server 3.5 *authresource* property in the deployment descriptor that triggers SOMDOBJS authorization checking is not supported in WebSphere Application Server for z/OS and OS/390 V4 and later.
- Applications running in any IBM HTTP Server based environment that are using the Web servers capability to retrieve user information from an HFS-based password file. Typically these non RACF users are mapped to a RACF SURROGAT user ID, while the authenticated user ID has no access rights in the OS environment. WebSphere does not automatically honor third party authenticated users. User mapping needs to be established within the Web container. Access to operating system resources will take place with the server’s identity.

5.2 Application migration strategies

When security features used in the past are no longer available, you need to migrate to the security features provided by the Web container as defined by the J2EE standards:

- The Web container provides security using EJBROLES, Java 2 security, and JAAS. Applications running in the plug-in need to be migrated to run in a Web container and have appropriate security configured. Refer to *Migrating WebSphere Applications to z/OS*, SG24-6521, for details about how to migrate your application to a modern Web container.
- If you are protecting your Web Applications SOMDOBJS, you need to migrate to using EJBROLES as described in 5.7, “Migrating SOMDOBJS to EJBROLE” on page 99.
- If you are using a password file and a surrogate user ID, we recommend that you investigate RunAs role. This will run a method with the user ID associated with the authorized EJBROLE profile. You can still use a password file to authenticate users, or you could create RACF user IDs with no TSO segment, no OMVS segment, and a non-expiring password.

The limitation of these scenarios is that operating system resources are accessed using the user ID of the Web container servant and not the authenticated user. This is discussed in more detail in 5.3, “Migrating IBM HTTP Server thread level-based security” on page 93.

5.3 Migrating IBM HTTP Server thread level-based security

IBM HTTP Server enables you to specify protection setups for all incoming requests. It also enables you to configure an executing user to which the server will change its identity when serving the request. Any Web container running in this address space would also run under the user ID selected by the protection setup. The result is that actions performed by the Web container would execute under a specified user ID, thus enabling thread-level security. See Figure 5-1 on page 94.

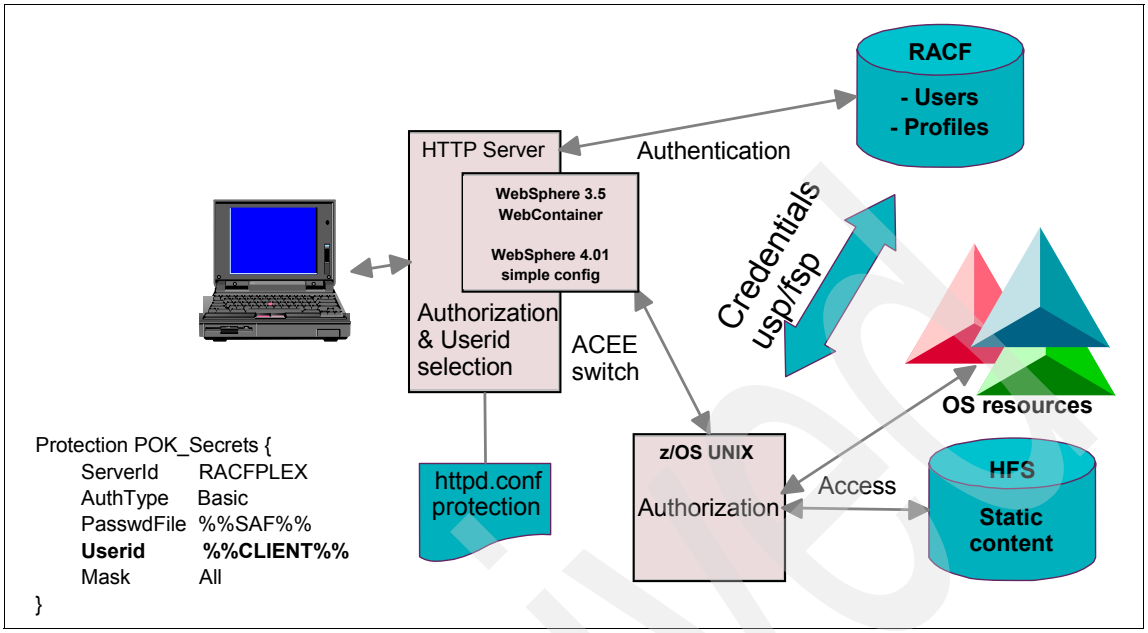


Figure 5-1 Protection setup-based operating system resource access

Access to operating system resources, such as HFS files, is now controlled by RACF, z/OS UNIX File Security Package (FSP), and User Security Package (USP).

Example:

```
Protection POK_Secrets {
ServerId        ZOSHTTPServer
AuthType        Basic
PasswdFile      %%SAF%%
Userid          %%CLIENT%%
Mask            All
}
Protect /Kaaterskill/* POK_Secrets %%CLIENT%%
```

This will not work in WebSphere Application Server V5 because the Web application is running in an address space with a fixed identity from a z/OS perspective

5.3.1 Affected environments

Applications that are impacted by the configuration changes are using Web server protect rules to control access to operating system resources and running in the following environments:

- WebSphere Application Server V4 local redirector plug-in
- WebSphere Application Server V3.5 applications

A typical scenario is a servlet that reads files in the HFS or simply Web content that is protected by z/OS UNIX. A protect rule would be configured to force authentication. Authentication would be done by RACF, and an effective user ID would be selected for the executing thread. The selected user ID would have access to the operating system resources, not the server.

5.3.2 What is causing this problem?

The WebSphere Application Server for z/OS and OS/390 V4.0.1 simple config option is not supported in a WebSphere Application Server for z/OS Version 5 environment. The WebSphere Application Server V3.5 environment is no longer a preferred environment. Applications should be migrated to a real Web container and run there.

5.3.3 How can you make it work again?

Your application needs to be migrated to a Web container, and then an appropriate security scenario needs to be implemented.

We are going to base our discussion on an application that needs to be able to access operating system resources. We can configure RACF profiles to protect the resources, which will allow access on an individual or group basis. We then use EJBROLES, RunAs Caller, and SyncToOSThread to enforce access decisions based on those RACF profiles.

How this can be achieved with the WebSphere Application Server for z/OS and OS/390 plug-in or from an EJB is shown in Figure 5-2 on page 96. We make the following observations about the diagram:

- The IBM HTTP Server plug-in is WebSphere Application Server for z/OS V3.5 or the simple configuration option. It shows a servlet running in the plug-in. The servlet readOSR is configured by protect rules in IBM HTTP Server to run under its caller's user ID.
- EJBROLES access profiles are defined for srvlt1 when they are deployed. This protects whoever is allowed to access the application itself.

- The EJBs readOSR1 and readOSR2 use SyncToOSThread and RunAs Caller. They are running in a WebSphere Application Server for z/OS and OS/390 EJB container. The caller has been authenticated and has a valid z/OS RACF user ID. Therefore, the operating system resource is accessed using the caller's ID:
  - The EJBs are protected by EJBROLEs. This protects who has access to specific resource types.
  - The physical access to the individual resource is controlled by RACF profiles against the caller's RACF user ID. This controls who has access to a specific resource (DATASET, HFS file, and so on).

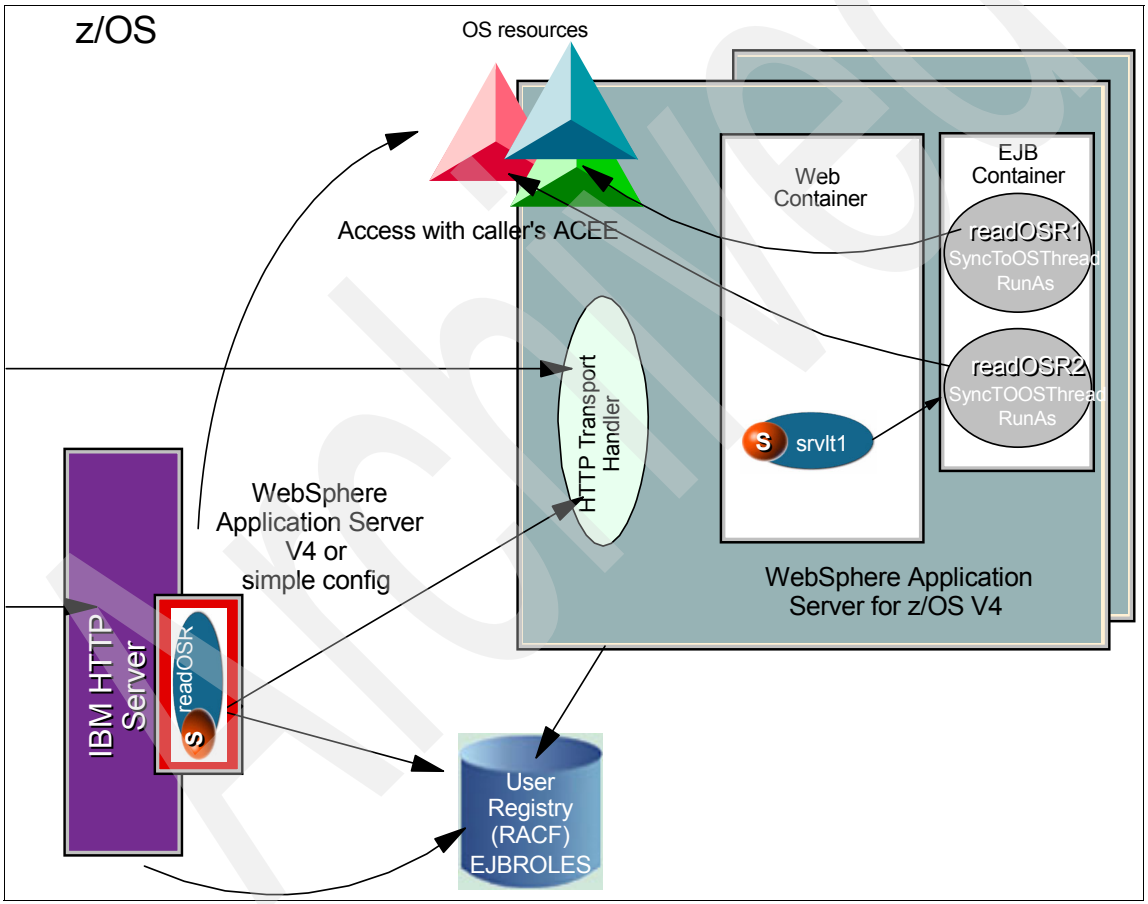


Figure 5-2 A typical WebSphere Application Server V3.5 or V4 scenario

5.4 Migrating WebSphere Application Server thread level-based security

WebSphere Application Server for z/OS and OS/390 V4.0.1 provided the SyncToOSThread setting that associated the current RunAs identity OS/390 with the thread Task Control Block (TCB). This replaced the current user ID of the WebSphere for z/OS server region, with that of the current RunAs identity. This allowed operating system resources to be accessed using the user ID of an authenticated caller.

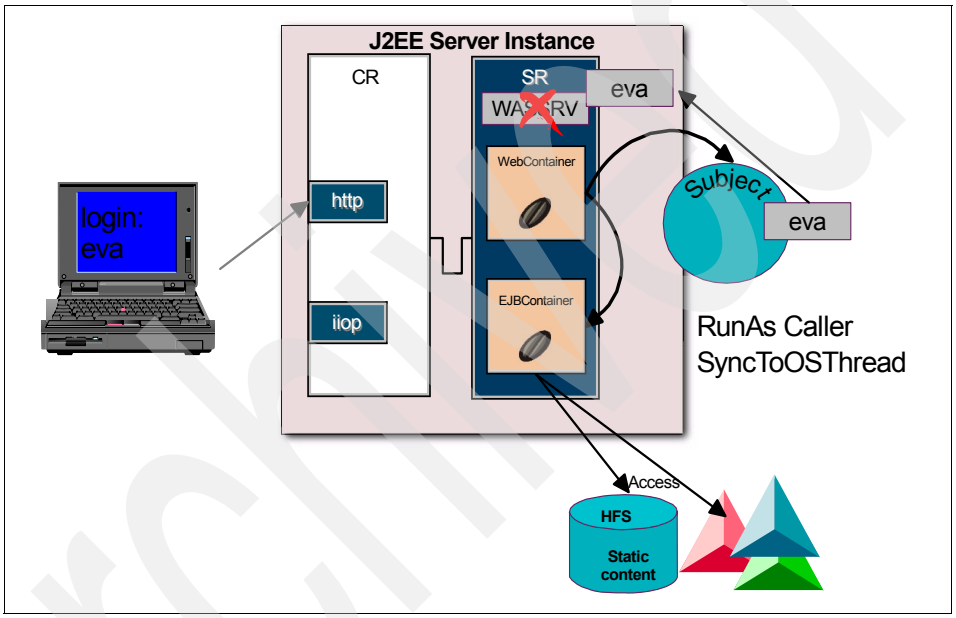


Figure 5-3 SyncToOSThread support for operating system resource access

This function is being deprecated. IBM did not deliver SyncToOSThread on method descriptors in the initial V5.0 availability. This function was delivered through the service stream through PTF UQ88257 (APAR PQ81584) in the W502008 Service Level.

Affected environments

Applications using SyncToOSThread and RunAs Caller to get access to operating system resources using the caller's user ID are affected.

A typical scenario would be that a J2EE container changes its identity (ACEE) based on RunAs and SyncToOSThread settings and uses the derived z/OS USERID for access to operating system resources such as files or sockets.

SyncToOSThread is not required when using a J2CA connector to access an EIS under the user ID of the current Java thread.

5.5 Security aspects when migrating Common Connector Framework (CCF) connectors

CCF connectors are not supported inWebSphere Application Server V5

5.5.1 Affected environments

The affected environments are applications using CCF connectors. CCF applications run in:

- WebSphere Application Server V4 local redirector plug-in (simple config option)
- WebSphere Application Server V3.5

5.5.2 What is causing this problem?

CCF connectors are being deprecated. Security in CCF connectors is based on thread-level security. It uses SyncToOSThread, which is available with PTF UQ88257 (Service Level W502008).

5.5.3 How can you make it work again?

Migrate your application to use J2CA connectors. This is described in *Migrating Applications from WebSphere for z/OS V4 and V3.5 to V5*, SG24-7044.

If you cannot migrate at this time, make sure that your system is upgraded to W502008 and enable SyncToOSThread in the application deployment descriptor. Also, be sure that the cell has Sync to OS Thread Allowed. This can be verified through the administrative console by navigating to **Global Security** and clicking **z/OS Security Options** under Additional Properties.

5.6 Security aspects when migrating J2CA connectors

We found no security issues when migrating J2CA connectors to WebSphere Application Server V5.

5.6.1 Affected environments

The affected environments are J2EE 1.2 applications using J2CA connectors.

5.6.2 What is causing this problem?

J2EE 1.2 applications are supported in WebSphere Application Server V5, but we recommend that they be migrated to J2EE 1.3.

5.6.3 How can you make it work again?

We imported a J2EE application into WebSphere Studio Application Developer and used its migrate function to migrate to J2EE 1.3. This is described in detail in the section “Migrating a J2EE 1.2 Ear file to 1.3 using WSAD” in *Migrating Applications from WebSphere for z/OS V4 and V3.5 to V5*, SG24-7044.

We had no problems doing this. The application ran successfully in WebSphere Application Server V5. The CTF/CCI code is compatible between v4 and v5. We continued to use RACF as the registry. In WebSphere Application Server V5 it is now possible to LDAP, but we kept our migration simple, and had no problems.

More information about J2CA connector security is presented in *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064.

5.7 Migrating SOMDOBJS to EJBROLE

In previous versions of WebSphere Application Server for OS/390, Web components could be secured using the System Authorization Facility (SAF) SOMDOBJS class. This section discusses migrating that security to the J2EE security model using the SAF EJBROLES class.

5.7.1 Using SOMDOBJS with WebSphere simple configuration option

This section describes the SOMDOBJS setup you might need to migrate, as explained in 5.7.2, “Migrating from SOMDOBJS to the Web container and the EJBROLE profiles” on page 101.

Early in our residency, we went as a team on a hike to Kaaterskill Falls, the tallest waterfall in New York. We took some pictures on our hike and wanted to make them available to the team, but not to the public, because we were not all wearing our most flattering outfits. So we published them to the intranet.

At that time, we were using the V4.0.1 WebSphere for z/OS local redirector plug-in within the HTTP Server address space as a runtime environment for our

Web applications. This provides a way to run non-J2EE-compliant Web applications in an environment that is identical to the environment provided by WebSphere Application Server for OS/390 Version 3.5. It is currently not available in WebSphere Application Server V5.

Example 5-1 shows the configuration statements used to define our Web application. The last one establishes a security constraint for the pictures, which are contained in the Pix directory:

```
deployedwebapp.Kaaterskill.authresource.KaatMapping=/Pix/*
```

Example 5-1 was.conf statements for the Kaaterskill Web application

```
host.KaatHost.alias=wtsc59.itso.ibm.com:80
deployedwebapp.Kaaterskill.host=KaatHost
deployedwebapp.Kaaterskill.rooturi=/Kaaterskill
deployedwebapp.Kaaterskill.classpath=/usr/lpp/was35/AppServer/hosts/def
deployedwebapp.Kaaterskill.documentroot=/usr/lpp/was35/AppServer/hosts/
deployedwebapp.Kaaterskill.autoreloadinterval=0
deployedwebapp.Kaaterskill.authresource.KaatMapping=/Pix/*
```

In order to ascertain which user was accessing our Web page, we caused the Web server to authenticate the request. This was accomplished through a protection setup in httpd.conf using the directives shown in Example 5-2.

Example 5-2 Protection setup for the Kaaterskill application

```
Protection POK_Secrets {
        ServerId        ZOSHTTPServer
        AuthType        Basic
        PasswdFile      %%SAF%%
        Userid          %%CLIENT%%
        Mask            All
}
Protect /Kaaterskill/*                  POK_Secrets wtsc59.itso.ibm.com
```

If the user TOT08, who is not permitted access to the pictures, tries to look at them, that person will receive the text of the HTML page, but no pictures. Because we are using RACF, the following message also appears at the operator's console:

```
ICH408I USER(TOT08   ) GROUP(CBADMGP ) NAME(SMEUI ADMINISTRATOR )
  KAATHOST.KAATERSKILL.KAATMAPPING CL(SOMDOBJS)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

In order to permit certain users to access the pictures, we created a profile in the SAF SOMDOBJS class. The form of the profile was <virtual-hostname>.<webapp-name>.<resource-name>.

In our case, we created it using this command:

```
RDEF SOMDOBJS (KAATHOST.KAATERSKILL.KAATMAPPING) OWNER(SYS1) UACC(NONE)
```

We were then able to permit users to issue HTTP GET requests for the protected resources, using, for example:

```
PERMIT KAATHOST.KAATERSKILL.KAATMAPPING CL(SOMDOBJS) ACC(R) ID(TOMHAC)
```

This was followed by:

```
SETR RACLIST(SOMDOBJS) REFRESH
```

In order to allow the HTTP PUT or DELETE methods, we would have to allow UPDATE and ALTER access, respectively.

5.7.2 Migrating from SOMDOBJS to the Web container and the EJBROLE profiles

This section discusses the steps necessary to implement the same level of protection described in 5.7.1, “Using SOMDOBJS with WebSphere simple configuration option” on page 99 using J2EE roles and EJBROLE profiles. We do not describe all of the steps necessary to migrate the application from previous versions of WebSphere Application Server. This information is available in *Migrating Applications from WebSphere for z/OS V4 and V3.5 to V5*, SG24-7044.

We performed the following steps:

1. We removed the following directive from httpd.conf:

   ```
   Protect /Kaaterskill/* POK_Secrets wtsc59.itso.ibm.com
   ```

   This is no longer necessary, because the Web container performs authentication. Note that we left the POK_Secrets protection setup in place, because it is referenced by other Protect directives.

   Note: was.conf is not used by the Web container.

2. We imported our .war file into WebSphere Studio Application Developer. For a detailed presentation on the steps involved, refer to *Migrating Applications from WebSphere for z/OS V4 and V3.5 to V5*, SG24-7044. In summary:
   a. Go to the Web perspective.
   b. Do a **File** → **Import** → **War file**.
   c. The “Import resources from war file” window opens. Enter the .war file name: Kaaterskill.war, the project name: Kaaterskill, and the enterprise application project name: KaaterskillEar. Click **Next**.

Note: Although we performed these steps using WebSphere Studio Application Developer, you can accomplish the same results with any tool, such as the Application Assembly Tool (AAT), which is capable of creating or editing an .ear file file with J2EE-compliant deployment descriptors.

3. Next, we need to define:
   - Security role
   - Security constraints
   - Authentication method

   Add the security role: Hiker

   a. Switch to the J2EE perspective.

   b. In the J2EE Hierarchy view, select **kaaterskill** Web Deployment Descriptor.

   c. Select **Open With** → **Deployment Descriptor Editor** from the context menu.

   d. On the Security page of the editor, click **Add**. The Add Security Role wizard opens.

   e. Type `Hiker` and a description for the new role.

   This is shown in Figure 5-4.

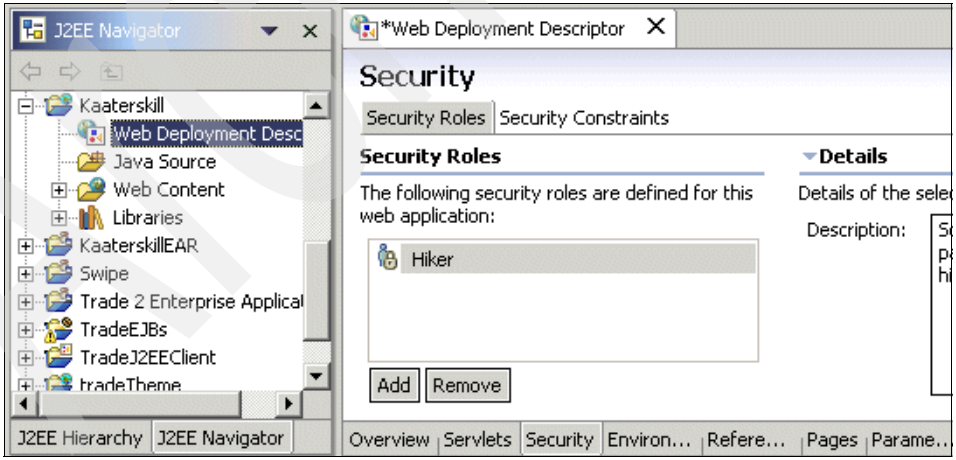


Figure 5-4 Adding EJBROLE: Hiker

4. Next, we created a security constraint that we called Hiker's constraint. Under the Authorization tab, we selected our Hiker role name, meaning that users

who are assigned to this role will be able to access resources that are subsequently defined in a Web resource collection.

a. Click the Security tab for the Web Deployment Descriptor.

b. Then click the **Security Constraints** tab at the top of the panel. Click **Add**.

c. Type the security constraint name and description.

d. Add roles required by clicking **Edit** under Authorized Roles.

e. A new window pops up.

f. Select the **Hikers** role and add a description.

This is shown in Figure 5-5.

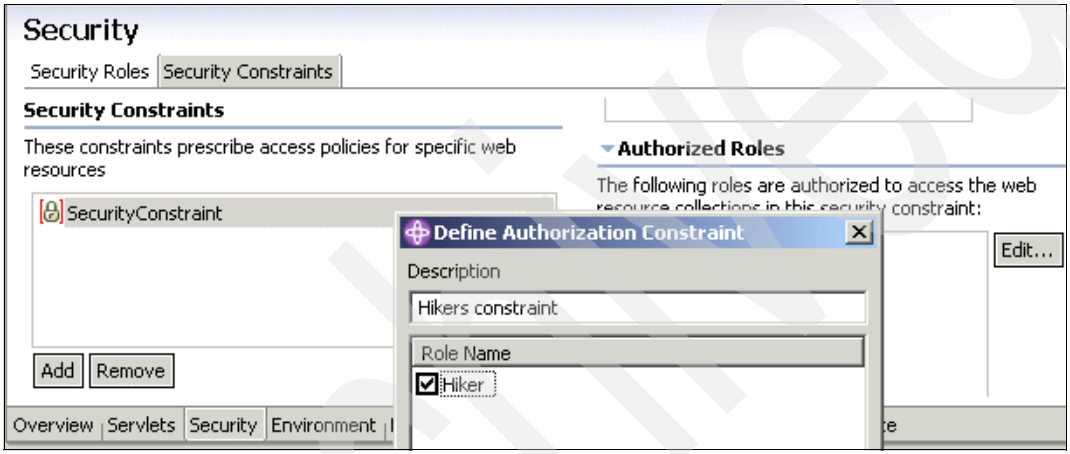


Figure 5-5 Adding a security constraint

After defining a security constraint, we needed to identify the resources to be protected by it. A set of resources is called a Web resource collection. Because we wanted to protect the pictures in our application, we defined a Web resource collection with the name Kaaterskill.

1. Click **Edit** in Web Resources Collection. A new panel displays.
2. Type a Web resource collection name and description.

   The next step in specifying the Web resource collection is to determine which HTTP methods are to be covered within the scope of this collection. In other words, we specify which of the HTTP methods for URLs ending in /Pix/* can be requested by the security roles selected in the constraint (Hiker's constraint). We allow the Hiker role to request the GET method.
3. Select the methods allowed: **get**.

After the Web resource collection was defined, we needed to specify what to protect. Note that the specification begins with a slash (/) and does not include the context root.

4. Click **Add** next to URL Patterns and type the URL pattern.

   This is shown in Figure 5-6.

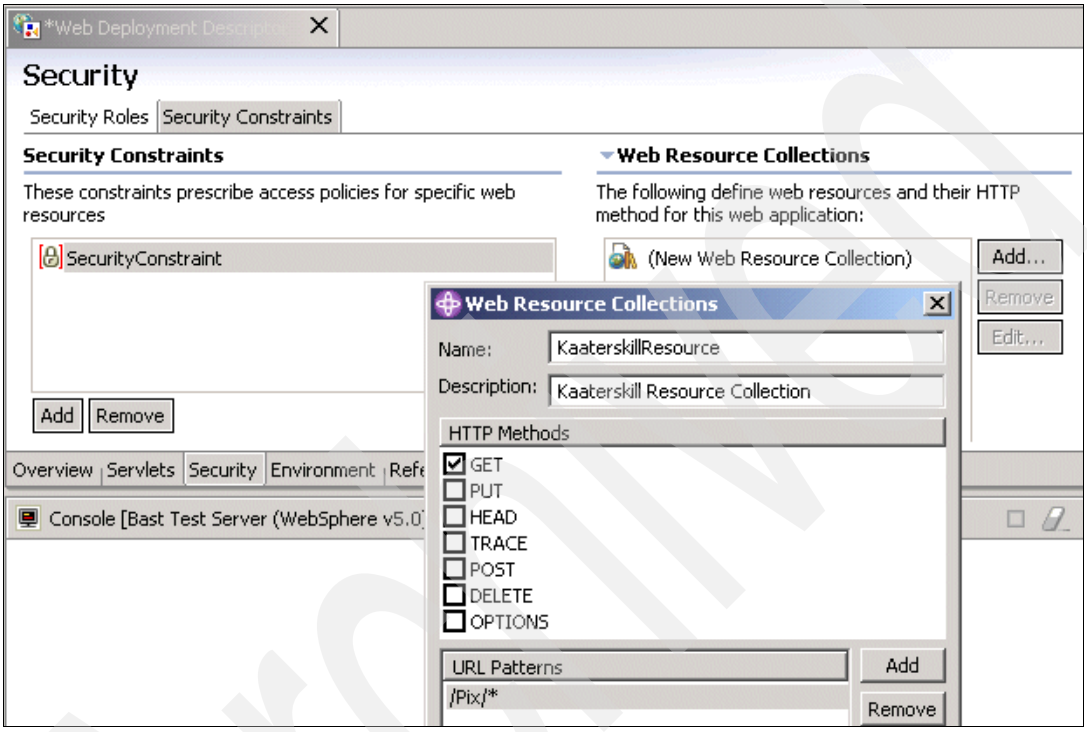


Figure 5-6 Defining a Web Resource Collection

Note: If you do not select the box for an HTTP method, that method will not be included in the Web resource collection, and unless the method is covered by some other Web resource collection in this or some other security constraint, it will be available to all users.

The exception to this rule is that if no methods are selected, all methods matching the URLs specified in the Web resource collection will be subject to the constraint. In other words, selecting no boxes is equivalent to selecting all the boxes.

5. Configure the login mechanism for the Web Module:

   a. Click the **Page** tab on the right navigation panel.

   b. Select the **Authentication method** option. Select the **Basic** authentication method from the drop-down list.

   This is shown in Figure 5-7.

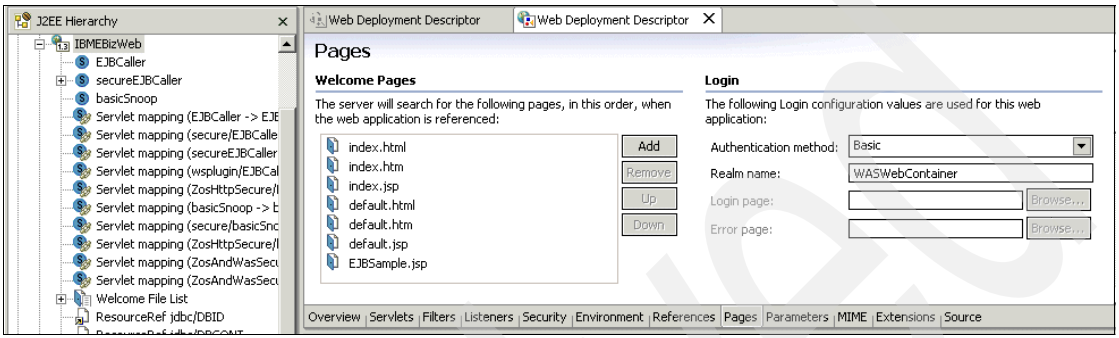


Figure 5-7 Specifying basic authentication

6. Close the Web Deployment Descriptor and save it.

   This completes the activities that take place in WebSphere Studio Application Developer. Example 5-3 shows our deployment descriptor, WEB-INF\web.xml.

Example 5-3 Deployment descriptor with security constraints and roles

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp">
   <display-name>Kaaterskill</display-name>
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>KaaterskillResource</web-resource-name>
         <description>Kaaterskill Resource Collection</description>
         <url-pattern>
         /Pix/*</url-pattern>
         <http-method>
         GET</http-method>
      </web-resource-collection>
      <auth-constraint>
         <description>Hikers constraint</description>
         <role-name>Hiker</role-name>
      </auth-constraint>
   </security-constraint>
   <login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>wtsc59.itso.ibm.com</realm-name>
```

```
   </login-config>
   <security-role>
      <description>Someone who participated in the hike</description>
      <role-name>Hiker</role-name>
   </security-role>
</web-app>
```

7. The final step was to create the necessary SAF profiles and permissions. We issued the following RACF commands to create the Hiker EJBROLE profile and assign the user TOMHAC to the Hiker role:

```
rdef ejbrole (Hiker) uacc(none)
permit Hiker cl(ejbrole) acc(r) id(tomhac)
setr raclist(ejbrole) refresh
```

As before, when a user attempts to access the pictures on our Web page, the Web browser requests a user ID and password. If the user ID supplied is a valid SAF user, and the password is correct, the user is allowed access to the page. If the user is assigned to the Hiker role, the pictures are displayed. If the user, such as TOT08, is not assigned to the Hiker role, the pictures are not displayed.

In this case, however, we no longer get the ICH408I message displayed at the operator's console. Instead, the following message is displayed:

```
++BBOO022OE SECJ0129E: Authorization failed for wsadmin while invoking
GET on default_host:/Kaaterskill/Pix/DSCN0035.JPG, Authorization
failed, Not granted any of the required roles: Hiker
ICH408I USER(WSADMIN ) GROUP(MISC    ) NAME(WAS ADMINISTRATOR   )
  Hiker CL(EJBROLE )
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

Part 2

SWIPE and our testing infrastructure

In this part, we describe the infrastructure we used to test security. Then, we describe the sample applications we developed to illustrate and understand the many security controls that you can implement in WebSphere Application Server for z/OS Version 5.

6

The sandbox infrastructure

This chapter describes the infrastructure we set up to exploit the security functionality of the WebSphere Application Server for z/OS runtime environment. This infrastructure is not appropriate for use with any other environment nor a recommendation for a secure infrastructure.

6.1 Physical integration into the network infrastructure

Figure 6-1 shows the setup we used to exploit security functionality for this book.

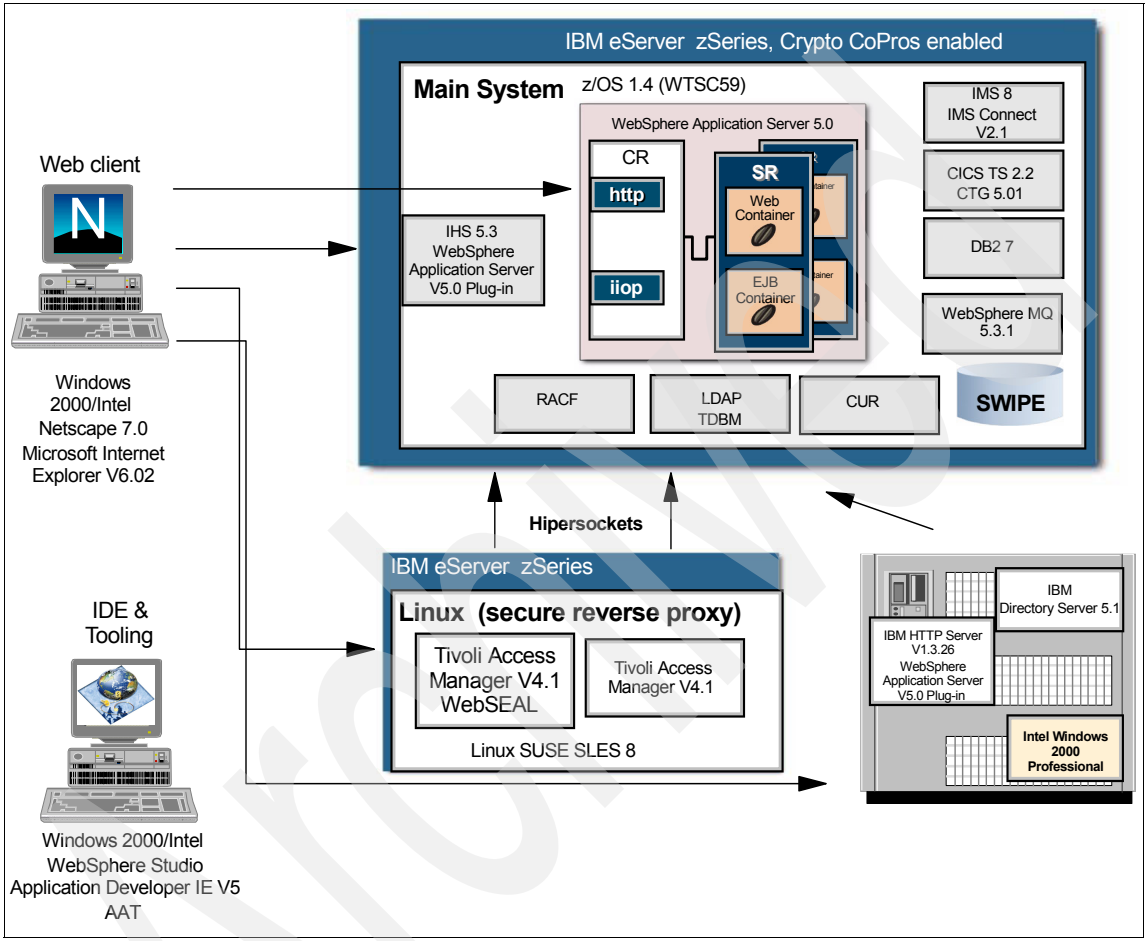


Figure 6-1 Our sandbox

We set up one z/OS 1.4 image with WebSphere Application Server V5 for z/OS, a Linux for zSeries SUSE SLES 8 image with Tivoli products and a Microsoft® Windows® 2000 Server with IBM Directory Server and IBM HTTP Server.

6.2 System setup and service levels

This section describes the infrastructure and service levels we used to develop this redbook.

6.2.1 Operating system and program products

We used the following operating system and program products:

- z/OS Version 1 Release 4
- System SSL with the PTFs UA00954, UA01625, UA01739 and UA01920
- z/OS Version 1 Release 4 Security Server LDAP Server
- IBM Developer Kit for OS/390, Java 2 Technology Edition at the SDK 1.3.1 level PTF level UQ77055
- WebSphere Application Server for z/OS and OS/390 Version 5.0 at PTF level W500103 (UQ79132)

  Note: For subsequent revisions of this book, we used higher maintenance levels. For Version 5.1, we used W501003. Where the maintenance level is important, it is indicated in the text.

- DB2 7.1 (JDBC driver at UQ77539 level)
- IMS 8
- IMS Connect 2.1
- CICS Transaction Server 2.2
- CICS Transaction Gateway 5.01
- WebSphere Message Queue 5.3.1 with the PTFs

6.2.2 Distributed environments

In the distributed environment tests, we configured the following platforms:

SUSE LINUX Enterprise Server for IBM eServer zSeries V8

This Linux® for zSeries system was configured to host:

- Tivoli Access Manager for e-business WebSEAL V4.1
- Tivoli Access Manager for e-business Policy Director V4.1

Microsoft Windows 2000 Professional

One Intel®-based server was set up to host:

- An LDAP server with IBM Directory Server V5.1 (it can be downloaded for free at:

  http://www14.software.ibm.com/webapp/download/search.jsp?rs=ldap&go=y)
- IBM HTTP Server V1.3.26
- WebSphere Application Server for Windows V5.0 plug-in

6.2.3 Development environment

We used the following operating development and assembly tools:

- WebSphere Studio Application Developer Integration Edition Version 5
- Application Assembly Tool

The Application Assembly Tool is provided with WebSphere Application Server for z/OS and OS/390 Version 5. The installation file (setup.exe) can be downloaded from the lib subdirectory of your WebSphere Application Server for z/OS and OS/390 Hierarchical File System target installation directory (/usr/lpp/WebSphere/lib on our system).

6.3 Naming conventions

To show you what you should consider, we describe our naming conventions. WebSphere Application Server for z/OS and OS/390 Version 5 offers more configuration options mixing cells and systems than WebSphere Application Server for z/OS and OS/390 V4.0.1 used. As a result, it is not possible to give a simple “one size fits all” naming convention.

Anyway, we tried to create names and objects that can be a source of inspiration for any installation.

Important: Naming conventions used for this project do not reflect the concept of security domains introduced with IBM WebSphere Application Server for z/OS V 5.02. If you model your RACF naming conventions on our concept, you will have to change the EJBROLE, CB, APPL, and PassTicket profiles to represent the security domain.

SAF subsystem naming conventions and considerations are discussed in 10.4, “SAF naming standards and conventions” on page 297, and WebSphere RACF

conventions are discussed in 10.10.3, “RACF definitions, naming conventions, and considerations for WebSphere environments on z/OS” on page 314.

6.3.1 WebSphere cells

We ran our tests using a monoplex system configuration but we tried to adopt a naming convention that could be used in a multisystem sysplex environment.

Figure 6-2 summarizes our complete IBM WebSphere Application Server for z/OS infrastructure.

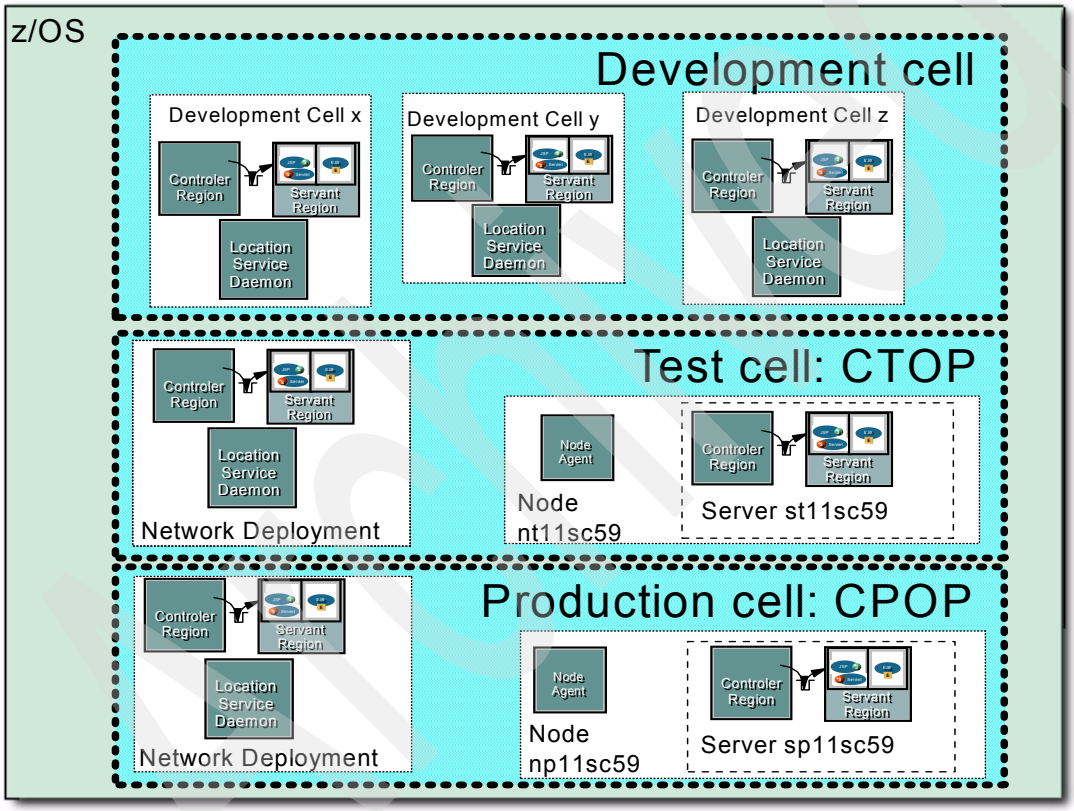


Figure 6-2 WebSphere cells

We decided to use three different environments:

- A test cell managed by its deployment manager.
- A production cell managed by its deployment manager.

- A development cell we called “development virtual cell” because it is a set of base Application Servers sharing the same naming convention and used individually by each team member. In fact, it is a logical group of base server cells, not a “cell” as defined in WebSphere Application Server configuration.

6.3.2 Naming convention variables

The naming convention we chose relies on the following variables:

- Full cell names are four letters long (DEVL,PROD,TEST). They appear with the **c** letter. When it is expressed with one **c**, it indicates the first letter of the cell; when it is expressed with two **c**, it indicates the first two letters of the cell. When the **c** is part of a number, a TCP/IP port for instance, the **c** should be replaced by a logical cell number assigned for each cell (such as DEVL is 0, PROD is 1, TEST is 2).
- Nodes are expressed using **n**, which represents the node index in a cell. It starts at 1 and is not intended to represent a specific system (but you might want to do so). The exception is for development base servers where the node number of the only node is the server number of the only server that is the number of the cell.
- Servers are expressed with **s**, a number that represents the server index in a node. It starts at 1. The exception is for development base servers where the node number of the only node is the server number of the only server that is the number of the cell.
- Our system is defined by four letters and it appears explicitly as `sc59` in some long names. It appears in objects that belong only on a specific system (node- and server-related configuration objects).
- Our sysplex name is `OPPLEX`. It is used in the cell names.

The development cells follow a slightly different convention because the purpose of these cells is to act as base servers that will never be federated. It offers a second advantage in that they can easily be separated from the other types of servers, in SDSF for instance, and do not interfere with production-oriented cells.

In definitions, “managed servers” refers to servers we defined with the ISPF dialogs and which were to be federated in a deployment manager cell. Some of the data that needs to be entered in ISPF will not survive federation.

6.3.3 Data sets and files

Table 6-1 on page 115 shows the conventions used for data set and file names.

Table 6-1 Data sets and files

	Dev. base server	Deployment manager	Managed server
Installation Variable data sets	WAS.V500.**cccc**.CELL 0**s**.SC59.CNFSAVE	WAS.V500.**cccc**.DM.S C59.CNFSAVE	WAS.V500.PROD.**ccns** .SC59.CNFSAVE
Installation dialog output	WAS.V500.**cccc**.CELL 0**s**.SC59.CNTL/DATA	WAS.V500.**cccc**.DM.S C59.CNTL/DATA	WAS.V500.**cccc**.**ccns**. SC59.CNTL/DATA
HFS mount points	/WebSphere/BS0**s**	/WebSphere/**cccc**	/WebSphere/**ccns**
HFS data set	OMVS.WAS.V500.SC5 9.BS0**s**.HFS	MVS.WAS.V500.SC59. **cccc**.DM.HFS	OMVS.WAS.V500.SC5 9.**cccc**.**ccns**.HFS
Logstream	WAS.V500.SC59.BS0**s** .LOG	WAS.V500.SC59.**cccc**. DM.LOG	WAS.V500.SC59.**ccns**. LOG
Component trace	WAS.V500.SC59.BS0**s** .CTRACE	WAS.V500.SC59.**cccc**. DM.CTRACE	WAS.V500.SC59.**ccns**. CTRACE

6.3.4 Component trace procedure names

The following conventions were used for component trace procedure names:

Dev. base server CTRBS0s

Deployment manager CTRcccc

Managed server CTRccns

6.3.5 Configuration objects

Table 6-2 shows the conventions used for WebSphere configuration objects.

Table 6-2 WebSphere configuration objects

	Development base server	Network Deployment	Future managed servers
WebSphere Application Server home directory	appserver	DeplMgr	appserver
Short cell name	CEL**cs**	CL**c**	CL**cns**
Long cell name	opplex[a]	c**c**op	c**cns**
Short node name	NOD**cs**	ND**c**DM	ND**cns**
Long node name	nt**cs**ct**cs**sc59	n**c**dmsc59	n**cns**sc59

	Development base server	Network Deployment	Future managed servers
Short app name	WAS**cs**	WS**c**DM	WS**cns**
Long app name	wt**cs**nt**cs**ct**cs**sc59	s**c**dmsc59	s**cns**sc59
Cluster	CLU**cs**	CL**c**DM	CL**cns**

a. Usually, associating a sysplex name with a cell is not a good idea. Because a sysplex can host different cells, it is better to have some name that describes the cell itself and not only where it is hosted.

6.3.6 Development base servers started tasks and user IDs

Table 6-3 shows the conventions used for the development base servers started tasks and user IDs.

Table 6-3 Development base servers started tasks and user IDs

	Controller region	Servant region	Location service daemon
Job name	WASD**c**	WASD**c**S	WASDD**c**
Procedure name	WAS5DC**c**	WAS5DS**c**	WAS5DD**c**
User ID	WDC**c**STU	WDS**c**STU	WDD**c**STU
UID	253**c**	263**c**	271**c**

6.3.7 Deployment manager started tasks and user IDs

Table 6-4 shows the conventions used for the deployment manager started tasks and user IDs.

Table 6-4 Deployment manager configuration started tasks and user IDs

	Controller region	Servant region	Location service daemon
Job name	WS**c**DM	WS**c**DMS	WS**c**DMD
Procedure name	WS5**c**DMC	WS5**c**DMS	WS5**c**DMD
User ID	W**c**CDMSTU	W**c**SDMSTU	W**c**DDMSTU
UID	shared	shared	shared

6.3.8 Node agent started tasks and user IDs

Table 6-5 shows the conventions used for the node agent started tasks and user IDs.

Table 6-5 Node agent started tasks and user IDs

Job name	WS**cn**N
Procedure name	WS5**cn**NC
User ID	W**c**C**n**NSTU
UID	Shared

The **addNode.sh** command does not offer parameters to set up the node agent. As a result, these settings need to be changed from the ones the deployment manager set when adding the node.

6.3.9 Managed servers started tasks and user IDs

Table 6-6 shows the conventions that were used for the managed servers started tasks and user IDs.

Table 6-6 Managed server STCs and user IDs

	Controller region	**Servant region**	**Location service daemon**
Job name	WS**cns**	WS**cns**S	WS**cns**D
Procedure name	WS5**cns**C	WS5**cns**S	WS5**cns**D
User ID	W**c**C**ns**STU	W**c**S**ns**STU	W**c**D**ns**STU
UID	Shared	Shared	Shared

6.3.10 TCP/IP ports

Table 6-7 shows the conventions that were used for the TCP/IP ports.

Table 6-7 TCP/IP ports

	Base servers	**Deployment manager**	**Node agent[a]**	**Managed servers**
SOAP JMX Connector	888**c**	**c**8800	**c**88**n**0	**c**88**ns**
DRS Client	787**c**	**c**7800		**c**78**ns**

	Base servers	Deployment manager	Node agent[a]	Managed servers
ORB	280**c**	**c**9800	**c**28**n**0	**c**28**ns**
ORB SSL	0	0		0
HTTP	908**c**	**c**9000		**c**90**ns**
HTTP SSL	944**c**	**c**9400		**c**94**ns**
Location Service Daemon	555**c** [b]	**c**5500		**c**55**ns**
Location Service Daemon SSL	565**c**	**c**5600		**c**56**ns**
Cell Discovery		**c**7200		
Node Discovery			**c**72**n**0	
Bootstrap			**c**29**n**0	

a. The **addNode.sh** command does not offer parameters to set up TCP/IP for the future node agent, so the node agent port setting needs to be changed from defaults after adding the node.
b. Except for node 5, which will be 15555, and for node 6, which will be 15556 (ports reserved for WebSphere Application Server for z/OS and OS/390 V4.0.1).

6.3.11 Common information

We chose the following objects to be the same for all of our servers:

- Component trace user ID and default group
- Configuration group
- Initial administrator user
- Servant group
- Unauthenticated user

6.3.12 Starting servers

To start the servers on our configuration, we used the commands that follow.

Base server

The console command we used to start a base server was:

```
START WAS5DCs,JOBNAME=WASDs,ENV=CELDs.NODDs.WASDs
```

Where **s** is the server identifier.

There is a way to simplify the command by editing the control region STC and hard-coding the ENV parameter:

```
//WAS5DC5  PROC ENV=CELD5.NODD5.WASD5,PARMS='',Z=WAS5DC5Z
//  SET ROOT='/WebSphere/BSO5'
//BBOCTL  EXEC PGM=BBOCTL,REGION=0M,
// PARM='TRAP(ON,NOSPIE),ENVAR("_EDC_UMASK_DFLT=007") / &PARMS.'
//BBOENV DD PATH='&ROOT/&ENV/was.env'
//  INCLUDE MEMBER=&Z
```

With this modification, the START command can be limited to:

```
START WAS5DCs,JOBNAME=WASDs
```

Deployment manager

The console command we used to start a deployement manager was:

```
START WS5cDMC,JOBNAME=WScDM,ENV=CLc.NDcDM.WScDM
```

Where **c** is the cell type.

Node agent

The console command we used to start a node agent was:

```
START WS5cn1C,JOBNAME=WScnN,ENV=CLc.NDcn1.WScnN
```

Managed server

The console command we used to start a federated application server was:

```
START WS5cnsC,JOBNAME=WScns,ENV=CLcns.NDcns.WScns
```

7

The security investigation application

In this chapter we introduce you to our security investigation application. We also explain the underlying security concepts very briefly. In-depth explanations of these concepts can be found in corresponding chapters of this book.

You can download our security investigation application from the Internet to check if your security settings deliver the expected results. See “Locating the Web material” on page 717 for downloading instructions. If you play around with this application, you can very quickly get a feel for security within your environment and begin to understand how the whole process works, from writing the application to deploying it all the way to z/OS and OS/390.

7.1 The SWIPE application

To enable us to test the various security scenarios, we developed an application that we named SWIPE, an acronym that stands for Security in WebSphere Investigation Program Example. We used WebSphere Studio Application Developer to develop the application, which consists of servlets and EJBs.

SWIPE was originally developed for the previous version of this book, SG24-6846, which dealt with WebSphere Application Server for OS/390 V4. For that version, it was at the J2EE 1.2 level. For this book, which is for WebSphere Application Server V5, the version is at the J2EE 1.3 and EJB 2.0 levels.

The following sections provide an overview of the features that SWIPE demonstrates. You need to have a basic understanding of J2EE security concepts to understand the functionality of SWIPE. If you are unfamiliar with J2EE security concepts, review Chapter 3, “J2EE 1.3 and WebSphere Application Server V5 security concepts” on page 41.

Servlet authentication[1]

The SWIPE application consists of a main servlet that can be invoked either with or without authentication. When invoked with authentication, this can be done using any of the following items:

- Basic
- Forms
- Client certificate
- run-as settings
- Programmatic security

EJB authorization[2]

The SWIPE application also consists of a session EJB with various remote methods defined. The aim here is to demonstrate:

- EJBROLEs
- Declarative security
- RunAs settings
- Programmatic security

[1] Authentication is the process by which an entity (for example, a server) verifies the identity presented by a user. This process usually requires some form of credential information such as a password known only to the user.

[2] Authorization is the process of determining what type of access (if any) the security policy allows to a resource (for example, file or method) by a principal.

A number of methods in the EJB are defined with different RunAs settings. They can be run in any combination to enable you to examine how the principal user ID associated with a process and the user ID of the caller of that method change.

File access

The SWIPE application enables you to access a file. Using this, you can see which user ID is used to access operating system resources outside the WebSphere environment.

Servlet run-as example

A servlet has been added to SWIPE to demonstrate how the run-as facility for a servlet functions. This capability is described in 16.4, “Web container identity delegation and propagation” on page 556.

JAAS

SWIPE enables you to do a JAAS logon. This demonstrates the concepts of JAAS described in 4.2, “JAAS for application security” on page 83.

Remote EJB

Another aspect of J2EE programming can be to invoke an EJB from an EJB. One of the features of EJBs is that the EJB being invoked can be in the same or a different WebSphere location. The SWIPE application can be installed into any number of WebSphere servers and provides a way for you to specify where the remote location of a second SWIPE application is. When you do, the EJB called by the servlet invokes methods on the remote EJB and provides some information as to what occurred.

Sync to OS Thread

WebSphere Application Server V4 supported a concept called SyncToOSThread. This concept allowed access to a resource outside of the WebSphere started task to be the user ID under which a servlet or EJB was running. The SWIPE program developed for the WebSphere Application Server V4 version of this book had several examples of this capability. This function is being deprecated. IBM did not deliver SyncToOSThread on method descriptors in the initial Version 5.0 availability. This function was delivered through the service stream through PTF UQ88257 (APAR PQ81584) in the W502008 Service Level.

Note that Connection Manager RunAs Identity Enabled Thread is still supported and needed for some JDBC connectors. SyncToOSThread is not required when using a J2CA connector to access an EIS under the user ID of the current Java thread.

7.1.1 SWIPE application structure

This section describes the contents of the sample application including servlets, servlet URL mappings, and the protection settings for each. It focuses on what authentication mechanisms and URLs have been set up to try them.

SWIPE application contents

The sample application is packaged in an .ear file, which contains:

- IBMEBizEJB.jar: Contains the session EJB classes, EJBSample, and tools.
- IBMEBizWeb.war: Contains the EJBCaller servlet class, configured to use basic authentication. It also contains the servlet RunAsServlet, which is used to demonstrate the RunAs concept for servlets.
- IBMForms.war: Configured to use Forms-based authentication.
- IBMAuthClientSSL.war: Configured to use ClientCert type authentication.
- IBMnoWebRole.war: Configured with basic type authentication; but no role is specified for accessing the servlet.

Context root

Each .war file has a different context root and servlets, as shown in Table 7-1.

Table 7-1 Contents of a .war file

.war file	Context root	Servlets
IBMEBizWeb	IBMEBizWeb	EJBCaller, RunAsServlet, formSnoop
IBMForms	IBMForms	fbaEjbCaller, formSnoop
IBMAuthClientSSL	IBMClientSSL	sslEjbCaller, formSnoop
IBMnoWebRole	IBMnoWebRole	nwrEjbCaller, nwrSnoop

URL mappings

Within each .war file, multiple URL mappings are defined for each servlet. This allows the servlet to be run in various ways from an authentication perspective. The mappings that are configured are shown in Table 7-2 on page 125. The Authenticated column refers to whether that particular URL mapping is configured to require authentication.

Table 7-2 URL mappings in SWIPE

.war file	Servlet	URL	Authenticated
IBMEBizWeb		index.html	No
	EJBCaller	EJBCaller	No
		secure/EJBCaller	Yes
		z/OSHttpSecure/EJBCaller	No
		z/OSAndWasSecure/EJBCaller	Yes
	RunAsServlet	RunAsServlet	No
		secure/RunAsServlet	Yes
IBMEBizWeb	formSnoop	formSnoop	No
		secure/formSnoop	Yes
		ZosHttpSecure/formSnoop	No
		ZosAndWasSecure/formSnoop	Yes
IBMForms	fbaEjbCaller	fbaEjbCaller	No
		secure/fbaEjbCaller	Yes
		ZosHttpSecure/fbaEjbCaller	No
		ZosAndWasSecure/fbaEjbCaller	Yes
IBMForms	formSnoop	formSnoop	No
		secure/formSnoop	Yes
		ZosHttpSecure/formSnoop	No
		ZosAndWasSecure/formSnoop	Yes
IBMClientSSL	sslEjbCaller	sslEjbCaller	No
		secure/sslEjbCaller	Yes
		ZosHttpSecure/sslEjbCaller	No
		ZosSAndWasSecure/sslEjbCaller	Yes
IBMclientSSL	formSnoop	formSnoop	No
		secure/formSnoop	Yes
		ZosHttpSecure/formSnoop	No

.war file	Servlet	URL	Authenticated
		ZosAndWasSecure/formSnoop	Yes
IBMnoWebRole	nwrEjbCaller	nwrEjbCaller	No
		secure/nwrEjbCaller	Yes
		ZosHttpSecure/nwrEjbCaller	No
		ZosSAndWasSecure/nwrEjbCaller	Yes
IBMnoWebRole	formSnoop	formSnoop	No
		secure/formSnoop	Yes
		ZosHttpSecure/formSnoop	No
		ZosAndWasSecure/formSnoop	Yes

7.1.2 SWIPE application architecture and description

The SWIPE application consists of one .jar file containing the EJBs and four .war files.

EJBCaller

The main servlet is called EJBCaller, and resides in the IBMEBizWeb.war file. The deployment descriptor in this .jar file is configured to use basic type authentication.

Because we also wanted to demonstrate Forms-based and client certificate authentication, we set up two additional .jar files, as only one form of authentication can be specified per .war file.

This means that each .war file has a main servlet that is invoked. However, we did not want to have to repeat the code in each of these additional servlets. What we did was to code these servlets to invoke the EJBCaller servlet in the IBMEBizWeb.war file. This means that one servlet program can be used to demonstrate various authentication models. Changes to this one program can then be used regardless of which initial servlet is used. For example, the fbaEjbCaller servlet calls the doPostWork method of the EJBCaller servlet.

EJBSample.jsp

Output produced by the servlet is passed to a JSP called EJBSample.jsp. There must be a separate copy of this JSP in each .war file, as there does not appear to be a way to easily reference a JSP residing in another .war file.

EJBSample

In the IBMEBiz.jar file is the session EJB called EJBSample. It contains the RunAs methods used to demonstrate declarative security, which in turn do a isCallerInRole call to demonstrate programmatic security.

There is another EJB in this .jar file called Tools. It is just used to help obtain the principal user ID that a method is currently running under and to store some variables that are needed.

index.html

This is an HTML page that contains a number of the URLs shown in Table 7-2 on page 125.

The major components are shown in Figure 7-1.

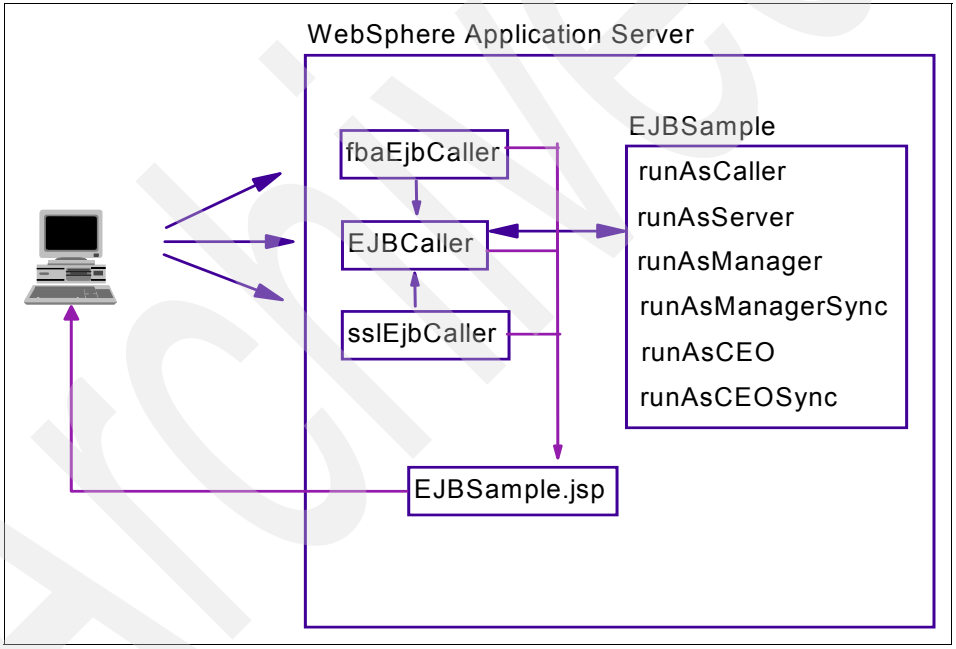


Figure 7-1 Components of the SWIPE application

EJBCaller servlet

This servlet is the core servlet containing the code to invoke the EJB.

If the user has entered a file name, the servlet invokes a method to access the file from the servlet, calls the method getEjbToReadFile() on the instantiated

EJBSample Session Bean to access the file, and also passes it as a parameter so that the various RunAs methods also access the file.

The servlet performs a programmatic check the role name entered on the form. It calls a method in EJBSample to have it perform a programmatic check, and also passes it as a parameter so that the various RunAs methods can also perform a programmatic check.

If the user has entered a remote JNDI location, the servlet passes these parameters to the EJB. Each of the RunAs methods will access this remote EJB and report on the result.

EJBSampleBean

The main methods are:

- RunAsCaller()
- RunAsCallerSync
- RunAsServer()
- RunAsRoleCEO()
- RunAsRoleManager()
- RunAsRoleCEOSync()
- RunAsRoleManagerSync()

The end user specifies the combination of the above methods to be invoked, with six being invoked in sequence. The EJBCaller servlet first stores the method names selected in a Vector object and passes that to the EJBSample object, which in turn calls a method in the Tools object to save the method names there.

The EJBCaller servlet then invokes the first method specified. The first method runs and produces a report. It then creates a new instance of EJBSample and invokes the next method specified on this new EJBSample object.

If we had just coded to invoke the next RunAs method specified in the sequence on the existing EJBSample object, no change in the principal user ID occurs.

The following methods prepare formatted outputs of the actions asked of them:

- WhereAmI() gets the date and returns the value of an environment variable called IBMEBizJVM, which is set in the JVM properties.
- getEjbSecurityInfo() gets the Principal ID of the caller.
- getEjbToReadFile() reads a file and returns the results (if file access failed because of no permissions, the exception is returned).
- chkCallerInRole() checks to see if the caller has the specified role.

ToolsBean

This is a helper EJB used to store various parameters.

EJBSample JSP

Returns the results from running the EJBCaller servlet. Information displayed can include:

- HTTP Request Headers
- Cookies
- RunAs methods results
- Principal user ID of the servlet and EJB

7.2 SWIPE authentication features

The URLs shown below allow you to run the servlet without requiring authentication or using any of the three available security mechanisms.

General

Where you run the servlet with a URL of this structure:

```
http://yourhostname:port/IBMEBizWeb/EJBCaller
```

WebSphere will detect that the deployment descriptor specifies that no authentication is required in the Web container.

IBM HTTP Server authentication only

Where you run the servlet with a URL of this structure:

```
http://yourhostname:port/IBMEBizWeb/ZosHttpSecure/EJBCaller
```

WebSphere will detect that the deployment descriptor specifies that no authentication is required in the Web container. However, the idea here is to configure the IBM HTTP Server to perform authentication and then pass the request to the WebSphere HTTP plug-in.

Web container and IBM HTTP Server authentication

Where you run the servlet with a URL of this structure:

```
http://yourhostname:port/IBMEBizWeb/ZosAndWasSecure/EJBCaller
```

WebSphere will detect that the deployment descriptor specifies that basic type authentication is required in the Web container. Additionally, you would configure the IBM HTTP Server to perform authentication as well.

The authentication is not described in detail here because it is described in other chapters and it is assumed the user is authenticated and is mapped to one of the RACF-defined roles.

7.3 Authorization features

What we wanted to be able to do with this sample application is to demonstrate the J2EE security concepts explained in Chapter 3, “J2EE 1.3 and WebSphere Application Server V5 security concepts” on page 41 topics such as:

- Declarative security
- Programmatic security
- The “run as” concept
- JAAS

The SWIPE application demonstrates these topics and shows how they work from the J2EE security viewpoint, and then shows what the effect is in WebSphere Application Server for z/OS and OS/390 when the sample is executed.

We briefly describe these security concepts that the SWIPE application demonstrates. These are covered in greater detail in other chapters of this book.

Declarative security

Declarative security has to do with where the container that is running the EJBs is responsible for handling security. The deployment descriptor specifies what roles are defined and which role has access to which methods in an EJB. The container’s responsibility is to check if a caller is authorized to access a method in an EJB. It does this by determining which roles, if any, have access to the method, and whether the caller is in that role.

Programmatic security

In programmatic security the program itself makes security decisions. This can be done in any way the programmer wants. For example, the program might reference a database table to see if the user has access to some function. From a J2EE point of view, Java programs can use the isCallerInRole Java API to check whether the user ID the process is running under has access to a specified EJB role.

7.3.1 Web container authentication and authorization

To demonstrate authentication in the Web container, there are three .war files in SWIPE. Table 7-3 shows the authentication mechanism used by the .war files.

Table 7-3 The .war files and corresponding authentication

.war file	Authentication type
IBMEBizWeb	Basic
IBMForms	Forms based
IBMAuthClientSSL	Client certificate

To demonstrate authorization, an EJBRole and security constraint is set up in each .war file. This information is stored in the web.xml file in each .war file. Table 7-4 shows the EJBRole associated with the SWIPE .war files. When the servlet is run through a URL that invokes authentication, the authenticated user will only be able to run the servlet if it has access to the EJBRole shown.

Table 7-4 The .war files and corresponding EJBRole

.war file	EJBRole
IBMEBizWeb	employee
IBMForms	WasFormUser
IBMAuthClientSSL	WasSSLUser

7.3.2 EJB container authorization: EJBRoles

To demonstrate authorization in the EJB container, a number of EJBRoles and differently named methods are set up in SWIPE. The aim is to show that an authenticated user in a servlet or an EJB can only access methods of the EJBSample EJB if the user has access to the EJBRole that protects the method. Table 7-5 shows the methods set up in the SWIPE EJBSample EJB and the EJBRole used to protect each one.

Table 7-5 EJBRoles used to protect methods in the EJBSample EJB in SWIPE

Method name	EJBRole
RunAsCaller()	Worker
RunAsCallerSync	Worker
RunAsServer()	Worker
RunAsRoleManager()	Manager

Method name	EJBRole
RunAsRoleManagerSync()	Manager
RunAsRoleCEO()	CEO
RunAsRoleCEOSync()	CEO

Defining EJBRoles for the EJB

The EJBRoles used with the EJBSample EJB are defined by adding them to the ejb-jar.xml file that is in the EJB part of the project in WebSphere Studio Application Developer. Figure 7-2 shows the WebSphere Studio Application Developer display where we did this. EJBRoles are added by selecting the **Assembly Descriptor** tab and then simply clicking **Add** under the Security Roles heading and defining them.

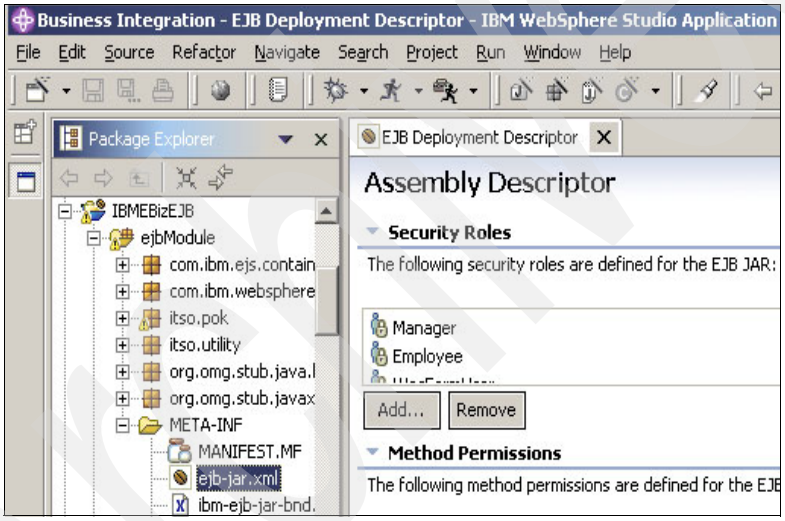


Figure 7-2 WebSphere Studio Application Developer V5.1: Defining EJBRoles

7.3.3 EJB container: Declarative security

Table 7-5 on page 131 showed the EJB roles defined. In our sample application, we set up several methods in the EJB and then established rules as to which EJB roles had access to which methods. Note that we did not specifically protect every method of the EJB, because that is not required for this application.

On the same tab of WebSphere Studio Application Developer where the EJBRoles where defined is a heading called Method Permissions. Under this heading are shown any method permissions that have already been defined and

where you can add, edit, and remove methods of the EJB that are to be protected by a nominated EJBRole.

Complete the following steps:

1. To add a method permission, click **Add**, which opens the window shown in Figure 7-3.

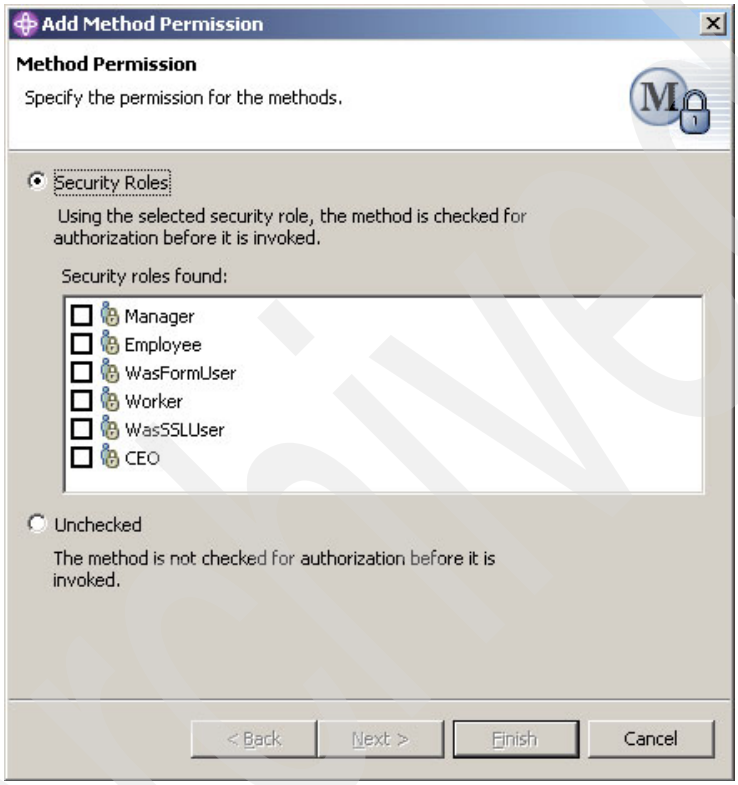


Figure 7-3 WebSphere Studio Application Developer V5.1: Selecting EJBRole

2. Select the EJBRole to use, and then click **Next**, which opens the window shown in Figure 7-4.

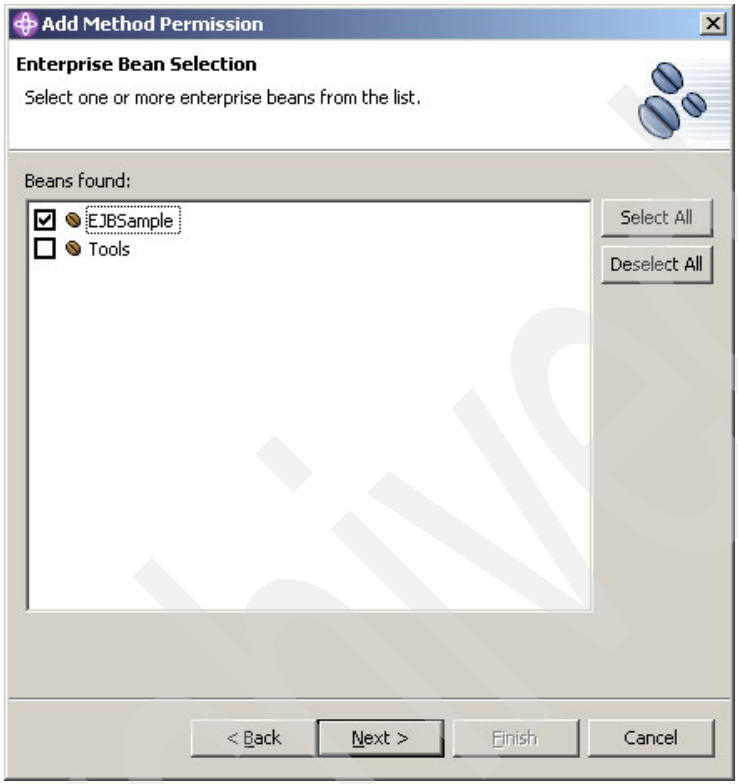


Figure 7-4 WebSphere Studio Application Developer V5.1: Selecting EJBs

3. Select the EJBs to action, and then click **Next**, which opens the window shown in Figure 7-5.

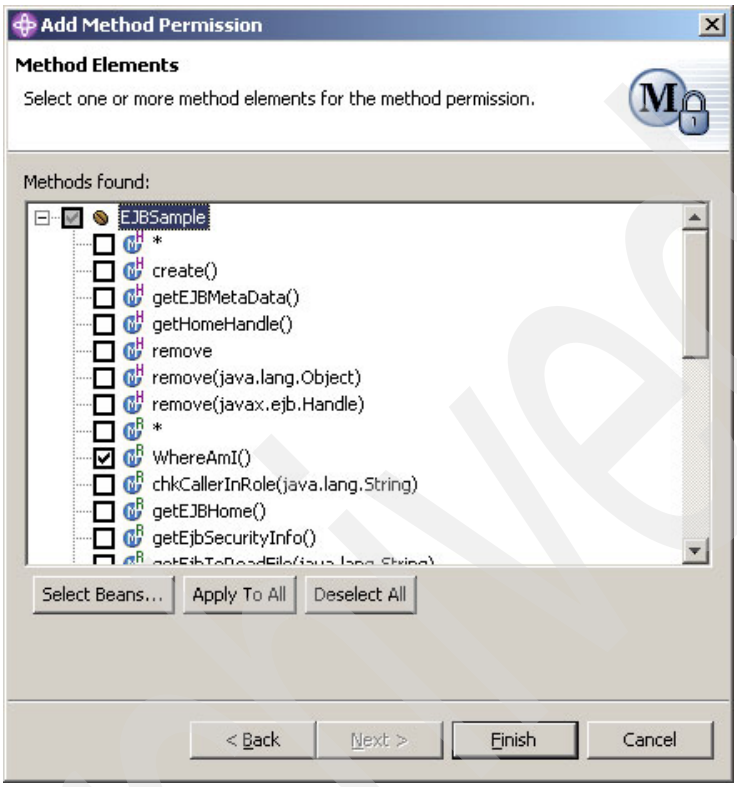


Figure 7-5 WebSphere Studio Application Developer V5.1: Selecting the methods

4. In this window, you then select the methods you want to protect with the EJBRole that you selected in Figure 7-3 on page 133. When the methods have been selected, click **Finish**.

Note that any number of roles can be granted permission to a method.

Table 7-6 on page 136 shows the methods the example uses to demonstrate declarative security, and what roles have access to them.

Table 7-6 SWIPE: Roles and their access to methods

Method	Worker	Manager	CEO
RunAsCaller	X		
RunAsCallerSync	X		
RunAsServer	X		
RunAsManager		X	
RunAsManagerSync		X	
RunAsCEO			X
RunAsCEOSync			X

Note, for example, that the Manager and CEO roles do not have permission to the RunAsCaller method. This was done deliberately. The intention was that users in the Manager and CEO role will be authorized to run the RunAsCaller and RunAsServer methods.

7.3.4 EJB container: Programmatic security

To demonstrate programmatic security, the sample application allows you to select from a list of role names to see if the method caller has access. The caller has access if a result of `true` is returned when the isCallerInRole method is invoked. The sample application does no more than this, but a real application having determined that the caller had access could then proceed accordingly.

The list of roles you can select from includes the roles defined for use in demonstrating declarative security, plus these three:

- GrantPayRise
- PromoteWorkers
- CompanyCar

These roles are referred to as role references or logical references. The normal approach is to *link* which EJB security role has access to a logical role reference, the information being stored in the deployment descriptor.

This approach was used for the CompanyCar logical security role.

Using RACF for logical security role references

However, we did not use this approach for the other two logical role references.

For example, as mentioned above, using WebSphere Studio Application Developer V5.1 or the AAT, only one EJB security role can be given access to a logical role reference. In RACF, however, we can give any combination of groups or users, or both, access to the logical EJB role. Also, because the security information is no longer held in the deployment descriptor, it is much easier to change the access.

Changing who has access to a logical role reference defined in the deployment descriptor would require modification through WebSphere Studio Application Developer V5.1, the AAT, or by some other manual editing process, and then redeployment of the application.

If you want to define the logical security roles in the EJB descriptor, you can use WebSphere Studio Application Developer V5.1 or the WebSphere AAT to do this.

Defining logical security role references through WebSphere Studio Application Developer V5.1

In WebSphere Studio Application Developer V5.1 the logical security roles are defined by adding them to the ejb-jar.xml file that is in the EJB part of the project in WebSphere Studio Application Developer. Open the ejb-jar.xml entry and select the **References** tab. Figure 7-6 shows the WebSphere Studio Application Developer display when this is done.

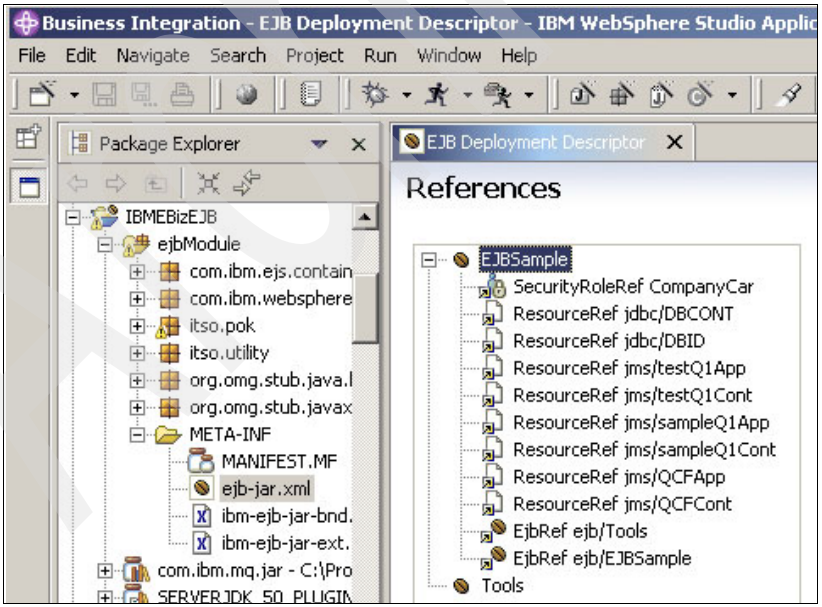


Figure 7-6 WebSphere Studio V5.1: Logical security role references

To add a logical security EJBRole reference, click **Add**, which will open the window shown in Figure 7-7.

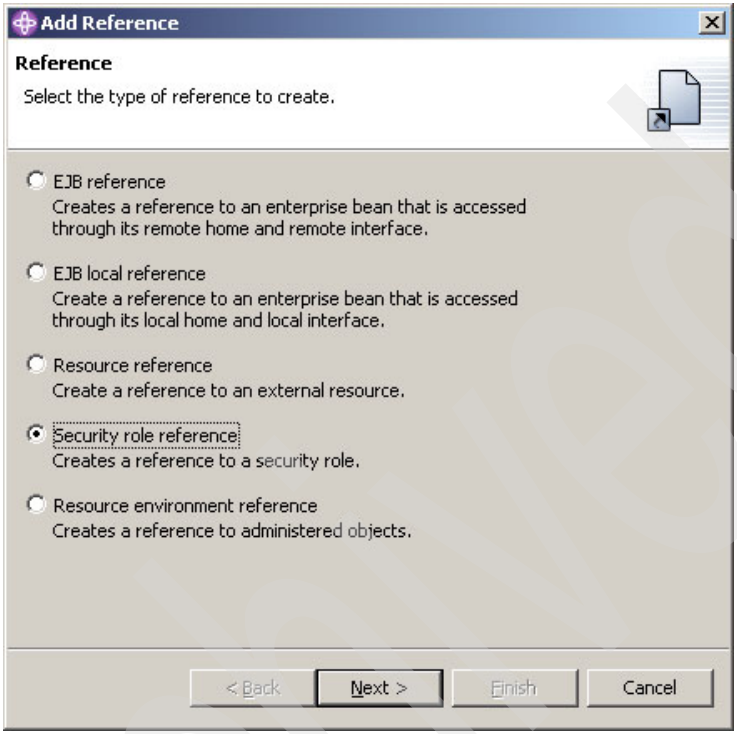


Figure 7-7 WebSphere Studio Application Developer V5.1: Security role reference

Select the **Security role reference** option and click **Next**, which will open the window shown in Figure 7-8 on page 139.

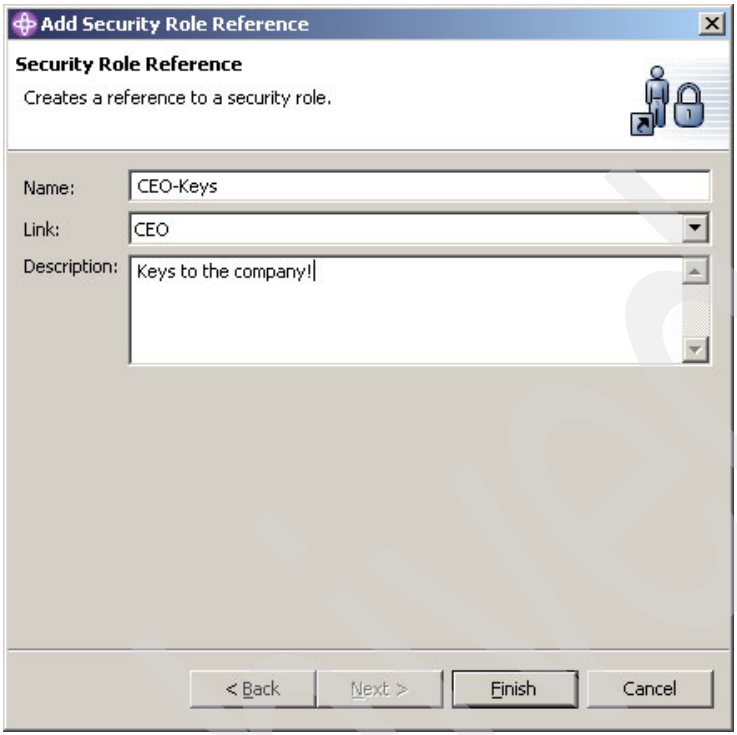


Figure 7-8 WebSphere Studio Application Developer V5.1: Security Role Reference

Then fill in the fields to define the logical Security Role Reference you require and what EJBRole has access to it and click the **Finish** button.

Defining logical security role references through AAT

The WebSphere AAT can also be used to define which servlet or EJB security role had access to a logical role reference. In the AAT, we selected the servlet or the bean in the tree. We then clicked the **Security Role Reference** submenu. Then we clicked the pencil icon and then **Add**. This brought up the display shown in Figure 7-9 on page 140.

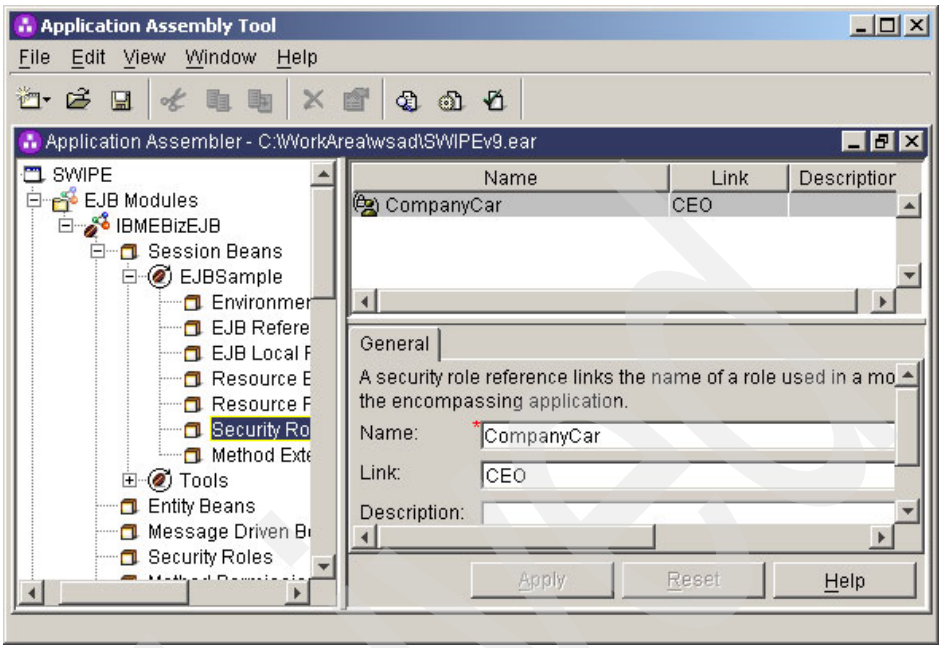


Figure 7-9 Specifying which EJB security role has access to a logical role

Here, we enter the name of the role as it is used in the application and the linked role that will be checked by WebSphere.

Note: Only one logical EJB security role can be assigned to a logical role name.

Logical security role reference XML

The XML, created through either WebSphere Studio Application Developer V5.1 or the AAT in the ejb-jar.xml file, is shown in Example 7-1.

Example 7-1 Logical security role reference XML

```
<security-role-ref>
   <role-name>CompanyCar</role-name>
       <role-link>CEO</role-link>
</security-role-ref>
```

7.3.5 EJB: RunAs concept

This concept relates to which user ID the process runs as inside the container, after the container has verified that the caller has access to the method in the servlet or the EJB. This can be one of the following IDs:

- The user ID of the caller
- The user ID of the container (only for EJBs)
- A specific user ID associated with the role

The names of the methods we are using in the sample application, listed in Table 7-6 on page 136, were deliberately chosen to indicate the role we wanted the method to run as in the EJB container.

We defined the role ID we wanted a method to run as using WebSphere Studio Application Developer.

To do this in WebSphere Studio Application Developer, double-click the file ibm-ejb-jar-ext.xmi. This then displays a pane in WebSphere Studio Application Developer V4 where you set the RunAs mode. Figure 7-10 on page 142 shows this display.

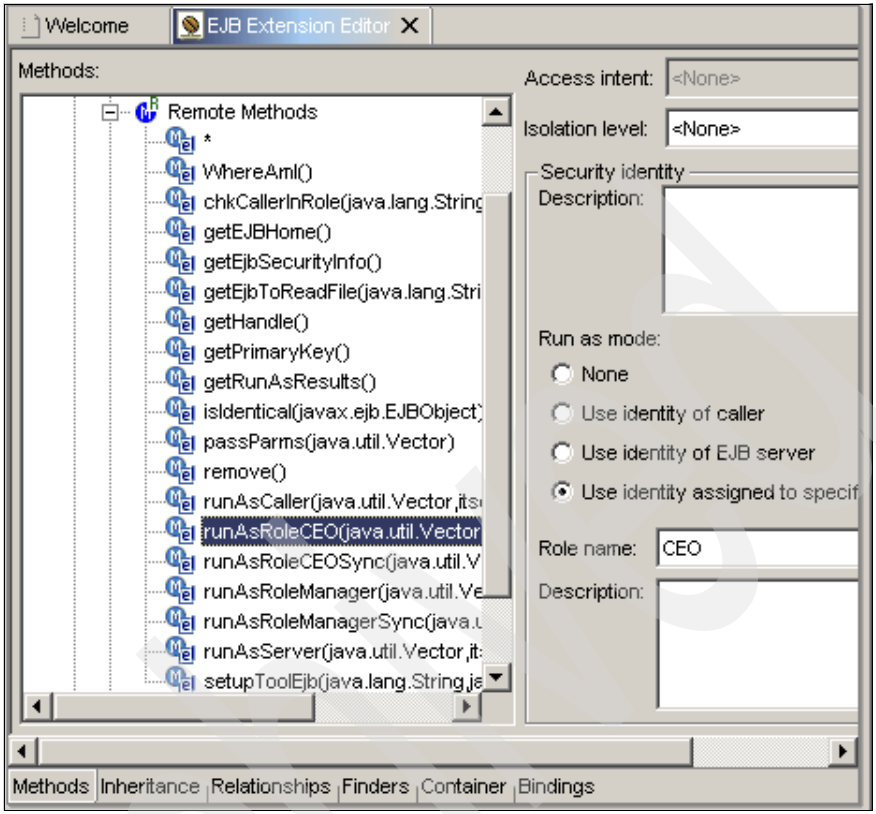


Figure 7-10 Defining which runAs mode a method runs as

To set the RunAs mode, we selected the method name and then selected the RunAs mode we required.

Table 7-7 shows the RunAs modes we set for the methods.

Table 7-7 Setting RunAs modes

Method	RunAs mode
RunAsCaller	Caller ID
RunAsServer	Server ID
RunAsManager	Identity assigned to role of Manager
RunAsManagerSync	Identity assigned to role of Manager
RunAsCEO	Identity assigned to role of CEO
RunAsCEOSync	Identity assigned to role of CEO

7.3.6 Servlet run-as example

16.4, “Web container identity delegation and propagation” on page 556 describes the capability that now exists to associate a run-as role with a servlet. To demonstrate this with SWIPE, a servlet called RunAsServlet was created in the IBMEBizWeb .war file. Additionally, a new security role called Cleaner was created.

WebSphere Studio Application Developer V5.1 now provides the means to set up a run-as role for a servlet.

The RunAsServlet basically only contains code for the doGet and doPost methods. It then calls methods of the main EJBCaller servlet.

Creation of the Cleaner security role and setting the run-as security setting for the RunAsServlet is done by making changes to the deployment descriptor stored in web.xml. Using WebSphere Studio Application Developer V5.1, locate the web.xml file for the .war file and double-click it to open it.

Creating the Cleaner security role

Click the **Security** tab and then click **Add**. A new entry is displayed, and you then type in the name of the new security role you want to add.

Setting run-as on the RunAsServlet

Click the **Servlets** tab and select **RunAsServlet**; the display will be as shown in Figure 7-11.

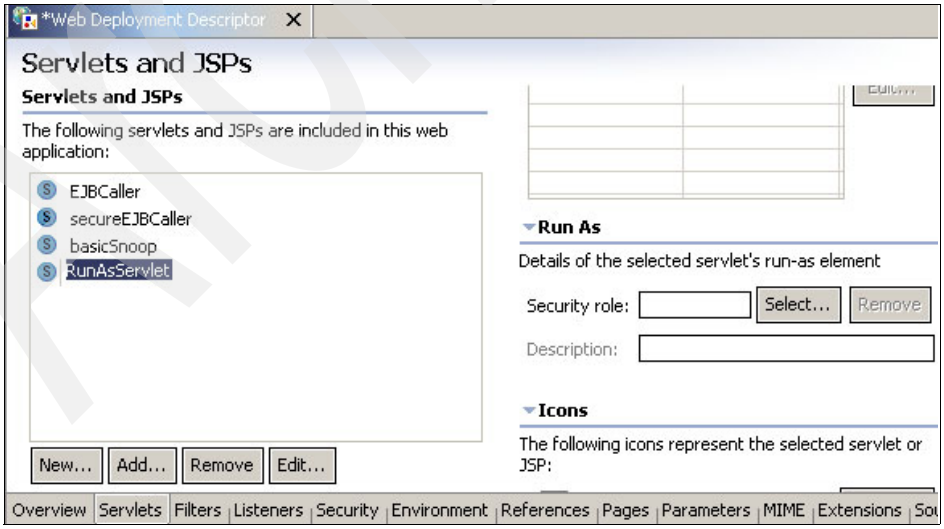


Figure 7-11 WebSphere Studio Application Developer V5.1: Run As security for a servlet

Click the **Select** and WebSphere Studio Application Developer will display the pane shown in Figure 7-12.

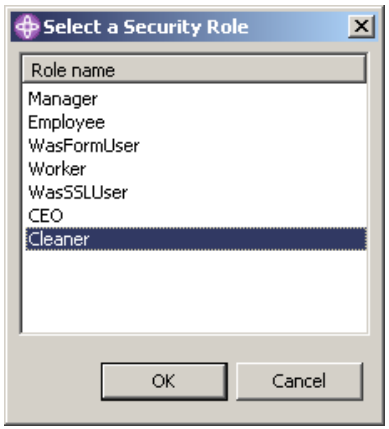


Figure 7-12 WebSphere Studio Application Developer V5.1: Security role to servlet

We then selected the **Cleaner** security role to be assigned as the Run As setting for the RunAsServlet servlet and clicked **OK**.

Update the Application Deployment Descriptor

The above process updated the security settings in the deployment descriptor of the .war file. However, it is necessary to update the Application Deployment Descriptor of the .ear file, otherwise the new EJBRole will not show up during the deployment process into WebSphere.

To do this, select the **J2EE Hierarchy** tab, right-click the project **SWIPE**, and then select **Open With** → **Deployment Descriptor Editor**. You will see the window shown in Figure 7-13 on page 145.

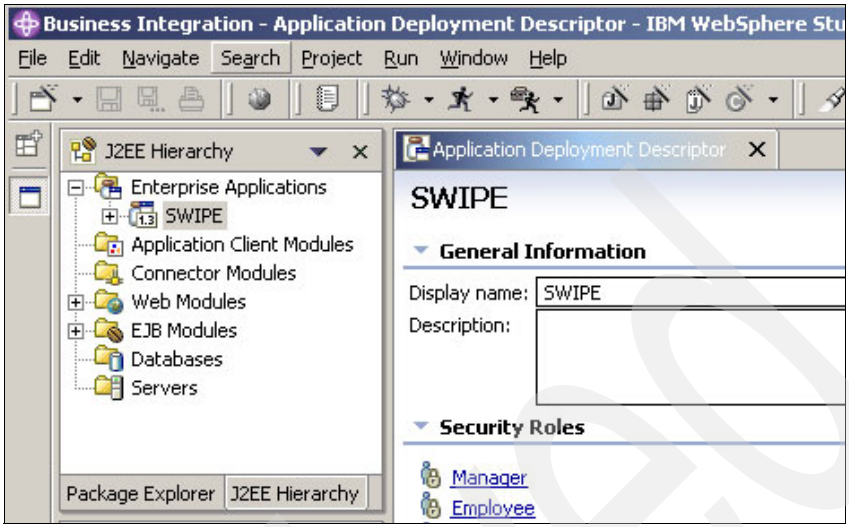


Figure 7-13 WebSphere Studio Application Developer V5.1: Deployment Descriptor

Select the **Security** tab and the display will show the list of already known EJBRoles. The Cleaner EJBRole added in the Deployment Descriptor of the .war file does not show in the list. To add it, click **Gather** and the Cleaner EJBRole is added to the list, as shown in Figure 7-14.

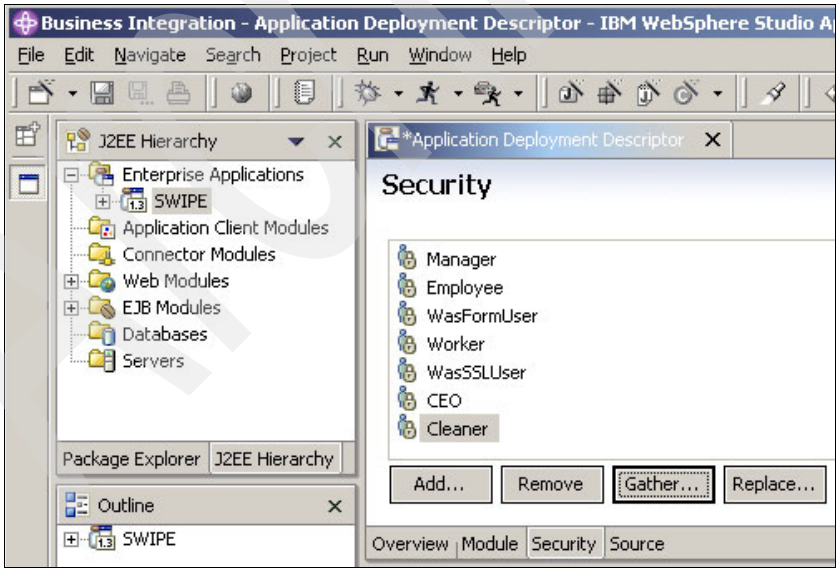


Figure 7-14 WebSphere Studio Application Developer V5.1: "Gathering" EJBRole

Then, just close the Application Deployment Descriptor to save it.

Setting RunAs using the AAT

Figure 7-15 shows how the AAT can be used to set the security role to use for a servlet.

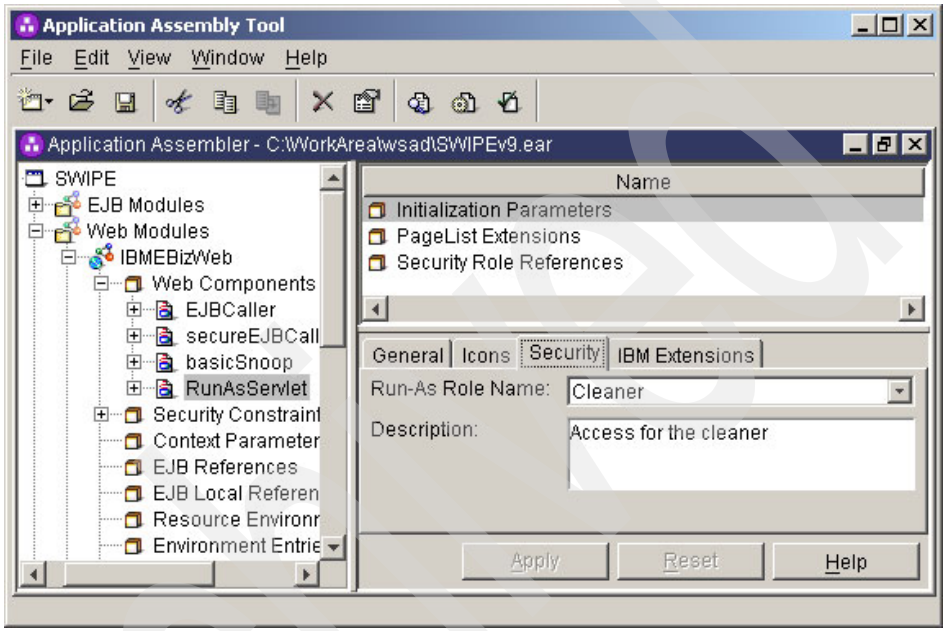


Figure 7-15 Setting RunAs for a servlet through the AAT

7.3.7 The "Sync to OS Thread" concept

In the redbook *z/OS WebSphere and J2EE Security Handbook*, SG24-6846, which dealt with security in WebSphere Application Server for OS/390 V4, this corresponding section discussed the SyncToOSThread concept. This feature allowed access to a resource outside of WebSphere, such as a file, to be achieved under the user ID under which the servlet or EJB was running, as opposed to the user ID under which the WebSphere started task was running. SyncToOSThread is not required when using a J2CA connector to access an EIS under the user ID of the current Java thread.

To show the effect of this capability in WebSphere Application Server for z/OS and OS/390 Version 4 to synchronize the Java identity to a z/OS RACF identity, SWIPE used these two methods:

- RunAsManagerSync

- RunAsCEOSync

This function is being deprecated. IBM did not deliver SyncToOSThread on method descriptors in the initial V5.0 availability. This function was delivered through the service stream through PTF UQ88257 (APAR PQ81584) in the W502008 Service Level.

These methods function in the same way as their corresponding non-sync methods, namely RunAsManager and RunAsCEO.

7.4 The downloadable SWIPE package

The ZIP file containing the SWIPE application that can be downloaded from the IBM Redbooks site (described in Appendix B, “Additional material” on page 717) contains the contents shown in Figure 7-16.

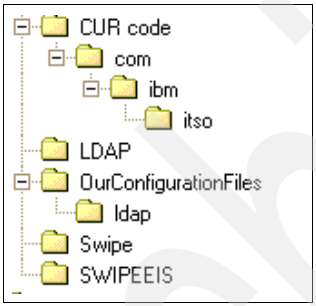


Figure 7-16 Contents of downloadable ZIP file

7.5 Deploying SWIPE

This section describes the process we used to deploy the SWIPE application on our test system.

Deployment of SWIPE into a WebSphere Application Server for z/OS and OS/390 system is the same as for WebSphere Application Server V5 on distributed systems.

To deploy the SWIPE application, use the WebSphere Application Server for z/OS administrative console. With a browser, enter the URL to point to your server IP address and IP port followed by the /admin URI. The logon page should appear as shown in Figure 7-17 on page 148.

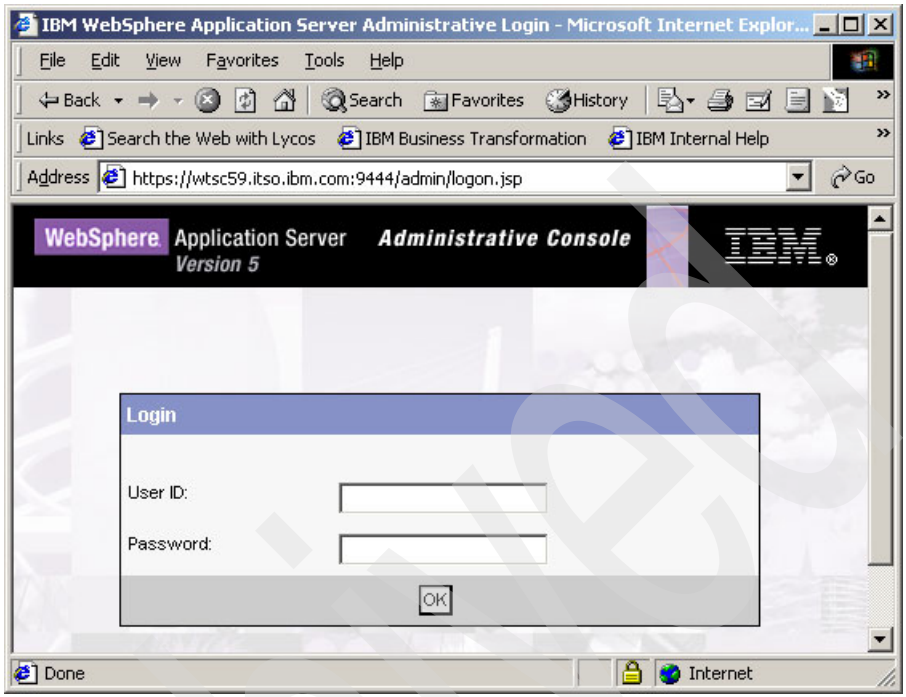


Figure 7-17 Administrative Console login page

After you have authenticated with a valid administrator, you are then presented with a new display showing the WebSphere administrative console menus, as shown in Figure 7-18 on page 149.

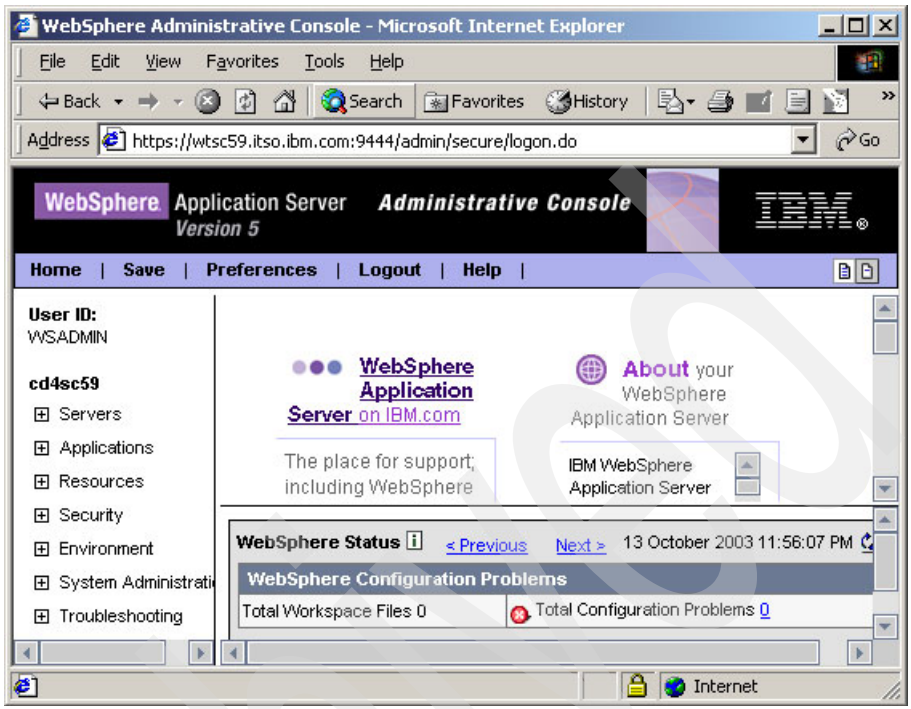


Figure 7-18 Administrative Console main menu

To install SWIPE, you have to expand the Applications choice on the left frame and then click the **Install New Application** option. The display will then change to allow you to enter the SWIPE .ear location as shown in Figure 7-19.

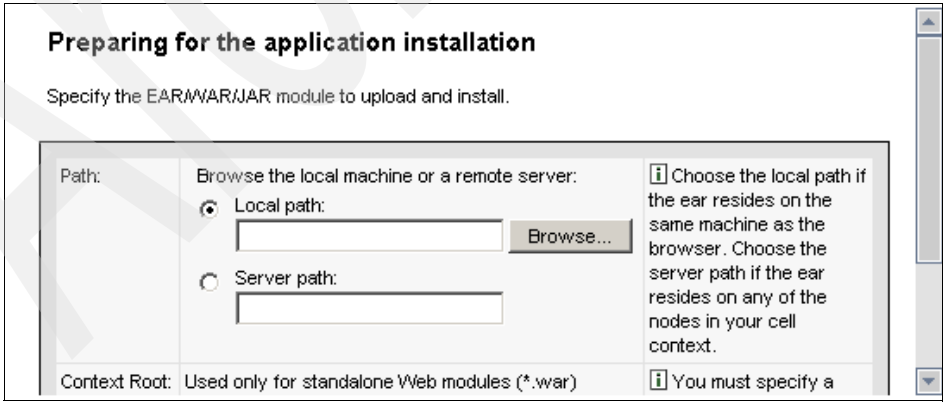


Figure 7-19 Administrative application .ear location

After you have entered the SWIPE.ear path and clicked **Next**, you are prompted to enter the default bindings and mappings as shown in Figure 7-20. You can leave all options as default except if you have a virtual host different than `default_host`.

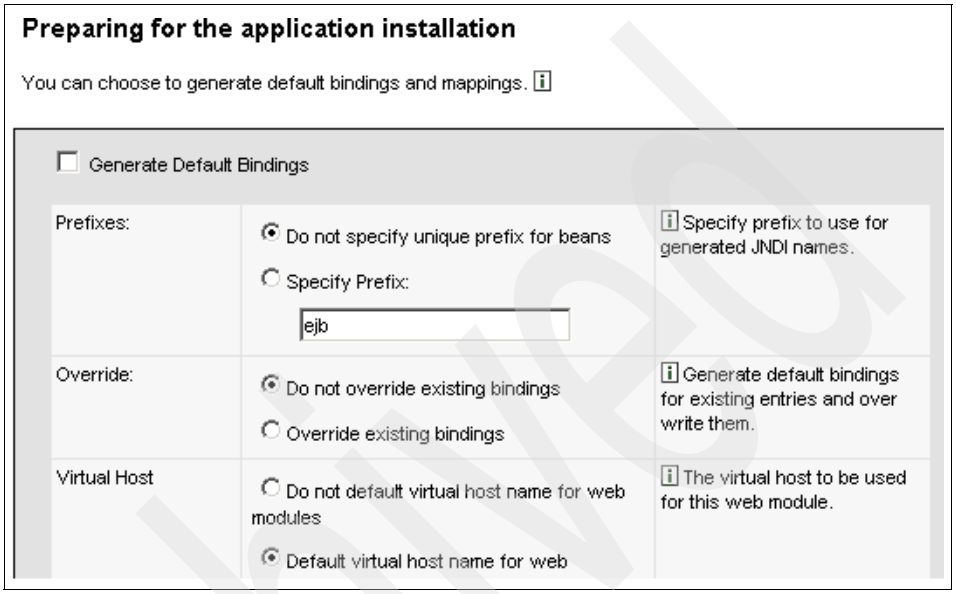


Figure 7-20 Administrative application default bindings and mappings

After you have clicked **Next**, you are prompted to enter the application's options, as shown in Figure 7-21 on page 151.

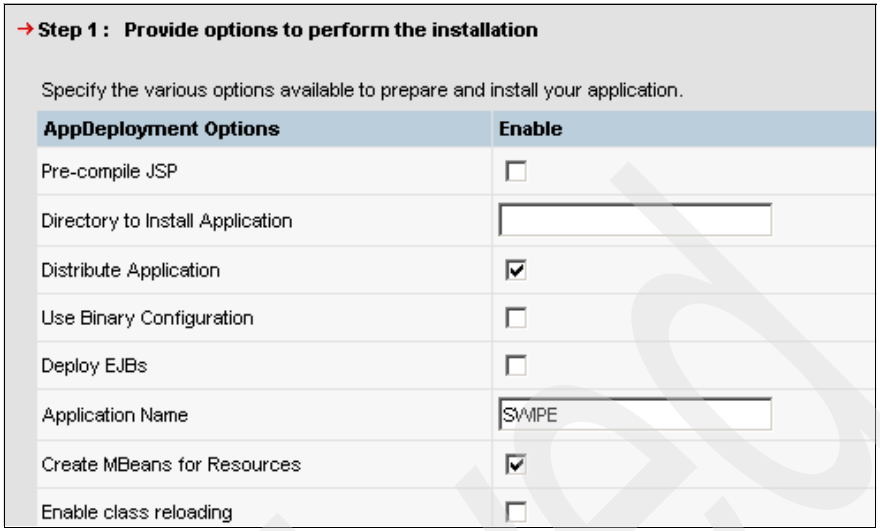


Figure 7-21 Administrative console application options

You can leave all the defaults and click **Next**.

The next screen will show the JNDI names of application EJBs, as illustrated in Figure 7-22.

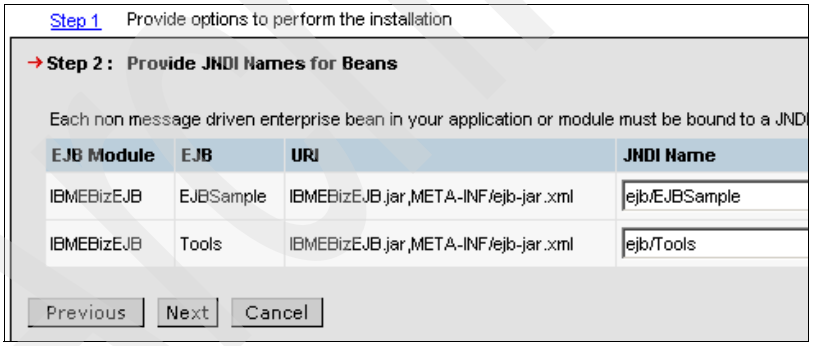


Figure 7-22 Administrative console EJB JNDI names

You can leave all the defaults and click **Next**.

The next panel (Figure 7-23 on page 152) allows you to change the references between EJB references in the application and the target JNDI name.

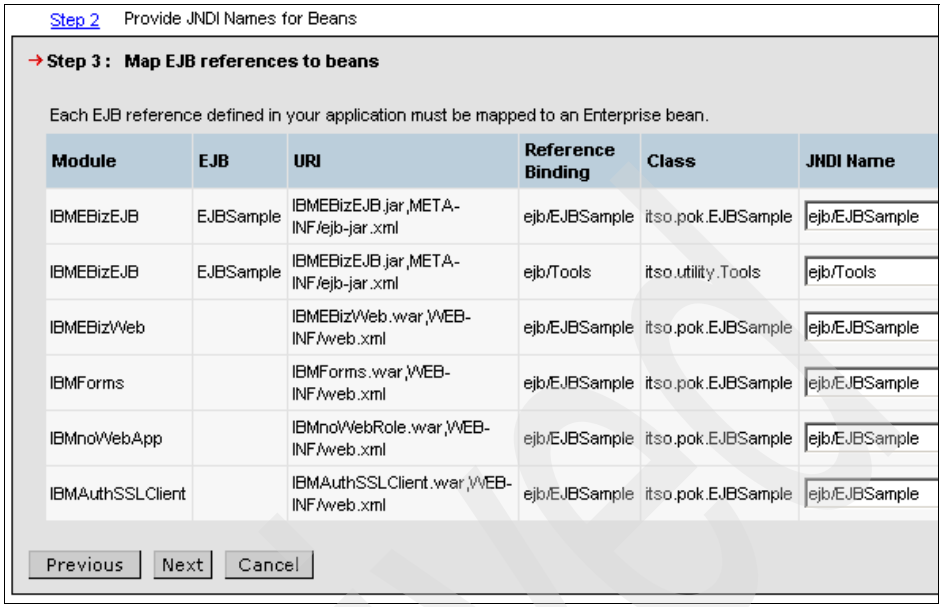


Figure 7-23 Administrative console EJB references

The next panel (Figure 7-24) allows you to change virtual host mapping on a .war level.

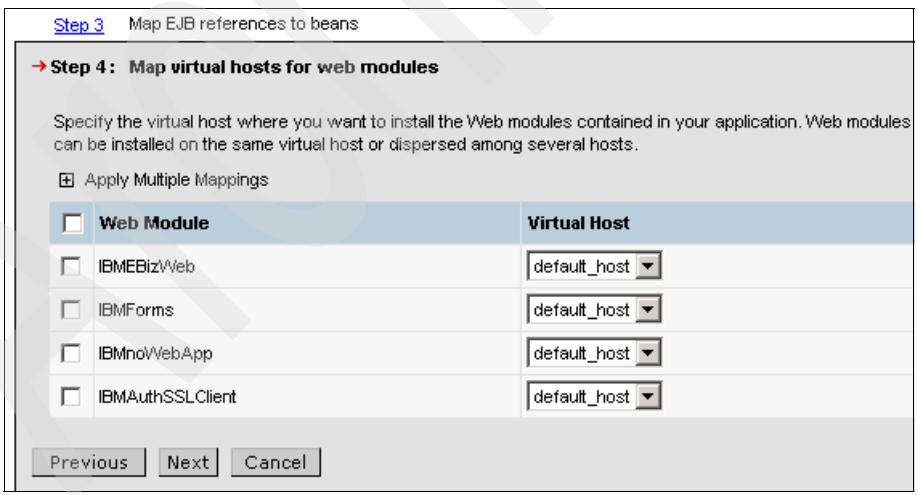


Figure 7-24 Administrative console virtual host mapping

You can leave all of the defaults and click **Next**.

The next panel (Figure 7-25) asks you to select in which application server or cluster you want the application to run.

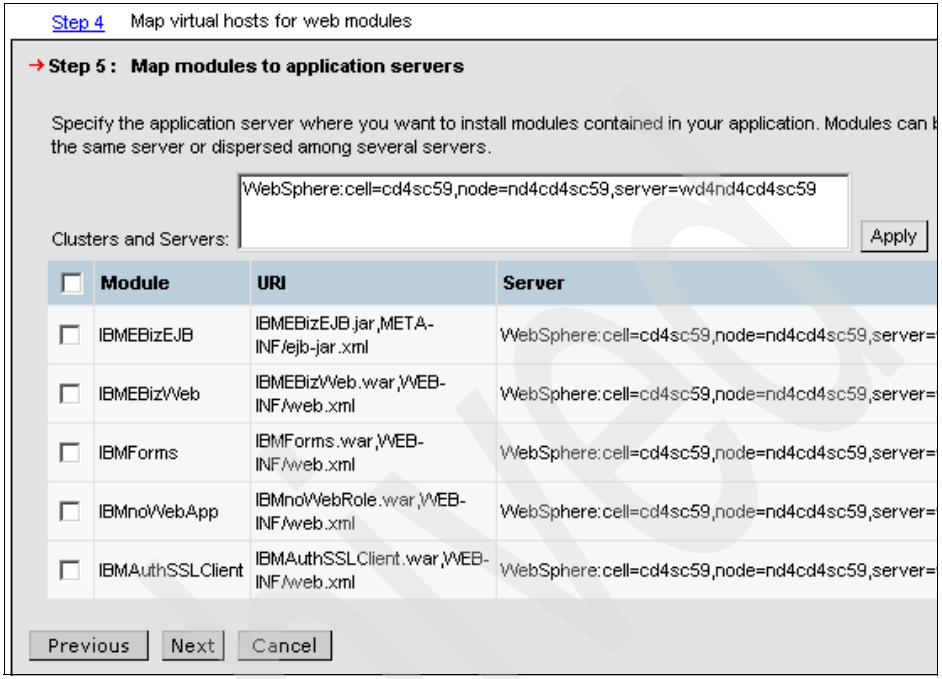


Figure 7-25 Administrative console application server mapping

You can leave all of the defaults and click **Next**.

The next panel (Figure 7-26 on page 154) is used only if you do not want to use either the OS registry (`com.ibm.security.SAF.authorization set to false`) or bindings set under WebSphere Studio Application Developer to determine which users are associated with roles.

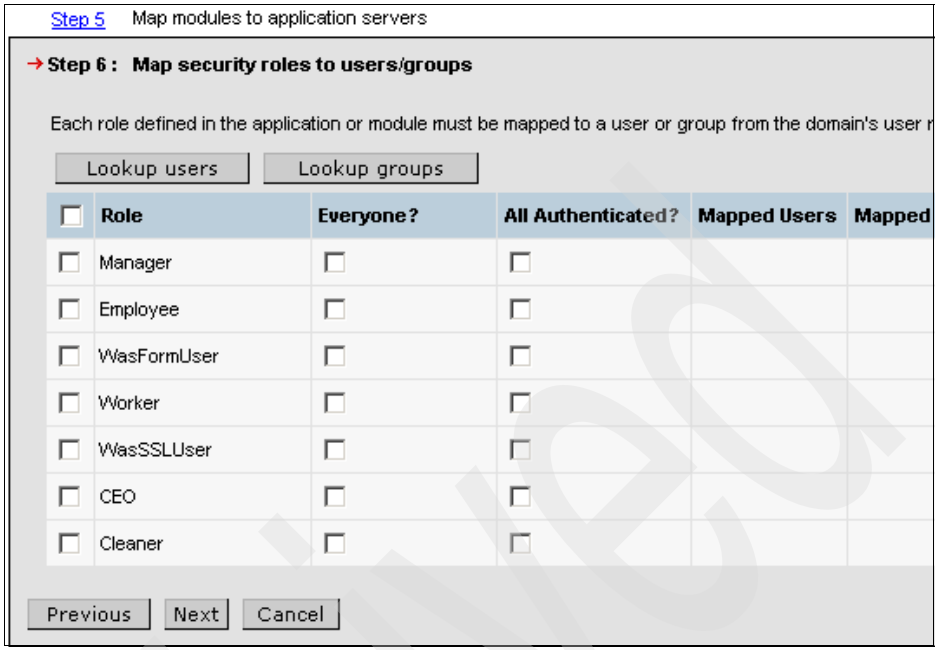


Figure 7-26 Administrative console security role mapping

If the server you are deploying into is configured to use OS and you have not added the following setting, click **Next**:

```
com.ibm.security.SAF.authorization set to false
```

If you do have the above setting, then if you have set up associations in the deployment descriptor, you can just click **Next**. If you did not set up settings in the descriptor or if you have configured the server to use LDAP or a CUR, you should assign users/groups to the EJBRoles.

The next panel (Figure 7-27 on page 155) allows you to associate a user ID with each role declared in a RunAs tag.

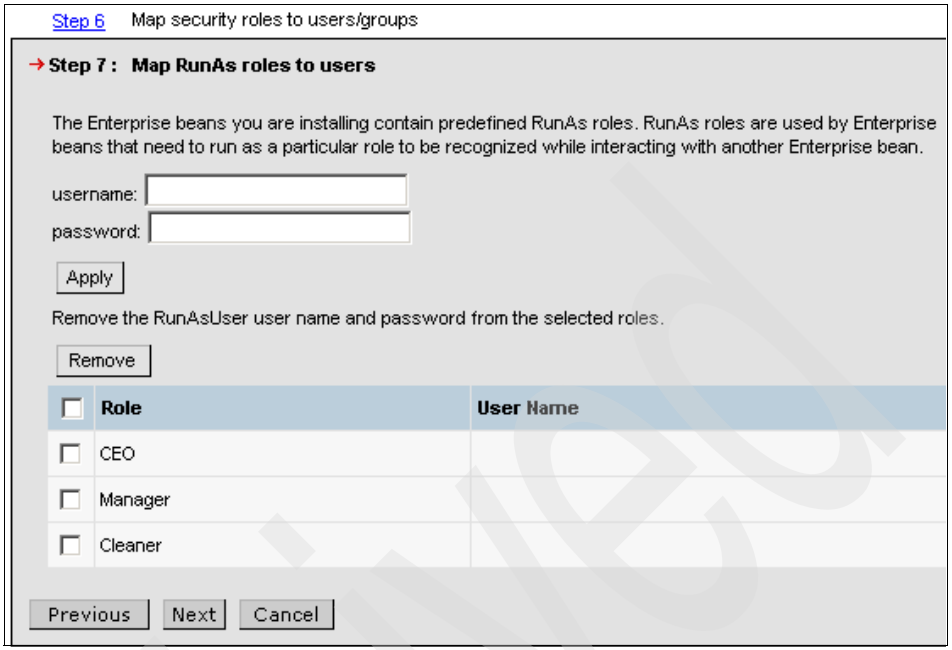


Figure 7-27 Administrative console RunAs mapping

If the server you are deploying into is configured to use OS and you have not added the following setting, click **Next**:

```
com.ibm.security.SAF.authorization set to false
```

If you do have this setting, if you have set up associations in the deployment descriptor, you can just click **Next**. If you did not set up settings in the descriptor, or if you have configured the server to use LDAP or a CUR, you should assign users/groups to the EJBRoles.

The next panel (Figure 7-28 on page 156) proposes to change all server RunAs methods with role RunAs.

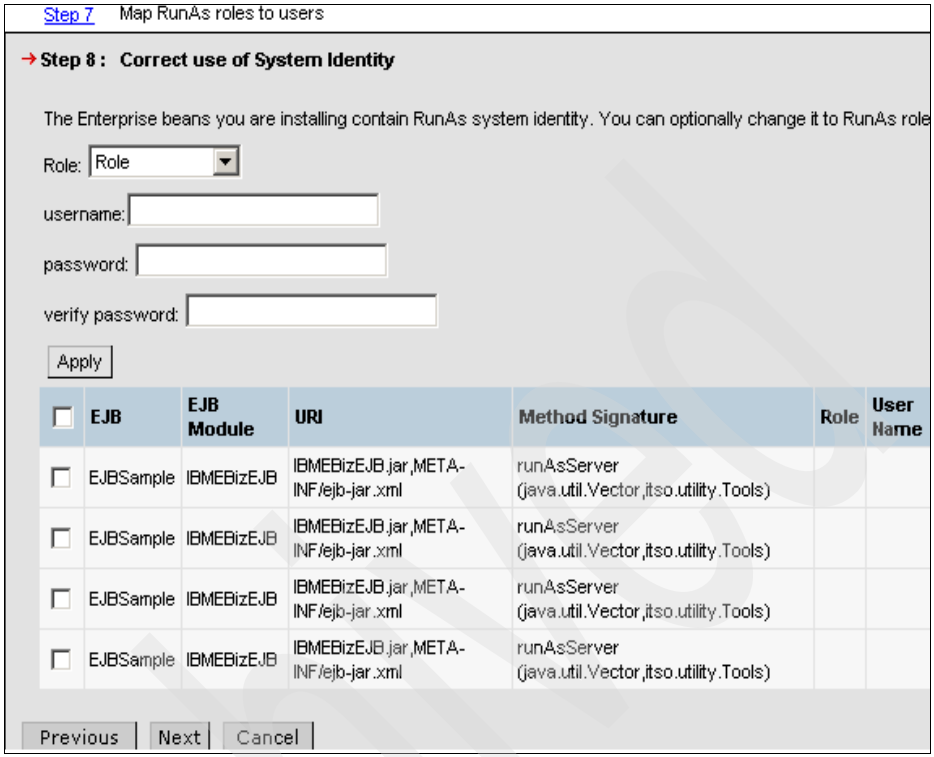


Figure 7-28 Administrative console system identity

You can leave all of the defaults and click **Next**.

The next panel (Figure 7-29 on page 157) proposes to assign security roles to unprotected EJB 2.0 methods.

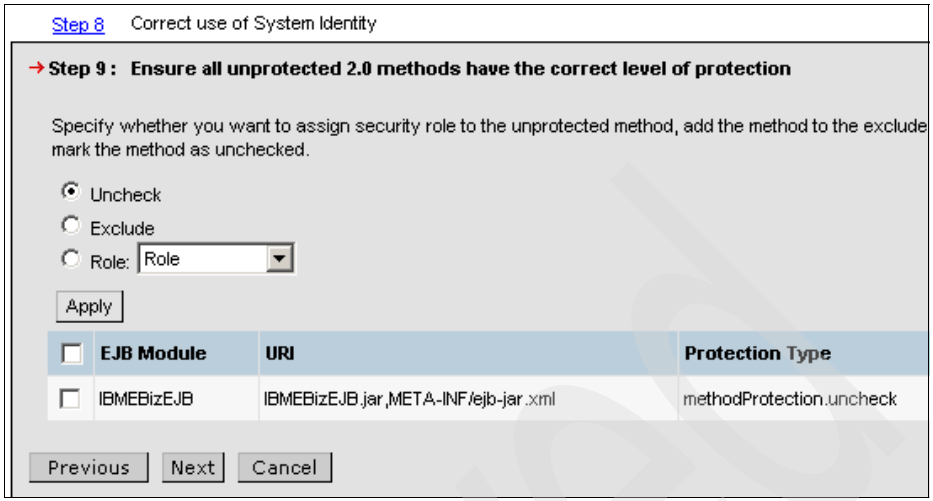


Figure 7-29 Administrative console unprotected methods

You can leave all of the defaults and click **Next**.

Finally, you are presented with a summary of the installation options (Figure 7-30).

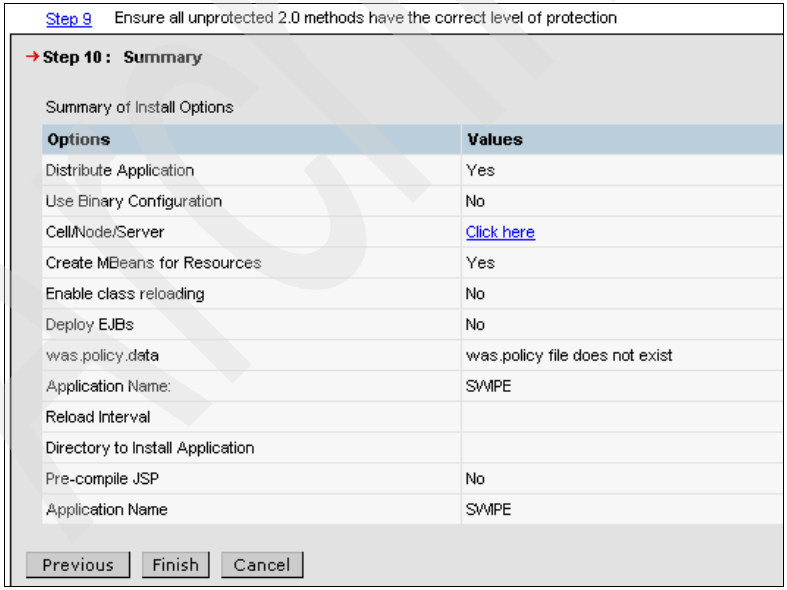


Figure 7-30 Administrative console summary

Click **Finish**.

After SWIPE has installed, you should see the messages shown in Figure 7-31.

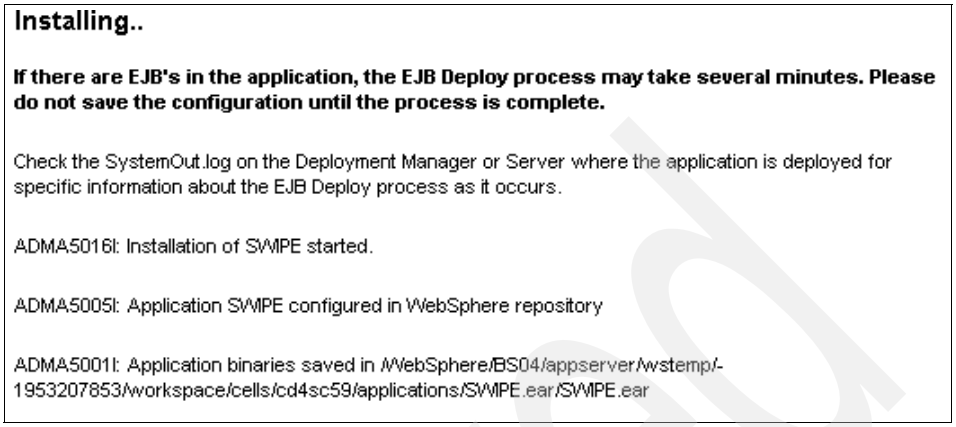


Figure 7-31 Administrative console installation completion

You can now click **Save to Master Configuration**, and **Save** on the next screen.

When the configuration is saved, you have to go to the Applications menu, click **Enterprise Applications**, select the check box in front of IBMEBiz, and click **Start**.

Successful start of the application is shown by a message similar to this:

```
Application SWIPE on server wd5nd5cd5sc59 and node nd5cd5sc59 started
successfully
```

In addition, the job log for the server displays the following message:

```
BBOOO222I WSVR0221I: Application started: SWIPE
```

7.5.1 SWIPE: JVM property for location discovery

In the EJBSample EJB there is a method called 'WhereAmI'. This method contains this Java statement:

```
System.getProperty("IBMEBizEnv");
```

This statement obtains the value that a JVM property called IBMEBizEnv has been set to. The WhereAmI method returns this value to the caller. The idea of this JVM property is that you can set it in the WebSphere server where SWIPE is deployed, and when you run SWIPE the value is displayed in the browser output.

This is useful when the SWIPE EJBSample EJB invokes the WhereAmI method of a SWIPE deployed in another WebSphere server, to verify that the remote call has been made.

7.5.2 SWIPE and Java 2 security

When enabling global security in WebSphere, it is common practice to disable Java 2 security, as suggested in 15.7.4, “Enabling global security on a base Application Server node” on page 490.

If you choose instead to enable Java 2 security, you should read 3.5.2, “Java 2 security” on page 66. Java 2 security enables you to secure low-level host system resources, such as files and TCP/IP sockets, and low-level SDK system classes and methods that are not normally accessed directly by J2EE application code.

Because of the nature of the resources being secured, a successful configuration of Java 2 security tends to be application specific, and the process of determining which resources are required by an application can be tedious and iterative. Enabling Java 2 security in a test environment WebSphere cell with only the SWIPE application installed provides a useful opportunity to explore the issues involved and the techniques required to complete a Java 2 security configuration.

Java 2 security configuration, from the perspective of the administrative console, is simply a check box that enables or disables the feature. The difficult task is to identify the changes that must be made to Java 2 security policy files. We present here the process that we used to identify policy file changes required for the SWIPE application.

Resources and techniques for identifying requirements

The first resource you will need to analyze policy requirements for your application is the Sun documentation about Java 2 security permissions, available at:

```
http://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html
```

The second resource you will need is the *WebSphere Application Server for z/OS Version 5 Information Center*, which identifies the policy files and three useful properties:

- Trace settings:
  - com.ibm.ws.security.core.SecurityManager=all=enabled
  - com.ibm.ws.security.policy.*=all=enabled
- Servant JVM property:
  - com.ibm.websphere.java2secman.norethrow=true

Search the *Information Center* using the names of the variables to find the most relevant pages. We found that the most useful property was the JVM property to

prevent the rethrow of security manager exceptions. When this is set to true, each Java 2 security violation results in one or more messages in the job log, and the application continues to flow through the logic normally.

Identifying SWIPE permission requirements

We execute SWIPE functions to identify the Java 2 permissions that are required for SWIPE to function correctly. One of the SWIPE functions is to read an arbitrary HFS ASCII text file specified by name in the browser. We create a directory structure containing text files on the HFS so that we can explore the effects of granting and blocking access to files with the Java 2 security policy:

```
drwxr-xr-x   4      8192 Jul 30 11:18 /policy_examples/
drwxr-xr-x   2      8192 Jul 30 11:17 /policy_examples/blocked
-rw-r--r--   1        14 Jul 29 11:00 /policy_examples/blocked/ascii.txt
drwxr-xr-x   2      8192 Jul 30 11:18 /policy_examples/permitted
-rw-r--r--   1        14 Jul 30 10:14 /policy_examples/permitted/ascii.txt
```

The plan is to use Java 2 policy to enable access to files within the permitted subdirectory, but not to those in the /blocked subdirectory.

We use the administrative console to define the servant region JVM custom property com.ibm.websphere.java2secman.norethrow=true, start the server, and request the following URL:

```
http://our_host_and_port/IBMEBizWeb/secure/RunAsServlet
```

We enter `/policy_examples/permitted/ascii.txt` in the “Name of file to read” field, and a valid OMVS user name and password in the JAAS user ID and login. Click the **Yes** button for JAAS login. After clicking the **Submit** button, we observe the following exception trace in the servant region’s job log:

```
./bborjtr.cpp+842 ... BBOS1007W Current Java 2 Security policy reported a
potential violation of Java 2 Security Permission. Please refer to Problem
Determination Guide for further information.
 Code:

     itso.pok.EJBSampleBean  in
{file:/WebSphere/BS01/appserver/installedApps/

permission: printIdentity
Message: access denied (java.security.SecurityPermission printIdentity)
```

The relevant portions of the trace are highlighted. First, we see that `itso.pok.EJBSampleBean` is causing the exception. Second, we see that the required permission is a `SecurityPermission` with a `printIdentity` target.

A stack trace follows this exception, and from the stack trace, we identify the line of code in itso.pok.EJBSampleBean that is causing the problem:

```
java.security.Principal callerID = mySessionCtx.getCallerPrincipal();
System.out.println("getEjb...getCallerPrincipal() is... " + callerID);
```

The highlighted line shows that the code is attempting to print an instance of `java.security.Principal`, which requires the Java 2 security permission `printIdentity`. This permission is required to view the name of a principal, the scope in which it is used, and its trust status in that scope. Explanations of this and other permissions are available at the following Web site:

`http://java.sun.com/j2se/1.3/docs/guide/security/permissions.html`

Another type of permission is seen in a second message that was found further down in the job log:

```
Message: access denied (java.io.FilePermission
/policy_examples/permitted/ascii.txt read)
```

The Java 2 security permission `FilePermission` is required for actions against a file, such as READ or WRITE.

Two additional types of permissions are seen further down in the log, arising from the JAAS login:

```
Message: access denied (javax.security.auth.AuthPermission
createLoginContext.WSLogin)

Message: access denied (javax.security.auth.AuthPermission doAs)
```

Again, you can use the information on the Sun Web site to learn about these permissions.

In summary, we find the following required permissions by searching the job log for the string "`permission:`":

```
java.security.SecurityPermission printIdentity
java.io.FilePermission /policy_examples/permitted/ascii.txt read
javax.security.auth.AuthPermission createLoginContext.WSLogin
javax.security.auth.AuthPermission doAs
```

Enabling SWIPE permissions in the policy

According to the *Information Center* documentation about Java 2 static and dynamic security policies, we can enable the required permissions at many levels, listed here from the most general to the most specific:

- All application servers in the cell
- A specific application server

- A specific application
- Specific Web or EJB modules within an application

The names and locations of the policy files in this hierarchy are documented in “Java 2 security policy” on page 69. The first three levels are separate files in different subdirectories of the WebSphere configuration directory. Policy at the last level, specific Web or EJB modules, is accomplished by specifying permissions for WebSphere-specific code bases entries in the application-specific policy file. Refer to Table 3-1 on page 72.

We choose a restrictive and specific security policy, in which we issue the minimum permissions required for SWIPE, and issue them only to the SWIPE application (that is, not to any servers or to any other applications); it is often true that each application requires a different set of permissions.

We used the following steps to enable SWIPE permissions in the policy:

1. We start by copying the was.policy file for the ivtApp application from the HFS down to a Microsoft Windows workstation using FTP binary mode.

   Important: We choose to start with an existing was.policy file because it contains the code base entries that can be processed successfully by the WebSphere software during server initialization. The names of the code bases are very specific. If the dynamic policy file contains code base entries with invalid names, the policy will fail to initialize during server initialization.

   The ivtApp file contains the following empty code bases:

   ```
   grant codeBase "file:${application}" {
   };
   grant codeBase "file:${jars}" {
   };
   grant codeBase "file:${connectorComponent}" {
   };
   grant codeBase "file:${webComponent}" {
   };
   grant codeBase "file:${ejbComponent}" {
   };
   ```

2. We then use a command line on the workstation to start the Policy Tool to edit the file so that we can add the required SWIPE permissions. The Policy Tool executable is included with the SDK and is in either the jre\bin\ or the bin subdirectory of a Java 1.4 installation, depending on how Java has been installed.

   We illustrate adding the SecurityPermission.

3. The main window, as shown in Figure 7-32, displays a list of “entries.” The was.policy file is a “dynamic” policy file, and as already noted, each of the entries has a special WebSphere-specific format. Do not add any new entries to this policy, or the policy will fail to work properly at runtime.

   Click the **Edit Policy Entry button** to display the permissions for the file:${application} code base (in other words, permissions to be issued to all code running in the SWIPE application, including servlets and EJBs).

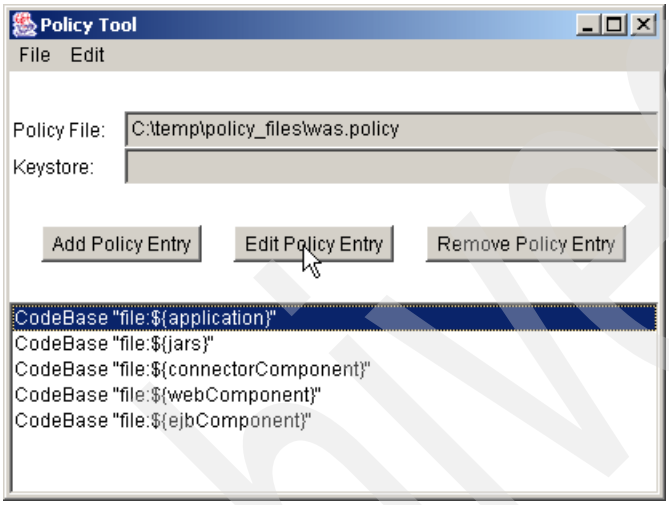


Figure 7-32 Policy Tool main window

4. The file:${application} entry is empty in the ivtApp was.policy. Click the **Add Permission** button, as shown in Figure 7-33 on page 164.

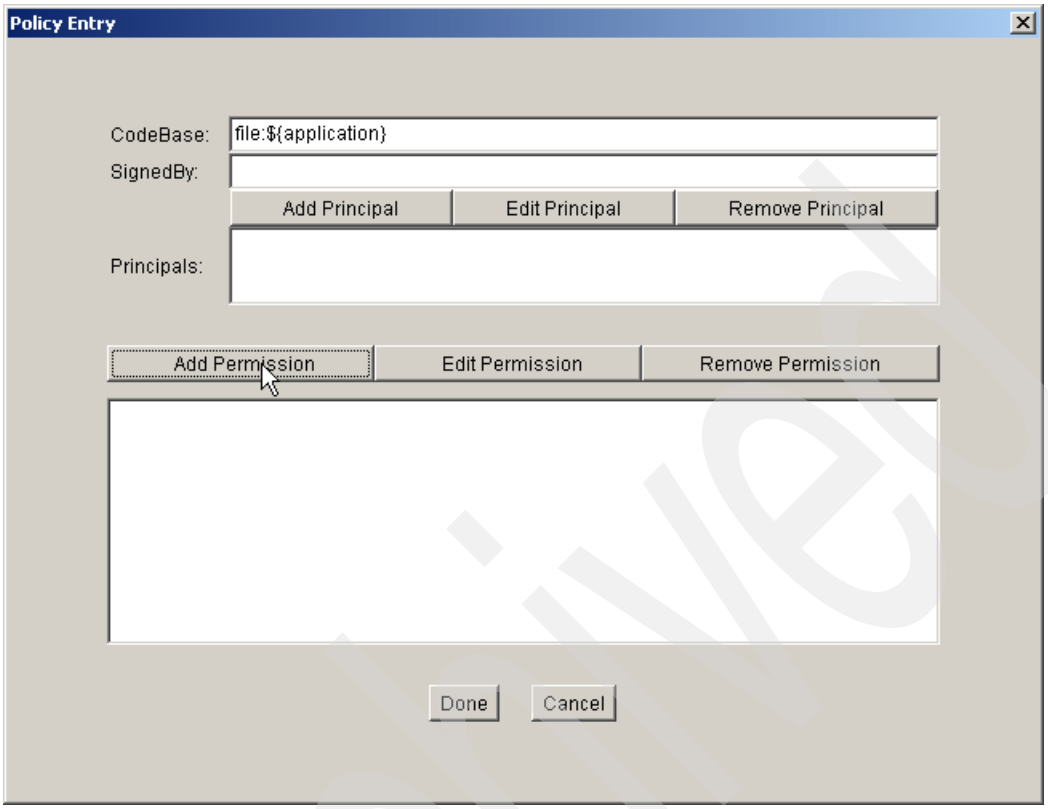


Figure 7-33 Policy Tool Policy Entry window

5. Select **SecurityPermission** from the top list box, and then select **printIdentity** from the second list box, as shown in Figure 7-34 on page 165. There are no actions associated with this target, so click **OK** to add the permission to the {application} entry.

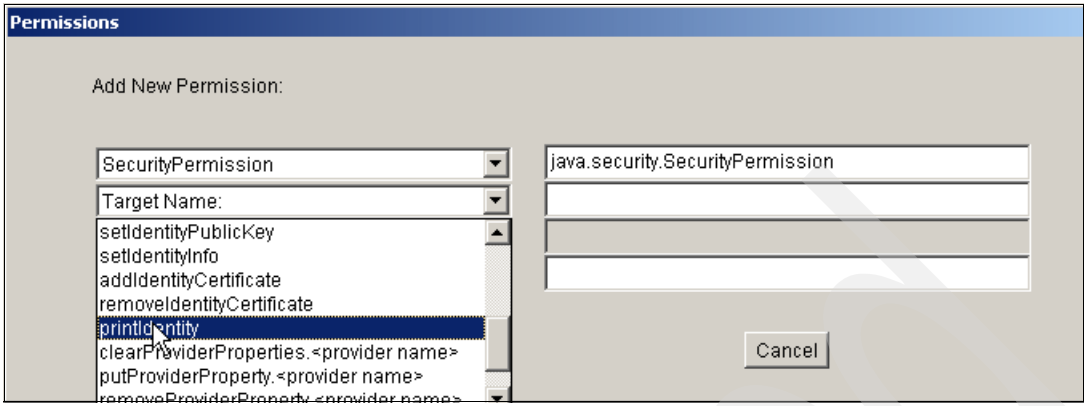


Figure 7-34 Policy Tool Add New Permission window

6. Now add the other three required permissions. When we add the `java.io.FilePermission`, we restrict the application to read access to all files in the /policy_examples/permitted directory, as shown in Figure 7-35.

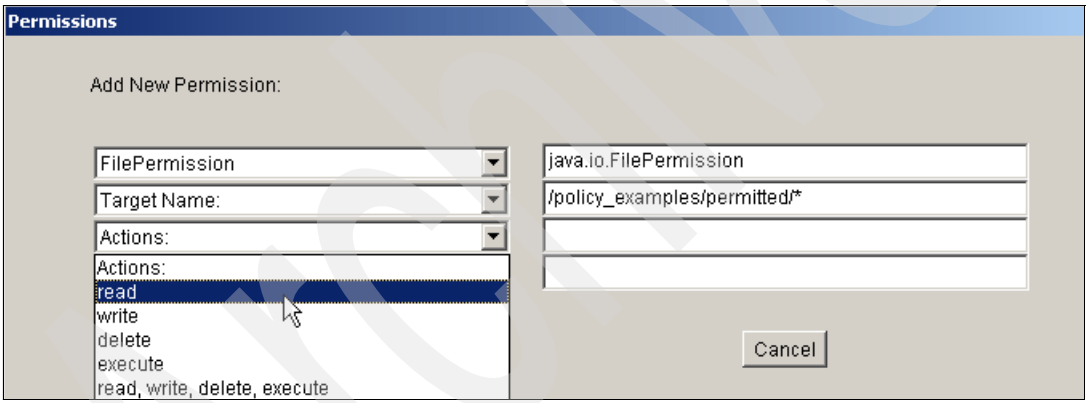


Figure 7-35 Policy Tool Add New Permission window: FilePermission

After saving the policy file, this is how the added permissions appear in the file:

```
grant codeBase "file:${application}" {
  permission java.security.SecurityPermission "printIdentity";
  permission java.io.FilePermission "/policy_examples/permitted/*", "read";
  permission javax.security.auth.AuthPermission
"createLoginContext.WSLogin";
  permission javax.security.auth.AuthPermission "doAs";
};
```

Testing the new SWIPE application policy

We then copy the modified was.policy file back to the HFS, using FTP binary mode, and place it in this subdirectory for WebSphere Application Server, as noted in the *Information Center* discussion of dynamic policy:

```
./config/cells/cd1sc59/applications/SWIPE.ear/deployments/SWIPE/META-INF
```

We set the permission bits and owner of the file so that they match the other files in the directory. We then use the administrative console to delete the “norethrow” custom property from the servant region JVM, start the server, and rerun the SWIPE test. This time, the page serves as expected, and there are no Java 2 security errors in the job log.

We then test the restrictions of our FilePermission by specifying the /policy_examples/blocked/ascii.txt file from the SWIPE browser page, and receive the expected exception when we request it from SWIPE:

```
Server caught unhandled exception from servlet [RunAsServlet]: access
denied (java.io.FilePermission /policy_examples/blocked/ascii.txt read)
```

Important: In a real business scenario, the was.policy file would be added to the /META-INF directory within the application .ear file so that the policy would be retained for future installations.

Exploring additional Java 2 security features

One important additional feature provided by WebSphere is the policy “filter.” This cell-level control enables specific permission types to be filtered from server-level and application-level policy files (in other words, application.policy and was.policy), thereby preventing the permissions from being granted even if they are specified in the policy files. The filter policy file is discussed in “Policy filters” on page 73.

This file is specific to WebSphere, and its structure is incompatible with the SDK Policy Tool; you must use a text editor to modify the file. We make a copy of the original file, and then edit the file, adding the line noted in bold to the `filterMask` stanza of the file:

```
filterMask {
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "setSecurityManager";
  permission java.security.SecurityPermission "setPolicy";
  permission javax.security.auth.AuthPermission "setLoginConfiguration";
  permission java.security.SecurityPermission "printIdentity";
};
```

After restarting the server, we check the job log and find this message as expected:

```
BBOOO222I SECJ0318I: The permission (java.security.SecurityPermission
printIdentity) specified in the application policy
file(/WebSphere/BSO1/appserver/config/cells/cd1sc59/applications/SWIPE
.ear/deployments/SWIPE/META-INF/was.policy) were filtered out.
```

We rerun the SWIPE test, and as expected, we receive an error page and see the permission exception in the servant region job log.

Restriction: The *Information Center* documentation recommends that additional permissions should not be added to the `runtimeFilterMask` stanza of the filter.policy file.

Conclusions

SWIPE provides an opportunity to explore the issues involved in enabling Java 2 security for a real business application. If you decide to enable Java 2 security in your system, you might want to experiment in your test environment with the SWIPE application, where you can try out other options for configuring the permissions and filters.

A real business application generally has many more functional paths than SWIPE, and this means that using the same process to identify required permissions is likely to be more time-consuming.

An alternative process is to find out whether or not the developers of your applications can provide a list of the necessary permissions based on source code analysis. This process would be based on reviewing the complete list of Java 2 permissions specified on the Sun Web site, and then using suitable search techniques to find uses of the protected classes and methods in the source code.

However, neither process (application testing versus source code analysis) is guaranteed to find all the required permissions. A Java 2 security implementation will require careful planning and testing and might require iteration of policy file changes within the production environment.

In addition, the following options should be reviewed before selecting the specific solution that you will use:

- Is standard J2EE security adequate without Java 2 security?

  This decision is probably based on your own familiarity with, and trust in, the application code that you run on your servers.

- Is the performance impact of Java 2 security acceptable?

  Additional processing is required for this feature. You might want to make your own measurements or talk with your IBM technical support before you enable Java 2 security.

- Do you want to design general or specific policy permissions?

  Our choice of policy for the SWIPE application is both specific (that is, at the application level) and restrictive (including only required permissions and restricting the FilePermission to one directory). There are many other ways to configure a real business application.

  There are several levels at which permissions can be issued, and for many of the permissions, such as FilePermission, there are different scopes that can be permitted. The filter policy can also be used to restrict certain permissions that you do not want any of your applications to have.

  Specific and restrictive permissions might be more difficult to establish and maintain, but offer a higher level of security. General permissions are likely to be easier to establish and maintain, but will not offer as much control over application behavior.

7.5.3 Setting the IBMEBizEnv environment variable

To set this JVM Property, in the WebSphere V5 administrative console, select **Servers** → **Application Servers**, and then click the server you want to modify. This will display information about the selected server. Make sure the configuration tab is chosen, and then under the heading Additional Properties click **Process Definition**. This will open the display shown in Figure 7-36.

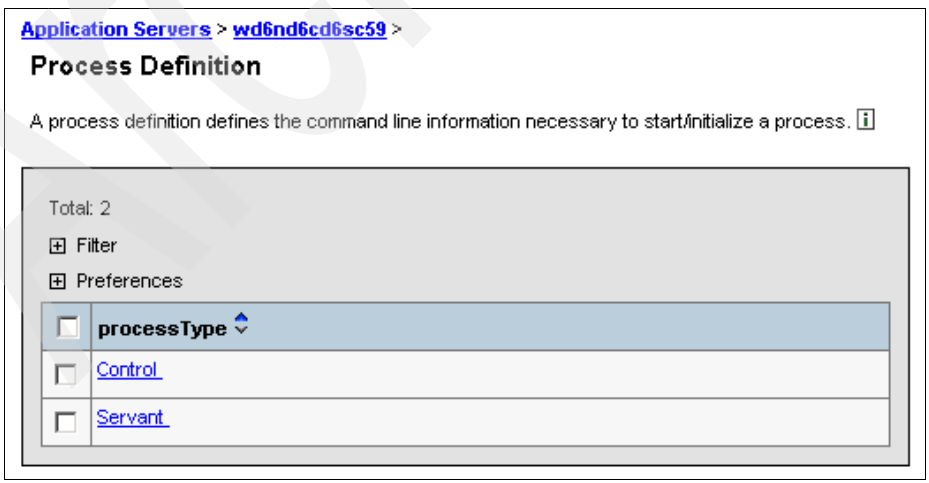


Figure 7-36 Administrative console: Process Definition Display

Because the SWIPE application runs in the servant region, click the **Servant** link and the display will then be as shown in Figure 7-37.

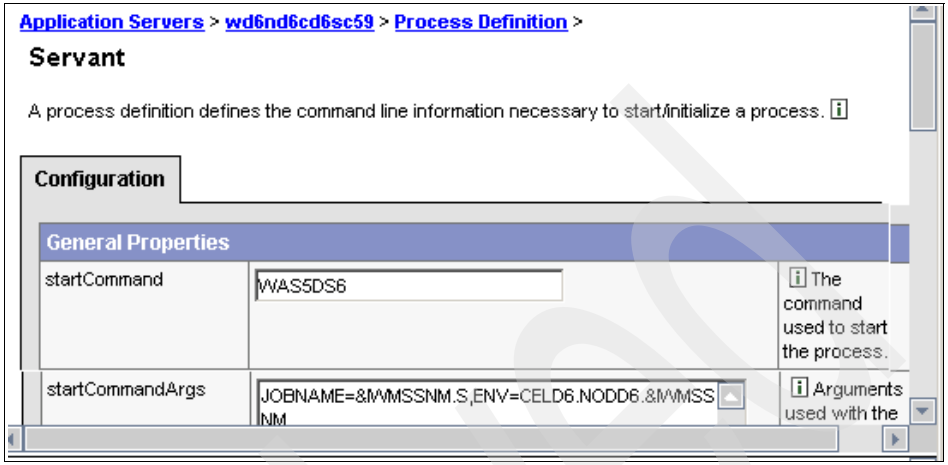


Figure 7-37 Administrative console: Servant display

Scroll down in the display and under the heading Additional Properties, click the link **Java Virtual Machine**, which will open the window shown in Figure 7-38.

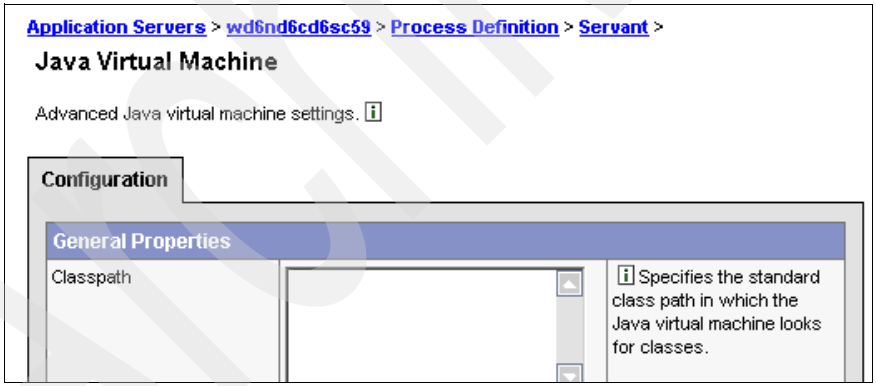


Figure 7-38 Administrative console: Java Virtual Machine settings

Scroll down in the display and under the heading Additional Properties, click the link **Custom Properties**, which will open the window shown in Figure 7-39 on page 170.

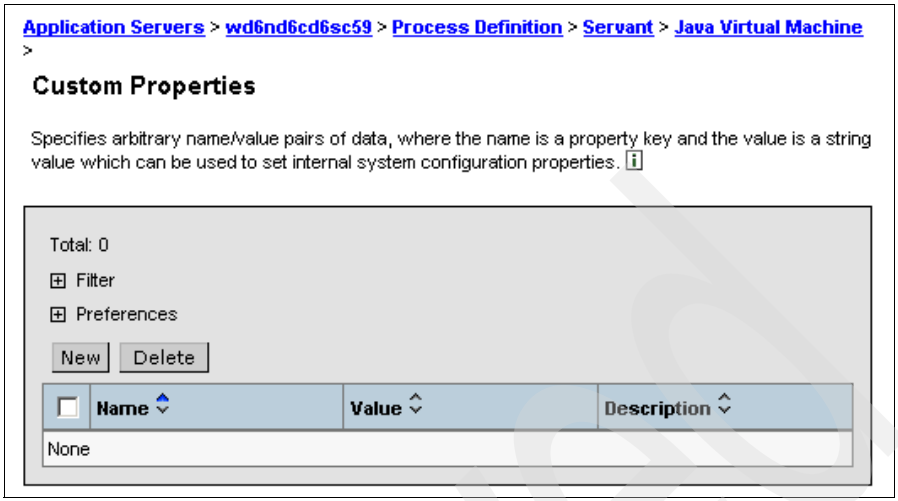


Figure 7-39 Administrative console: Custom Properties display

Click **New** and the administrative console will then display a form where you can enter the JVM Property. Figure 7-40 shows this display filled in with the SWIPE JVM Property. You can enter any string you want in the Value field.

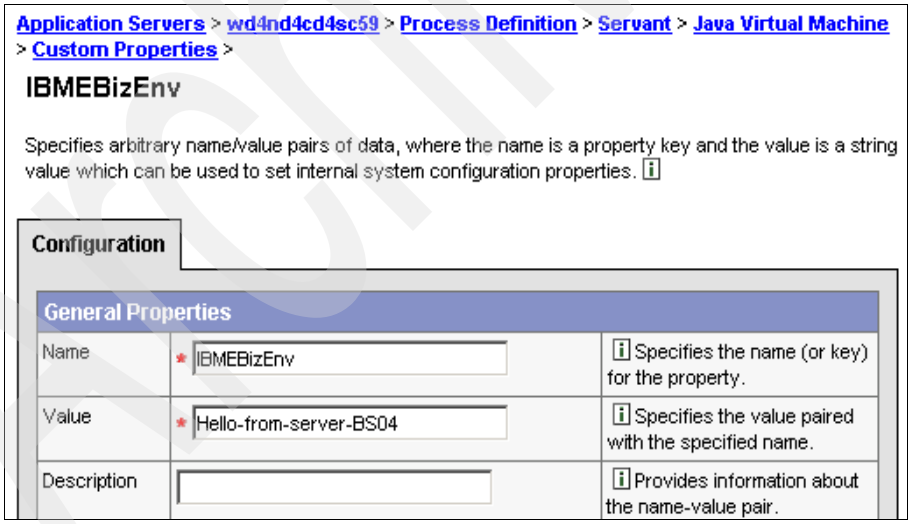


Figure 7-40 Administrative console: Setting the SWIPE JVM property

Important: Do not have any spaces in the value you specify for the Value field. If you enter a value like “Hello from server BS04” as opposed to “Hello-from-server-BS04”, your server will no longer start.

Click **Apply** and follow the normal save process to update the WebSphere configuration. The server will need to be restarted to pick up the change.

Verifying the result

When the EJBCaller servlet is run, the value you specify for IBMEBizEnv is displayed at the bottom of the window, similar to what is shown in Figure 7-41.

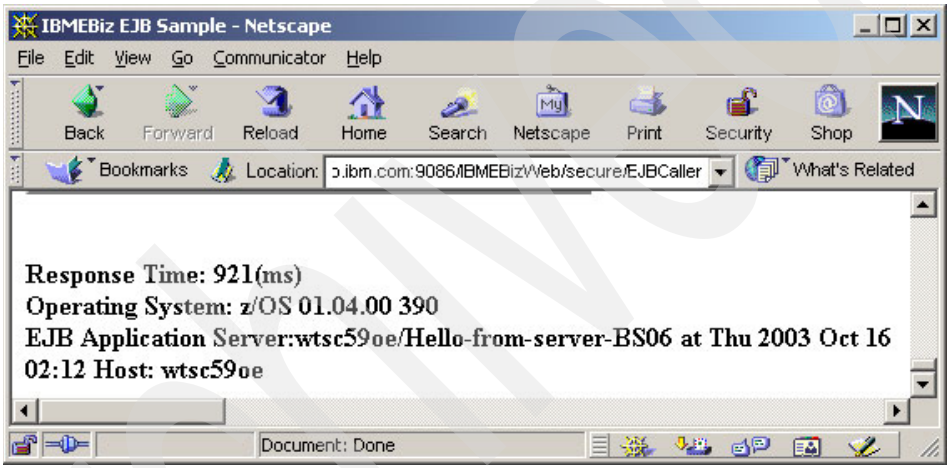


Figure 7-41 SWIPE output: Showing the value of IBMEBizEnv

If you use SWIPE in one server to call SWIPE in another server, you will see the value of the IBMEBizEnv displayed in the browser output. An example is shown in Figure 7-43 on page 173.

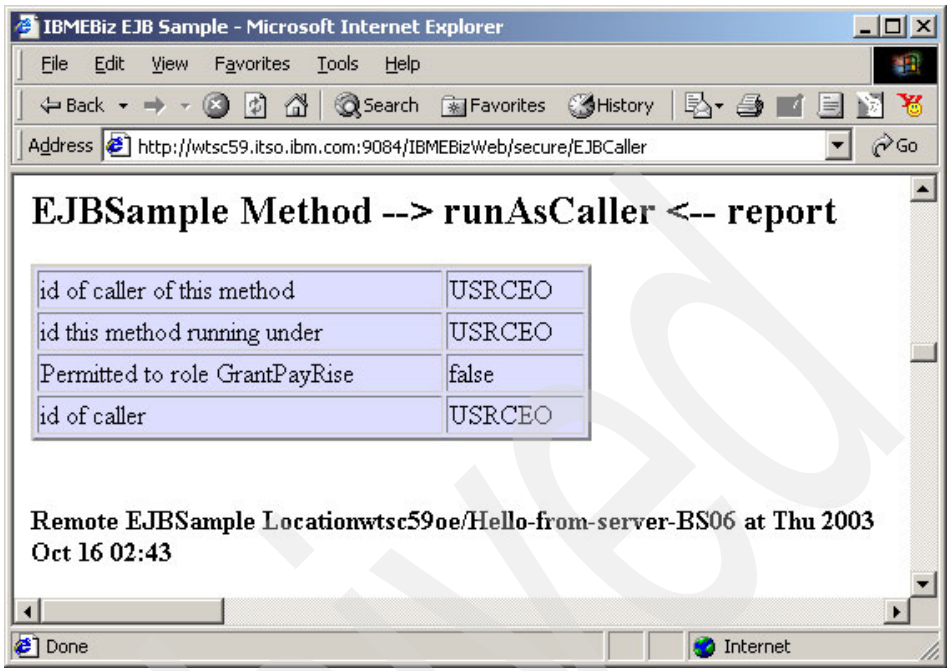


Figure 7-42 SWIPE output: Showing the value of remote IBMEBizWeb

7.6 SWIPE: Running EJBCaller

When the EJBCaller servlet of the SWIPE application is invoked and after authentication has completed, the display is the same, regardless of which URL is used to invoke it.

The output consists of three parts. The first part is shown in Figure 7-43 on page 173, the second part in Figure 7-44 on page 176 and the third part in Figure 7-47 on page 180.

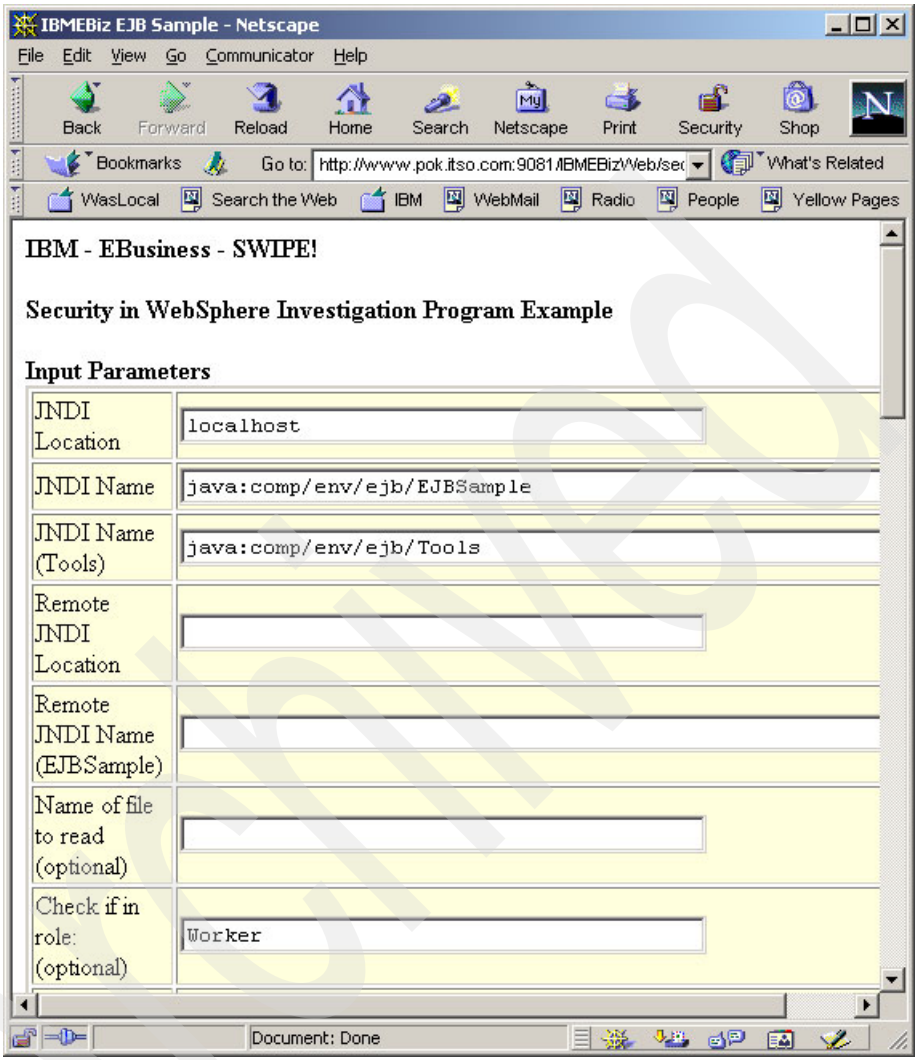


Figure 7-43 Part A of initial SWIPE output

7.6.1 SWIPE: EJBCaller - Input Part A

The first part is used to allow input of the parameters listed in Table 7-8.

Table 7-8 Part A: SWIPE input parameters

Field	Purpose
JNDI Location	Specify the value of the provider URL
JNDI Name	The JNDI value to use to look up the EJBSample EJB
JNDI Name(Tools)	The JNDI value the EJBSample EJB is to use to locate the Tools EJB
Remote JNDI Location	The value the EJBSample EJB is to use in the provider URL to locate another copy of the EJBSample EJB
Remote JNDI Name (EJBSample)	The JNDI value the EJBSample EJB is to use to locate another copy of the EJBSample EJB
Name of file to read (optional)	Name of a file that is to be read by the servlet and methods in the EJBSample EJB
Check if in role: (optional)	Name of an EJBRole that the servlet and methods in the EJBSample EJB should check access against

JNDI location

This field is set by the servlet to `localhost`. This value can be left blank or set to the location where the lookup should occur. If use, it is used in this code fragment:

Example 7-2 Code to set lookup location for initial context

```
                Hashtable env = new Hashtable();
                env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
                env.put(Context.PROVIDER_URL, jndiUrl);
                Context initialContext = new InitialContext(env);
                myCtx = new InitialContext();
```

JNDI Name

This field is set by the servlet to a value of `java:comp/env/ejb/EJBSample`. This is the preferred method to use when obtaining reference to an EJB. You can use the actual JNDI name assigned to the EJB if you want.

JNDI Name(Tools)

This field is set by the servlet to a value of `java:comp/env/ejb/Tools`. This is the preferred method to use when obtaining reference to an EJB. You can use the actual JNDI name assigned to the EJB if you want.

Remote JNDI Location

This field is not set by the servlet to any initial value. Use this field to identify the address of a WebSphere Application Server where another copy of the SWIPE application has been deployed. 7.6.6, “Remote JNDI example” on page 182 explains in more detail how to determine the JNDI value to use here. If it has been deployed into another J2EE server in the same WebSphere, use the value `localhost`. If SWIPE has been installed into a separate WebSphere, you would need to specify a value along these lines:

```
domainName:nnnn
```

Where `nnnn` is the port on which the WebSphere daemon listens.

Remote JNDI Name(EJBSample)

This field is not set by the servlet to any initial value. Enter in this field the JNDI value assigned to the EJBSample EJB when the second version of the SWIPE application was deployed into a WebSphere Application Server. 7.6.6, “Remote JNDI example” on page 182 explains in more detail how to determine the JNDI value to use here.

Name of file to read (optional)

Enter here the name of a file that the servlet, EJBSample, and the RunAs methods in EJBSample should try to access. If the file can be read, the first line is displayed in the browser. This is useful for checking which user ID is actually used to access a file in the z/OS UNIX HFS environment: Set the file protection bits so that no one can read it and then check the RACF access violation messages in the syslog.

Check if in role (optional)

Enter here the name of a role that the servlet, EJBSample, and the RunAs methods in EJBSample should perform a programmatic access check against. A value of `true` or `false` is returned.

7.6.2 SWIPE: EJBCaller - Input Part B

The second part of the initial SWIPE output is used to allow you to invoke a series of methods with different RunAs authorities set. Up to six RunAs methods can be invoked in sequential order. Additionally, for each RunAs method invoked,

an EJBRole can be selected to have the method perform a programmatic access check against.

Figure 7-44 shows where you can select the RunAs methods to be invoked and the EJBRoles to check access against.

Figure 7-44 Part B of initial SWIPE output

The SWIPE application lets you select any combination of six RunAs methods to be invoked.

Seven RunAs methods in the EJBSample EJB can be invoked; see Table 7-9 on page 177.

Table 7-9 The RunAs methods in the EJBSample EJB

RunAs method	Description
RunAsCaller	Will run with the principal ID set to the ID of the caller.
RunAsCallerSync	Will run with the principal ID set to the ID of the caller.
RunAsManagerRole	Will run with the principal ID set to the ID of the EJBRole Manager.
RunAsManagerRoleSync	Will run with the principal ID set to the ID of the EJBRole Manager.
RunAsCEORole	Will run with the principal ID set to the ID of the EJBRole CEO.
RunAsCEORoleSync	Will run with the principal ID set to the ID of the EJBRole CEO.
RunAsServer	Will run with the principal ID set to the ID of the J2EE server region.

To select the order and which RunAs methods to run, simply click the drop-down box to display all the RunAs methods and select the one you want to run. Repeat this process for each entry in the table.

There are six EJBRoles against which to perform a programmatic check:

- Worker
- Manager
- CEO
- GrantPayRise
- PromoteWorkers
- Company Car

When you click **Submit**, the servlet is invoked again and then calls the first nominated method on the EJBSample EJB, which then creates a new EJBSample object and calls the second method, and so forth. For each method called, a report is displayed in the browser showing information, as in Figure 7-45 on page 178.

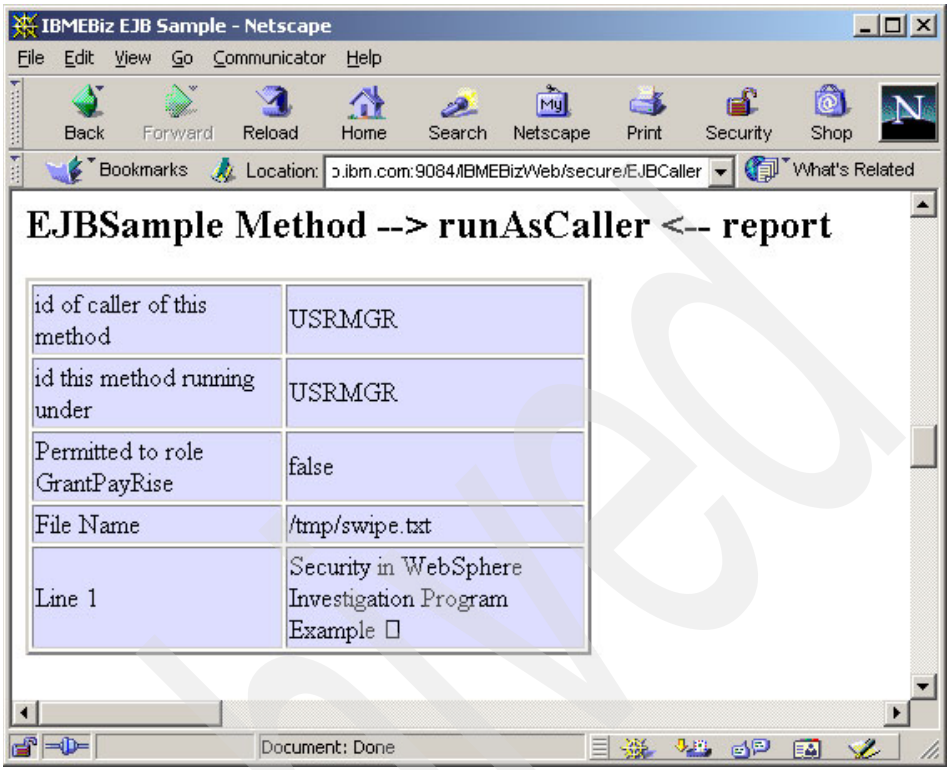


Figure 7-45 Sample SWIPE output for an invoked RunAs method

This output shows where we were logged on with user ID usrmgr and invoked the RunAsRoleManager method. The output is described in Table 7-10.

Table 7-10 Sample SWIPE output

Output field	Contents
ID of caller of this method	Displays the principal ID of the caller that called this method, in this case usrmgr.
ID this method is running under	Displays the principal ID that this method is running under, in this case ROLEMGR.
Permitted to role PromoteWorkers	Displays the result of the programmatic check, in this case `false`, as the caller usrmgr does not have access to the role PromoteWorkers.
FileName	Name of file the method is to access, in this case, /web/http/Unreadable.txt.

Output field	Contents
Exception	Because the file has had all read permissions removed, no one can read it, and an exception is reported here; if the method had been able to read the file, the first line would have been displayed.

If you try to call a RunAs method for which you are not authorized, an exception will occur, which is reported as shown in Figure 7-46.

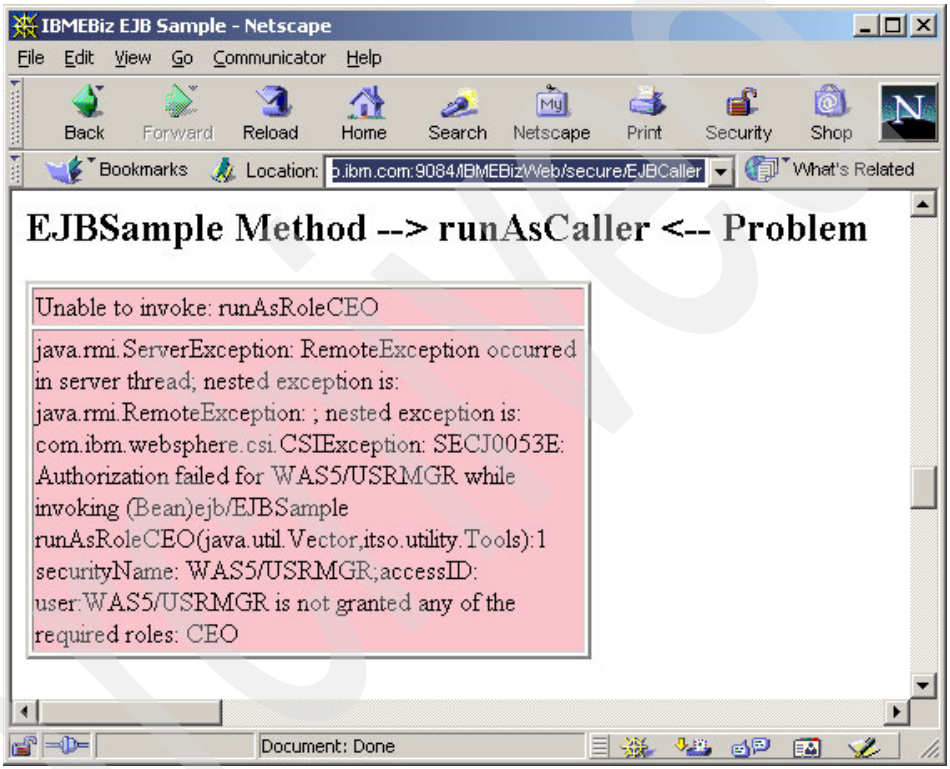


Figure 7-46 Sample output when not authorized to invoke a RunAs method

This shows that the RunAsCaller method tried to invoke the RunAsRoleCEO method. However, because the user ID usrmgr is not authorized to invoke this method, an exception was thrown.

7.6.3 SWIPE: EJBCaller - Input Part C, JAAS

The third part of the initial SWIPE output can optionally be used to try out a JAAS logon. When a JAAS login is tried, the SWIPE EJBCaller servlet still does what it

normally does, but additionally it then does a JAAS login, and then invokes the EJBSample EJB to show the ID being used to access the method of the EJB. Figure 7-47 shows where you can drive a JAAS logon using SWIPE.

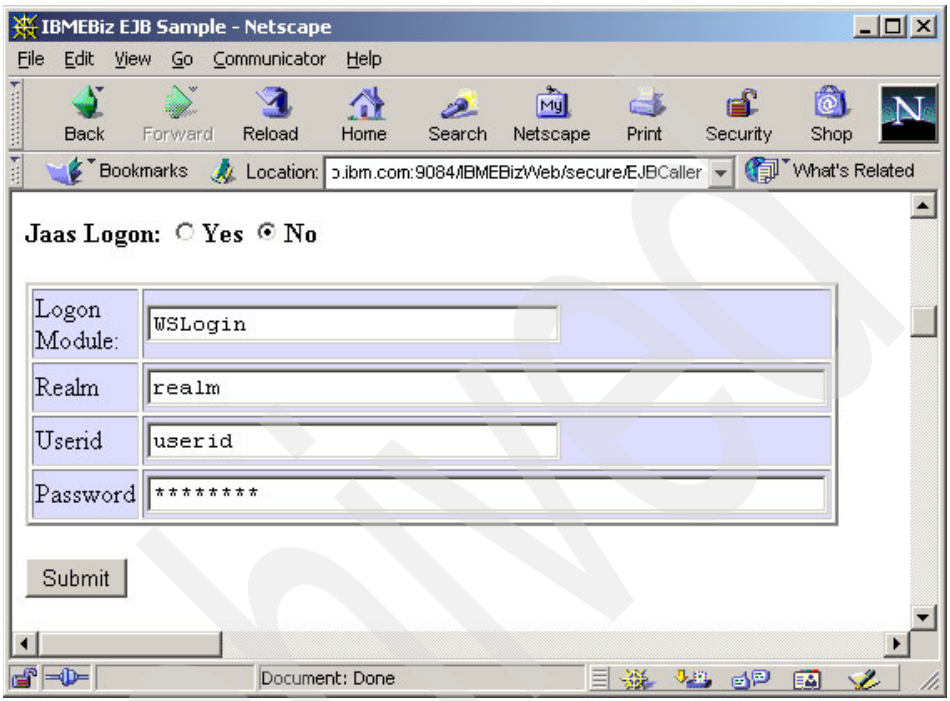


Figure 7-47 SWIPE: Jaas Logon input fields

Table 7-11 describes the fields in Figure 7-47.

Table 7-11 SWIPE: JAAS input field descriptions

Field	Description
Logon Module	This is the name of a defined application login configuration; WSLogin is a supplied default value in WebSphere.
Realm	Specifies the logon realm to use; leave as value “realm” when using WSLogin.
Userid	User ID to log on with.
Password	Associated password for the user ID.

To have SWIPE perform a JAAS logon, update the fields and select **Yes** before clicking **Submit**.

The actual line of code that performs the JAAS login is shown in Example 7-3.

Example 7-3 SWIPE: JAAS login code

```
lc =
   new LoginContext(
      jLogMod,
      new WSCallbackHandlerImpl(jUserid, jRealm, jPassword));
```

Figure 7-48 shows the start of the output SWIPE produces when a JAAS login is tried.

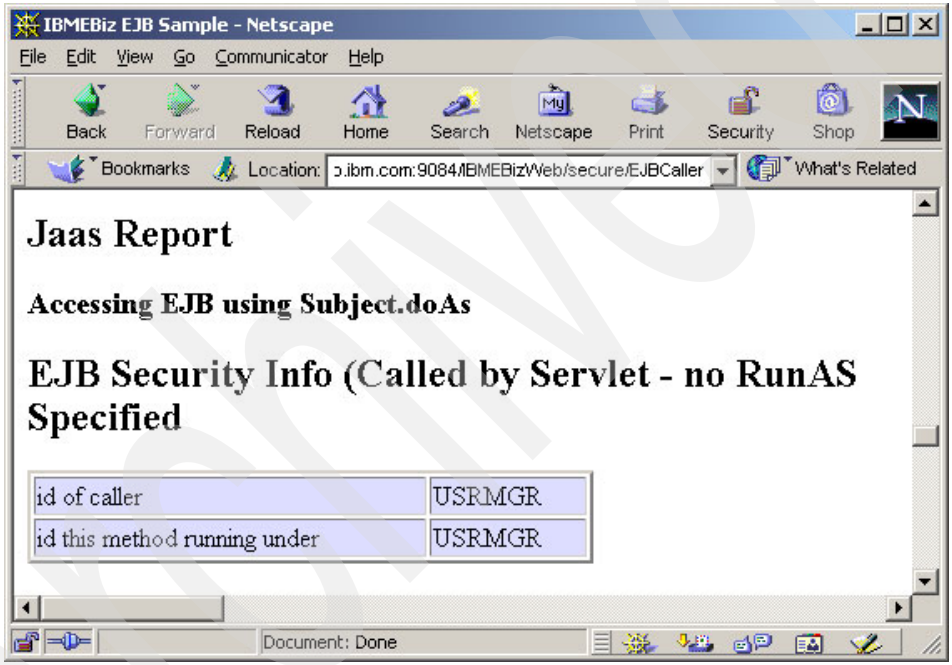


Figure 7-48 SWIPE: Results of trying a JAAS logon

7.6.4 SWIPE: RunAsServlet

The RunAsServlet demonstrates the use of a RunAs setting for a servlet. When it is run, the browser output is the same as described in 7.6.1, “SWIPE: EJBCaller - Input Part A” on page 174.

7.6.5 SWIPE: index.html

To make using SWIPE a little easier, this index page displays a number of the valid URLs that can be used. The URL to use to display this index page will be of the form:

```
http://domainName:Port/IBMEBizWeb/index.html
```

The output is shown in Figure 7-49.

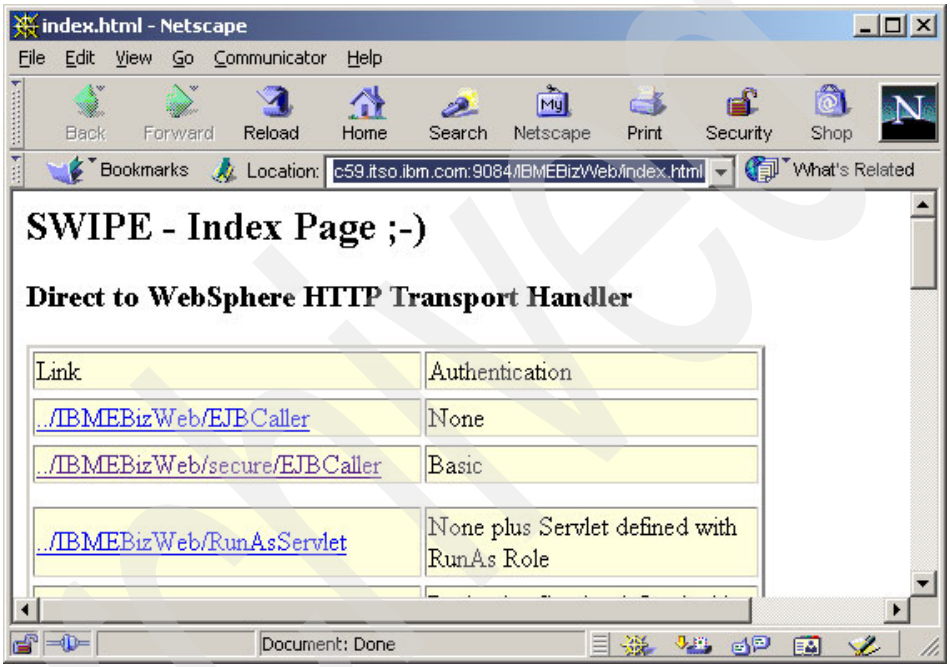


Figure 7-49 SWIPE: Index Page

7.6.6 Remote JNDI example

In 7.6.1, “SWIPE: EJBCaller - Input Part A” on page 174 was a brief description of how SWIPE can be used to call another SWIPE installed into another WebSphere Server. To do this requires supplying values in the two input fields called Remote URL and Remote JNDI Name (EJBSample).

Remote URL

In this field, you need to enter a value of the form:

```
<domainName>:<port>
```

Where <domainName> is the name of the machine where the WebSphere server is running, and <port> is the port the daemon is listening on.

In our testing, we entered the value:

```
localhost:2806
```

Remote JNDI Name (EJBSample)

In this field, you need to enter a value of the form:

```
cell/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59/ejb/EJBSample
```

Chapter 4 of *IBM WebSphere Application Server V5.1 System Management and Configuration: WebSphere Handbook Series*, SG24-6195, has an excellent description of JNDI naming concepts in WebSphere V5.

To work out what value to use in the Remote JNDI Name field, we used the dumpNameSpace.sh program. We accessed the directory corresponding to the directory of the server we wanted to access remotely. In this case we wanted to access SWIPE in a server called BS06 from a server called BS04.

In the directory /WebSphere/BS06/appserver/bin, we entered the following command, which dumped the name space of the BS06 server:

```
./dumpNameSpace.sh -port 2806
```

Part of the output is shown in Example 7-4. In the output, we looked for the JNDI name assigned to the EJBSample EJB.

Example 7-4 Name space contents

```
================================================================================
Beginning of Name Space Dump
================================================================================

    1 (top)
    2 (top)/nodes
javax.naming.Context
    3 (top)/nodes/nd6cd6sc59
javax.naming.Context
    4 (top)/nodes/nd6cd6sc59/persistent
javax.naming.Context
    5 (top)/nodes/nd6cd6sc59/nodename
java.lang.String
    6 (top)/nodes/nd6cd6sc59/domain
javax.naming.Context
    6    Linked to context: cd6sc59
    7 (top)/nodes/nd6cd6sc59/cell
javax.naming.Context
```

```
    7    Linked to context: cd6sc59
    8 (top)/nodes/nd6cd6sc59/servers
javax.naming.Context
    9 (top)/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59
javax.naming.Context
   10 (top)/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59/ejb
javax.naming.Context
   11 (top)/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59/ejb/EJBSample
   11
itso.pok._EJBSampleHome
_Stub
```

From this example, you can see that the JNDI name for the SWIPE EJBSample is:

```
/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59/ejb/EJBSample
```

The value to enter in the Remote JNDI Name field is then just this value prefixed with the value cell to give this:

```
cell/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59/ejb/EJBSample
```

7.7 RACF definitions

This section describes how to set up RACF to test out security in WebSphere using RACF.

7.7.1 Overview

There are a number of ways you can configure the RACF setup for this sample application so that a user has access to a role.

The approach we used is shown in Figure 7-50 on page 185. It shows examples of the following:

- Permitting a user specifically to an EJBROLE
- Permitting a group of users to an EJBROLE
- Use of GEJBROLE

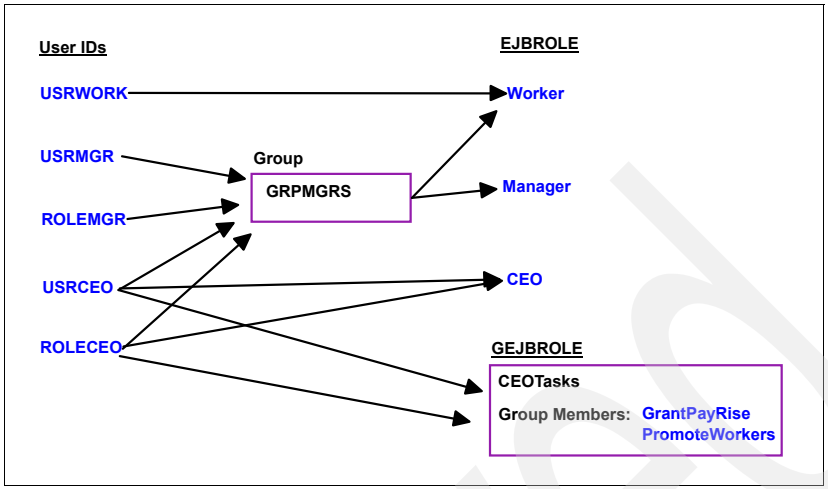


Figure 7-50 User ID to EJBRole mappings

Direct permission

The user ID USRWORK is given READ access to the EJBROLE Worker.

The user IDs USRCEO and ROLECEO are given READ access to the EJBROLE CEO.

Use of a group

The user IDs USRMGR, ROLEMGR, USRCEO, and ROLECEO are connected to the group called GRPMGRS. This group is then given READ access to the EJBROLE Workers, which results in all those users having access to that role.

Use of a GEJBROLE

The roles being used to demonstrate programmatic security, GrantPayRise and PromoteWorkers, where first defined as EJBROLEs. Then we defined an GEJBROLE called CEOTasks, which is a way to group EJBROLEs. We then added these two roles as members to the GEJBROLE definition. The user IDs USRCEO and ROLECEO are given READ access to the CEOTasks GEJBROLE.

Note: There is another EJBROLE used to demonstrate programmatic security. It is a role called CompanyCar but it is defined in the deployment descriptor in the .ear file. Also specified there is that only the CEO role has access to that logical EJBROLE. This allows you to contrast where security is specified. When it is specified in the deployment descriptor, changing access to it requires more steps than when the role is defined to RACF.

To test out RACF and SWIPE you need to configure global security in the WebSphere server so that the Active User Registry setting is set to Local OS.

7.7.2 Define user IDs

Note: The Web material available for this book contains a simple REXX script called swipeRACF.rexx. This file contains the commands listed in this section. If desired, it can be copied to a data set and executed by the desired user.

The following user IDs were used in our testing. Example 7-5 lists the commands we used.

- USREMP

  This user ID has the lowest level of access in the company. It is authorized to run the servlet, but not to run any of the methods in EJBSample.

- USRWORK

  This user ID has more access than USREMP and is authorized to access some methods in EJBSample.

- MRSHEEN

  This is the user ID coded in the APPLDATA field of the CLEANER EJB role.

- USRMGR

  This user ID has more access than USRWORK.

- ROLEMGR

  This is the user ID coded in the APPLDATA field of the Manager EJB role.

- USRCEO

  This user ID has the top level of access.

- ROLECEO

  This is the user ID coded in the APPLDATA field of the CEO EJB role.

- OMVS Segment ID

  All the user IDs are assigned an OMVS segment user ID number.

Example 7-5 Define user IDs

```
ADDUSER USREMP NAME('Employee') PASSWORD(USREMP) OMVS( UID (123))"
ALTUSER USREMP PASSWORD(USREMP) NOEXPIRED"
ADDUSER USRWORK NAME('Hard Worker') PASSWORD(USRWORK) OMVS(UID(124))
ALTUSER USRWORK PASSWORD(USRWORK) NOEXPIRED
ADDUSER USRMGR NAME('Company Mgr') PASSWORD(USRMGR) OMVS(UID(125))
ALTUSER USRMGR PASSWORD(USRMGR) NOEXPIRED
```

```
ADDUSER ROLEMGR NAME('Generic Mgr') PASSWORD(ROLEMGR) OMVS(UID(126))
ALTUSER ROLEMGR PASSWORD(ROLEMGR) NOEXPIRED
ADDUSER USRCEO NAME('Comp. CEO') PASSWORD(USRCEO) OMVS( UID (127))
ALTUSER USRCEO PASSWORD(USRCEO) NOEXPIRED
ADDUSER ROLECEO NAME('Generic CEO') PASSWORD(ROLECEO) OMVS(UID(128))
ALTUSER ROLECEO PASSWORD(ROLECEO) NOEXPIRED
ADDUSER MRSHEEN NAME('Mr Sheen') PASSWORD(MRSHEEN) OMVS(UID(129))
ALTUSER MRSHEEN PASSWORD(MRSHEEN) NOEXPIRED
```

7.7.3 Define a group

We created a group to group all the managers:

```
ADDGROUP GRPMGRS
```

We then connected the manager type users to this group with these commands:

```
CONNECT USRMGR GROUP(GRPMGRS)
CONNECT ROLEMGR GROUP(GRPMGRS)
CONNECT USRCEO GROUP(GRPMGRS)
CONNECT ROLECEO GROUP(GRPMGRS)
```

7.7.4 Define EJBRoles

The sample application uses six EJBRoles, similar to what might be used at a company. We defined the following roles. Example 7-6 on page 188 lists the commands we used.

- Employee

  This EJBRole signifies all the employees of a company. You need to be in this EJBRole to be able to run the EJBCaller servlet.

- Worker

  This EJBRole signifies a role for employees of the company that are classified as workers.

- Manager

  This EJBRole signifies a role for employees of the company that are classified as managers.

- CEO

  This EJBRole signifies a role for employees of the company that are classified as CEOs.

- Cleaner

  This EJBRole signifies a role for the people who have cleaning duties at the company.

- GrantPayRise

  This EJBRole is a logical role used for demonstrating programmatic security.

- PromoteWorkers

  This EJBRole is a logical role used for demonstrating programmatic security.

Example 7-6 Define EJBRoles

```
RDEFINE EJBROLE Employee UACC(NONE)
RDEFINE EJBROLE Worker UACC(NONE)
RDEFINE EJBROLE Manager UACC(NONE) APPLDATA('ROLEMGR')
RDEFINE EJBROLE CEO UACC(NONE) APPLDATA('ROLECEO')
RDEFINE EJBROLE Cleaner UACC(NONE) APPLDATA('MRSHEEN')
RDEFINE EJBROLE GrantPayRise UACC(NONE)
RDEFINE EJBROLE PromoteWorkers UACC(NONE)
```

Purpose of APPLDATA

The value specified for APPLDATA is the RACF user ID that comes into effect if you use the RunAs role approach. If a method is configured to run as an EJBROLE, the user ID specified in the APPLDATA field is the user ID that the process runs as in WebSphere.

7.7.5 Define GEJBROLE

We then defined a GEJBROLE called CEOTASKS using this command:

```
RDEFINE GEJBROLE CEOTASKS UACC(NONE)
```

We then added as members the logical roles with this command:

```
RALTER GEJBROLE CEOTASKS ADDMEM(GrantPayRise, PromoteWorkers)
```

7.7.6 Permit access

The commands shown in Example 7-7 were issued to assign the user IDs to the roles to which they are meant to have access.

Example 7-7 Permit access commands

```
PERMIT Employee CLASS(EJBROLE) ID(USREMP) ACCESS(READ)
PERMIT Employee CLASS(EJBROLE) ID(USRWORK) ACCESS(READ)
PERMIT Employee CLASS(EJBROLE) ID(GRPMGRS) ACCESS(READ)
PERMIT Employee CLASS(EJBROLE) ID(MRSHEEN) ACCESS(READ)
PERMIT Worker CLASS(EJBROLE) ID(USRWORK) ACCESS(READ)
PERMIT Worker CLASS(EJBROLE) ID(MRSHEEN) ACCESS(READ)
PERMIT Worker CLASS(EJBROLE) ID(GRPMGRS) ACCESS(READ)
PERMIT Manager CLASS(EJBROLE) ID(GRPMGRS) ACCESS(READ)
```

```
PERMIT CEO CLASS(EJBROLE) ID(USRCEO) ACCESS(READ)
PERMIT CEO CLASS(EJBROLE) ID(ROLECEO) ACCESS(READ)
PERMIT CEOTasks CLASS(GEJBROLE) ID(USRCEO) ACCESS(READ)
PERMIT CEOTasks CLASS(GEJBROLE) ID(ROLECEO) ACCESS(READ)
```

If not already done, the EJBROLE class must be activated. In order to activate the changes to the class, a RACLIST or RACLIST REFRESH of the EJBROLE class is needed:

```
SETROPTS CLASSACT(EJBROLE)
SETROPTS RACLIST(EJBROLE) GENERIC(EJBROLE) REFRESH
```

7.7.7 Verify security using SWIPE

Having defined all the required RACF definitions, you can now experiment with the various J2EE Authorization possibilities.

Choose a URL from the ones shown in 7.1.1, “SWIPE application structure” on page 124, or open the SWIPE index page and then select a link from there.

You could choose to run the servlet without authentication to see what effect that has on access to the RunAs methods. Or you could choose a URL that requires authentication by the Web container, and use one of the user IDs defined to see how access to the various RunAs methods works out.

After you have run the servlet and received the JSP output in the browser, you can then:

1. Specify the sequence of RunAs methods that are to be invoked. Up to six in any combination can be specified. You can also specify which EJBROLE names are to be checked in a programmatic fashion.
2. Specify which EJBROLEs are to be checked in a programmatic way through the isCallerInRole API.

Sometimes when you run a test, it might be that the principal ID of the caller does not have access to a RunAs method. When this happens, the output in the browser will show this in a pale red shaded box. Receiving this message indicates that security is working the way you wanted it to, assuming you have defined the RACF rules correctly.

8

The security investigation applications for EIS

In this chapter we introduce you to our security investigation applications for CICS, IMS, and DB2 connectors. We explain how the sample applications work, how to install them, and how we configured the deployment descriptors and our WebSphere server so we could test multiple scenarios.

Enterprise information system (EIS) has previously been used as a collective term for systems like CICS, IMS, and SAP, for example, but we do not usually discuss database connections using JDBC as being connections to an EIS. With the gradual convergence in the architecture of JDBC and J2CA adaptors, the way you use JDBC is increasingly similar to the way you use J2CA and so we can think of IMS/DLI and DB2 databases accessed by JDBC as being examples of EIS.

8.1 The SWIPE application for CICS, IMS, and DB2

We decided to create separate applications to look at the security aspects of J2CA connectors because there are a large number of different properties you can set to influence the user ID propagated to your EIS, and putting all this into SWIPE would have made SWIPE too cluttered.

You can download our security investigation application from the Internet to check if your security settings deliver the expected results. See “Locating the Web material” on page 717 for downloading instructions. If you play around with this application, you can very quickly get a feel for security within your environment and begin to understand how the whole process works, from writing the application to deploying it all the way to z/OS and OS/390.

8.1.1 How SWIPE for EIS works

To allow us to test the various security scenarios, we developed three small applications that we named SWIPECICS, SWIPEIMS, and SWIPEDB2. The SWIPE acronym stands for Security in WebSphere Investigation Program Example. We used WebSphere Studio Application Developer V5 Integration Edition to develop the applications. You can install this application on your own system and use it to experiment with security settings in IBM WebSphere Application Server for z/OS.

This section is an overview of how SWIPE for EIS works.You need to have a basic understanding of J2EE security concepts to understand the functionality of the three SWIPE applications. If you are unfamiliar with J2EE security concepts, review Chapter 3, “J2EE 1.3 and WebSphere Application Server V5 security concepts” on page 41.

Note: Collectively we will refer to these three applications as SWIPE/EIS. They were developed in a similar way and so have a lot in common with each other. In this section we use a lower case *eis* where you need to substitute your choice of EIS type (CICS, IMS, or DB2).

Servlet authentication[1]

The SWIPE applications for CICS, IMS, and DB2 each have a main servlet that can be invoked using Basic authentication method. The purpose of these samples is not to test every kind of authentication method. That is covered by the main SWIPE application. Instead we decided to implement only basic

[1] Authentication is the process by which an entity (for example, a server) verifies the identity presented by a user. This process usually requires some form of credential information such as a password known only to the user.

authentication so that we had an authenticated user running the applications. This becomes important when you look at how the unique WebSphere for z/OS thread identity support works.

Each application is driven by an index.jsp on which there is an input box that you can use to pass the name of a resource reference to the EJB component that makes the J2CA connection to the EIS. You can use any resource reference names you like on the index.jsp panel, but clearly you must bind any resource references you decide to use to the JNDI name of connection factories that you create using the WebSphere administrative console.

This should become clearer when you read about the setup instructions later.

By using different resource references you can drive requests through different connection factories that have different properties. For example, you could have a connection factory configured for a local mode connection and one for remote mode. You can test non-SSL and SSL. You can test the use of JAAS authentication aliases. Many different test scenarios are possible by coding variations in the resource reference definition (Res-Auth=Container or Application) and variations in the Custom Properties of connection factories.

The SWIPE/EIS applications already have Web deployment descriptors that include a role in the form *eis*ServletManager. If you choose to use this role you will need to create a RACF EJBROLE profile for *eis*ServletManager.

EJB authorization[1] and propagation of security identity

The SWIPE/EIS application also consists of a session EJB with one remote method defined. The aim here is to demonstrate the following:

- EJBROLEs and RunAs
- Declarative security
- ThreadIdentity and ThreadSecurity support

The SWIPE/EIS applications already have EJB deployment descriptors that include various resource references and roles. If you choose to use these, you will need to create connection factories that use the JNDI names coded on the resource references and you will need to create a RACF EJBROLE profile of *eis*EJBManager. The method permissions for the EJB (set in the EJB deployment descriptor) have been set so that both roles, *eis*ServletManager and *eis*EJBManager, can run the EJB.

[1] Authorization is the process of determining what type of access (if any) the security policy allows to a resource (for example, file or methods) by a principal.

8.1.2 SWIPE EIS application structure

This section describes the contents of the sample application including servlet, servlet URL mappings, and the protection settings for each. It focuses on what J2EE resource references have been set up to try them out.

SWIPE application contents

The sample J2EE applications are packaged in a SWIPE*eis*.ear file, (SWIPECICS.ear, SWIPEIMS.ear, and SWIPEDB2.ear), which contains:

- SWIPE*eis*EJB.jar: Contains the session EJB class.
- SWIPE*eis*Web.war: Contains the servlet class, configured to use basic authentication.
- SWIPE*eis*Common.jar: Contains auxiliary classes that use the CCI API to access your EIS. Actually, this .jar file is only available for IMS and DB2 EIS types, but new versions of SWIPE for CICS might use this same kind of model.

Note: The SWIPECICS.ear is supplied in a SWIPECICS.zip file that also contains some procedural language components for installation in CICS.

Context root

Each .war file has a different context root and a servlet, as shown in Table 8-1.

Table 8-1 Contents of a .war file

.war file	Context root	Servlets
SWIPEDB2Web.war	SWIPEDB2Web	SWIPEDB2Servlet
SWIPECICSWeb.war	SWIPECICSWeb	SWIPECICSServlet
SWIPEIMSWeb.war	SWIPEIMSWeb	SWIPEIMSServlet

URL mappings

Within each .war file, unique URL mappings are defined for each servlet. Instead of driving the servlet mapping directly, you should use the index.jsp mapping that has been created to allow you to drive the application from a browser with:

```
http://websphere_hostname/swipeeis
```

index.jsp

This JSP is a simple HTML form that allows you to select various options and pass data to SWIPE*eis*Servlet using the POST method invocation.

SWIPE eisServlet

This servlet is the core servlet containing the code to invoke the EJB session bean. The servlet gets the input values filled by the index.jsp and passes it to the session EJB. Results are displayed on results.jsp and errors on error.jsp.

SWIPE eisTester

This is the session EJB that uses the common package SWIPE*eis*Common to call the EIS.

8.1.3 Define security roles for SWIPE/EIS

The sample applications use the following roles:

- *eis*EJBManager
- *eis*ServletManager

These roles and related security definitions have already been created in the deployment descriptors, so this section just explains what we did in customizing the deployment descriptors. If you want to use the SWIPE/EIS applications to learn how security works, it is probably best to use our roles and resource references at first and then, after you understand what is happening, change the deployment descriptors and connection factories to perform your own specific tests.

Roles and permissions for SWIPE/EIS Web components

The security roles were defined by adding them to the ejb-jar.xml file for EJB, and to the web.xml file for the servlet.

Note: We will use the swipedb2.earfile in these illustrations, but the configuration for other applications is very similar.

Figure 8-1 on page 196 shows the WebSphere Studio Application Developer display where we did this configuration for the servlet.

Roles are added by simply clicking **Add** and defining them.

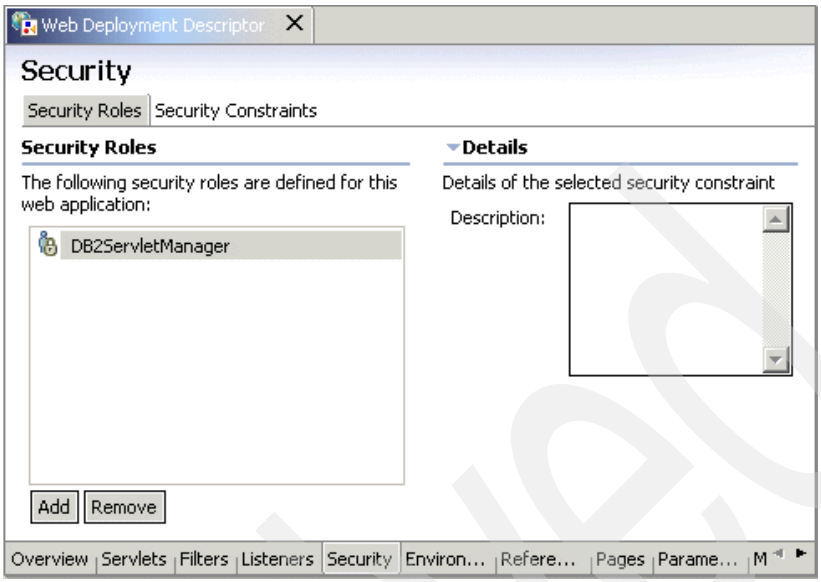


Figure 8-1 Defining EJB roles for Web container in WebSphere Studio: Part 1

Figure 8-2 shows the equivalent AAT display.

Roles are added by clicking on the Security Roles item in the left navigation window and selecting **New** from the drop-down list box.

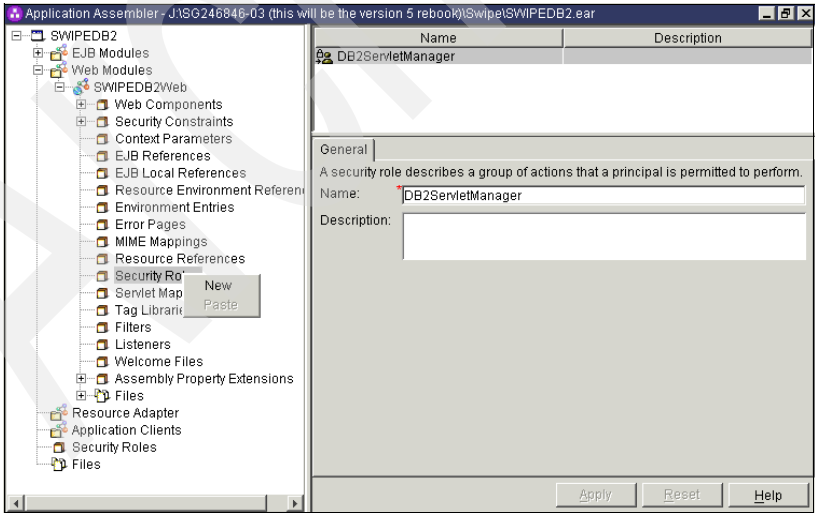


Figure 8-2 Creating a security role for the Web modules using the AAT

After the role exists in web.xml, you can define a security constraint. The security constraint describes the nature of the authorization check to be performed when the Web components are accessed. You can assign the security role that the user must have in order to run the Web application.

Using WebSphere Studio Application Developer, click the **Security Constraint** tab and then **Add** to add the constraint.

Then move to the Web Collection box and click **Add** to add the Web Collection. The Web Collection is where you specify the URL mask that the security constraint will apply to. You can also say which HTTP methods (for example, GET or POST) you want the security constraint to apply to.

Then on the Authorized Roles box on the bottom right-click **Edit** and select the role you want to be able to use the Web application.

Your WebSphere Studio Application Developer window should like Figure 8-3 when you have finished.

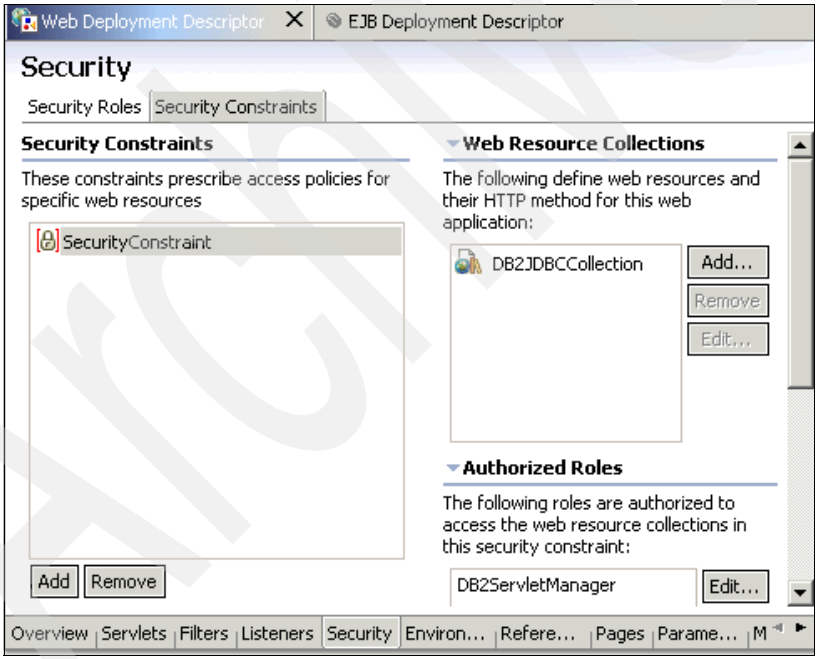


Figure 8-3 Defining Security Constraints for the Web application

With the AAT you would right-click **Security Constraints** in the navigation panel on the left, and then select **New** from the drop-down list. Give the security constraint a name and then click **Add** in the Authorization Constraints box. This

opens a window and you can select the role you want to allow access to the Web components.

Roles and permissions for SWIPE/EIS EJB components

After creating the role *eis*ServletManager for use in the Web container, we opened the ejb.xml file and created role *eis*EJBManager for use with EJBs. This is shown in Figure 8-4.

Next we set Method Permissions on the EJB so that only people in role *eis*EJBManager could run the methods of the EJB.

Using WebSphere Studio Application Developer, open the EJB Deployment Descriptor and go to the Assembly Descriptor tab. Then click **Add** in the Method Permissions box to select the role that is allowed to run the methods of the EJB.

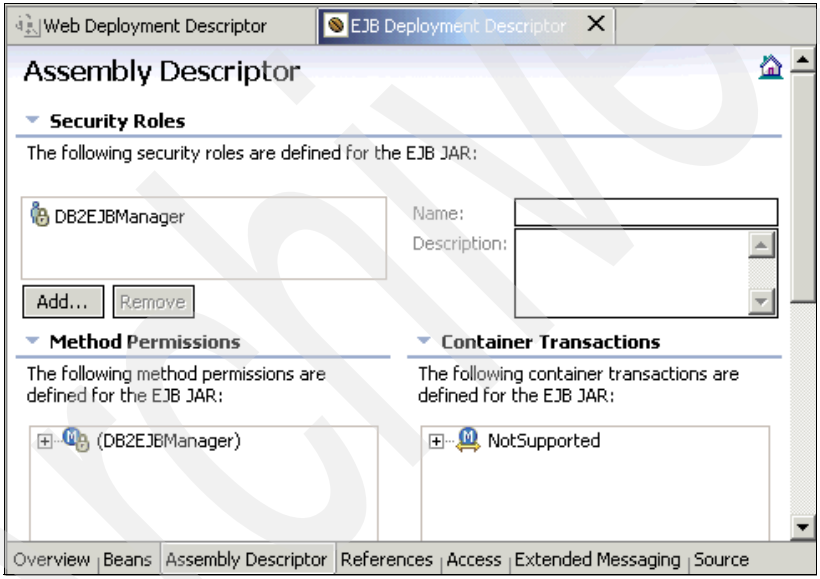


Figure 8-4 Roles and method permissions using Application Developer

Note: A role must always map to a user ID, so if you associate the role *eis*ServletManager with a RunAs for the servlet, the user ID which *eis*ServletManager maps to must have permission to run the EJBs.

You can achieve this by making sure that the user ID is in the role *eis*EJBManager, or you could add the role *eis*ServletManager to the Method Permissions for the EJB.

Now that roles have been defined for the Web and EJB components, you should find that the roles are now visible in the J2EE application EAR deployment descriptor. If you open the EAR deployment descriptor with WebSphere Studio Application Developer, it should look like that in Figure 8-5.

Tip: The roles in the application deployment descriptor are added by WebSphere Studio Application Developer depending on what you define in the Web and EJB deployment descriptors. If you want to delete a role, you need to delete it from the Web or EJB component descriptor and then from the application deployment descriptor. If you just try to delete it from the application EAR deployment descriptor, WebSphere Studio Application Developer will add it back again.

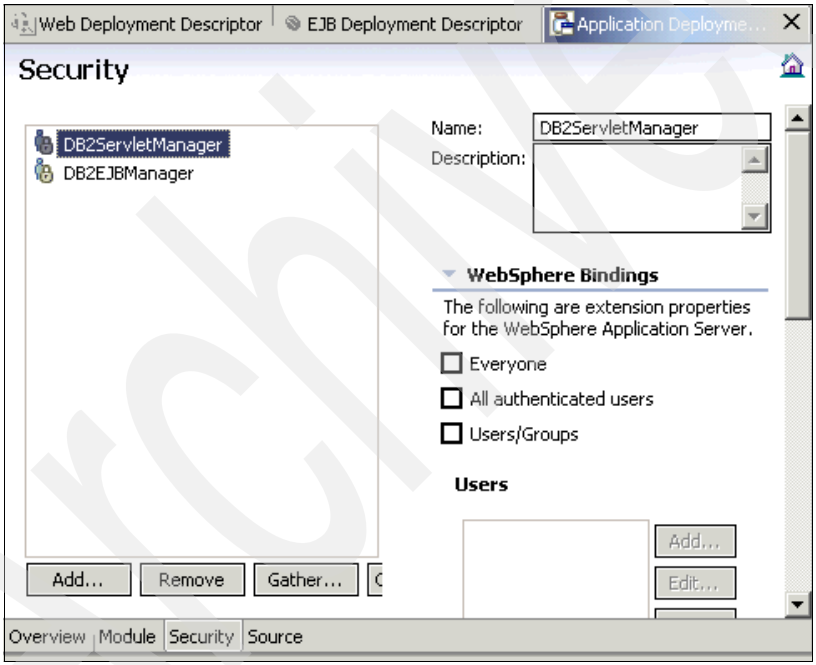


Figure 8-5 Roles in EAR deployment descriptor using Application Developer

Note: In Figure 8-5, there are WebSphere bindings, which are the J2EE way to map roles to users and groups of users. This mapping is made in XML files. With WebSphere Application Server for z/OS, we usually prefer to use RACF EJBROLEs to make this mapping from role to user ID, and therefore these options are left unused.

Setting RunAs for the servlet, EJB, or both

Finally, and of interest when using J2CA connectors, is the setting of RunAs.

If you do not set a value for RunAs, the default is RunAs(Caller), which in WebSphere Application Server for z/OS is the user ID that authenticated to WebSphere. You might want to associate a RunAs(Role) with the servlet or EJB in order to switch the current Java thread identity from the authenticated user ID to the user ID associated with the RunAs role.

Note: With WebSphere Studio Application Developer you can set a RunAs role for an EJB, but currently not for a servlet. You can set a RunAs role for both servlet and EJB using the AAT and because you are likely to want to change this as you experiment with the use or non-use of roles, we recommend that you use the AAT.

Using AAT, expand the Web Modules and navigate down to the servlet in the Web Components directory.

Click the servlet, and then click the **Security** tab in the right pane.

Click the drop-down list box and select from the available roles, as shown in Figure 8-6.

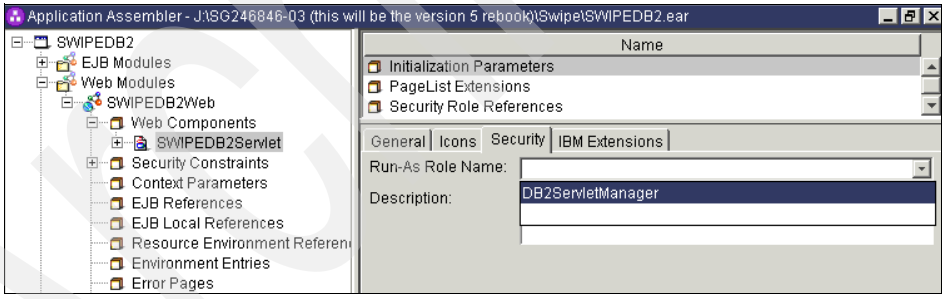


Figure 8-6 Defining RunAs for a servlet using the AAT

The process is the same for the EJB.

8.1.4 Prepare WebSphere security to run the samples

Note: Significant new security function was added in WebSphere Application Server for z/OS V5 Service Level W500103. We used this service level when testing. You should be on at least this service level before attempting to use security in WebSphere Application Server for z/OS V5. The instructions and screen shots in this section are based on W500103.

The deployment descriptors of the SWIPE/EIS samples have no role to user mappings, because with WebSphere Application Server for z/OS, there is the ability to use profiles in the RACF EJBROLE class to perform that mapping.

In order to do this, your WebSphere must have global security enabled and you must be using the Local OS registry. You also need to set some Custom Properties for the Local OS registry.

Important: We did all our EIS security verification with Local OS as a cell-wide configured registry. You need to be aware that if you configure CUR or LDAP as a registry, you will lose the capability to delegate the current thread identities from the J2EE server into the EIS space.

To do this, use the WebSphere administrative console and navigate to **Security** → **User Registries**, and then click **Local OS**. Then scroll down and click **Custom Properties**. Click each property and set them as shown in Figure 8-7.

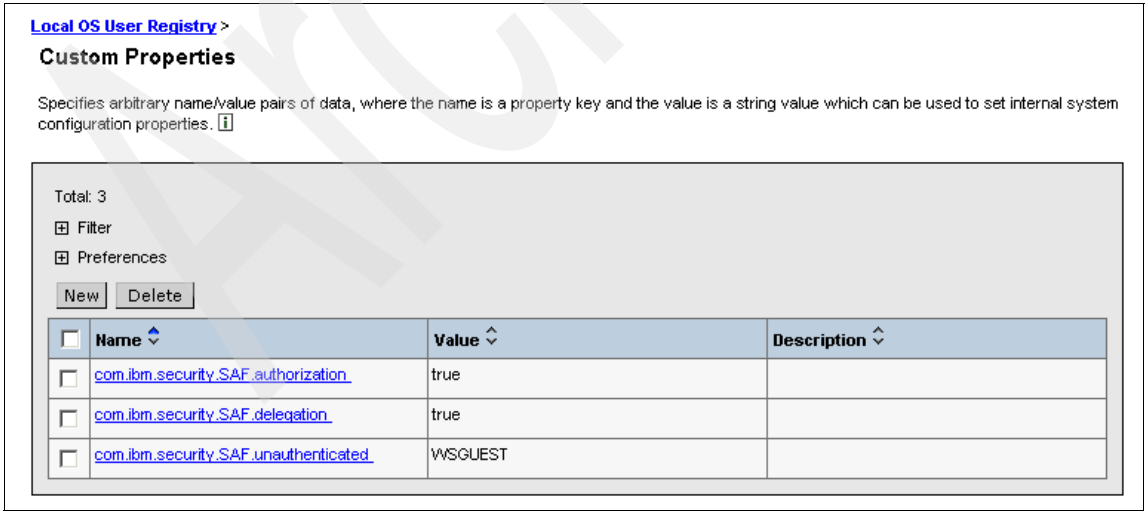


Figure 8-7 Local OS properties for SAF-based authorization and RunAs delegation

Defining RACF SWIPEEIS test users in a group

You will need to plan and create some test users in a RACF group. Do this before creating EJBROLE profiles.

Run some commands similar to those in Example 8-1.

Example 8-1 Creating a group and user IDs for EJB roles and thread identity checking

```
AG TESTUG OWNER(SYS1)SUPGROUP(SYS1) DATA('Group for Test Users')  +
OMVS(GID(20000))

AU SERVMAN DFLTGRP(TESTUG) NAME('CICSServletManager Role Userid')  +
OMVS(UID(20001) PROGRAM('/bin/sh') HOME('/u/servman')) NOPASSWORD

AU EJBMAN DFLTGRP(TESTUG) NAME('CICSEJBManager Role Userid')  +
OMVS(UID(20002) PROGRAM('/bin/sh') HOME('/u/ejbman')) NOPASSWORD

AU TESTU1 DFLTGRP(TESTUG) NAME('WAS ZOS Test User 1') +
PASSWORD(TESTU2) NOEXPIRED +
OMVS(UID(20003) PROGRAM('/bin/sh') HOME('/u/testu1'))
PASSWORD NOINTERVAL USER(TESTU1)

AU TESTU2 DFLTGRP(TESTUG) NAME('WAS ZOS Test User 2') +
PASSWORD(TESTU2) NOEXPIRED +
OMVS(UID(20004) PROGRAM('/bin/sh') HOME('/u/testu1'))
PASSWORD NOINTERVAL USER(TESTU2)

AU USERA DFLTGRP(TESTUG) NAME('WAS ZOS JAAS Auth Alias') +
PASSWORD(USERA) NOEXPIRED +
OMVS(UID(20005) PROGRAM('/bin/sh')HOME ('/u/usera'))
PASSWORD NOINTERVAL USER(USERA)

AU USERB DFLTGRP(TESTUG)NAME('WAS ZOS JAAS Auth Alias') +
PASSWORD(USERB) NOEXPIRED +
  OMVS(UID(20006) PROGRAM('/bin/sh')HOME ('/u/userb'))
  PASSWORD NOINTERVAL USER(USERB)

RDEFINE EJBROLE CICSServletManager UACC(NONE) APPLDATA('SERVMAN')
RDEFINE EJBROLE CICSEJBManager     UACC(NONE) APPLDATA('EJBMAN')
RDEFINE EJBROLE IMSServletManager  UACC(NONE) APPLDATA('SERVMAN')
RDEFINE EJBROLE IMSEJBManager      UACC(NONE) APPLDATA('EJBMAN')
RDEFINE EJBROLE DB2ServletManager  UACC(NONE) APPLDATA('SERVMAN')
RDEFINE EJBROLE DB2EJBManager      UACC(NONE) APPLDATA('EJBMAN')
 PE CICSServletManager CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
 PE CICSEJBManager     CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
 PE IMSServletManager  CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
 PE IMSEJBManager      CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
 PE DB2ServletManager  CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
 PE DB2EJBManager      CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
 SETR RACLIST(EJBROLE)REFRESH
```

The first two user IDs in Example 8-1 on page 202 are to be associated with the EJBROLE profiles we create next. The second two user IDs will be used when running the SWIPE/EIS applications. For example, you could authenticate to WebSphere with TESTU1, and then use TESTU2/password on the SWIPE*eisWeb* index.jsp when running with a resource reference set to Authentication=Application (res-auth=Application). The user ID/password you enter will be set on the ConnectionSpec object and passed on the getCconnection when running Authentication(Application).

The last two user IDs, USERA and USERB, will be associated with JAAS authentication aliases. See 8.2.2, “Set up JAAS authentication aliases” on page 207.

The user IDs shown in Example 8-1 on page 202 are probably the minimum you need to use to see how WebSphere security works. If you want to do more sophisticated testing, you might need more groups and more users so that you can permit different groups to different resource profiles. Our objective with the SWIPE/EIS applications was to simply understand how roles and user ID propagation work so we have all users in one group, TESTUG, and that group is permitted to all roles and all other RACF resource profiles in order for the applications to function.

z/OS RACF EJBROLE configuration

You will need to create some RACF EJB role profiles that will match the role names in the SWIPE/EIS application deployment descriptors.

Example 8-2 RACF configuration for security roles

```
RDEFINE EJBROLE CICSServletManager UACC(NONE) APPLDATA('SERVMAN')
PE CICSServletManager CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
RDEFINE EJBROLE CICSEJBManager UACC(NONE) APPLDATA('EJBMAN')
PE CICSEJBManager CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
RDEFINE EJBROLE IMSServletManager UACC(NONE) APPLDATA('SERVMAN')
PE IMSServletManager CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
RDEFINE EJBROLE IMSEJBManager UACC(NONE) APPLDATA('EJBMAN')
PE IMSEJBManager CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
RDEFINE EJBROLE DB2ServletManager UACC(NONE) APPLDATA('SERVMAN')
PE DB2ServletManager CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)
RDEFINE EJBROLE DB2EJBManager UACC(NONE) APPLDATA('EJBMAN')
PE DB2EJBManager CLASS(EJBROLE) ID(TESTUG) ACCESS(READ)

SETR RACLIST(EJBROLE) REFRESH
```

Permitting to the APPL CBS390 profile

If APPL class is active and you have defined the CBS390 profile in class APPL during your WebSphere configuration, you need to ensure that all the test users have READ access to the CBS390 profile. Permit the test user group as follows:

```
PE CBS390 CLASS(APPL) ID(TESTUG) ACCESS(READ)
```

Permitting to EIS RACF resource profiles

Your EIS might have implemented various forms of authorization checking to control access to its resources. For example, a CICS system might have RACF profiles in class TCICSTRN defined. If you want your test users to run the DPLC sample program provided with SWIPECICS, you must allow the test users to run the CICS mirror transactions (CSMI) or any custom mirror transaction you might create.

In some circumstances, your request might reach the EIS under the RACF user ID of the WebSphere servant region, so you should also permit the WebSphere servant region to any protected resources in your EIS.

8.1.5 Plan resource reference to connection factory bindings

The SWIPE/EIS applications come with deployment descriptors that contain various resource references. You can use WebSphere Studio Application Developer or the AAT to create resource references in the EJB deployment descriptor that bind to different connection factories that you create using the WebSphere administrative console.

You will find that you might need a lot of different connection factories, each with different Custom Properties. Then for each different connection factory, you might want to see the effect of running with a resource reference that sets authentication (container) and another that sets authentication (application).

The connection factories we tested with are not identical for all SWIPE/EIS applications, but the following list should give you some idea of the things you can try:

- Local mode versus remote mode
- JAAS authentication alias defined
- Invalid user ID set for the JAAS authentication alias
- Invalid password set for the JAAS authentication alias
- SSL used for the connection to the EIS

Then for each of these you could bind a resource reference with Authentication=Application and one with Authentication=Container, making 10

combinations in all. And for JDBC providers you might want to test with ThreadSecurity (Connection Manager RunAs Identity Enabled) enabled and disabled.

The thread identity and thread security concepts

Briefly, the thread identity and thread security (also known as SyncToOSThread) are two options exclusive to IBM WebSphere Application Server for z/OS available for J2CA connectors.

Thread identity support is used to flow the current thread identity to the back-end EIS. Most z/OS customers will want to use this feature because it enables WebSphere Application Server for z/OS to behave in a way that traditional z/OS address spaces behave, that is, after you have authenticated, your user ID flows with any work you do within the z/OS system cutting SMF audit trail records as it goes.

Thread security can be used to “push down” the current thread identity onto the OS thread. This feature is required for the IMS and DB2 JDBC providers in order to flow the current thread identity to IMS and DB2. The IMS and CICS J2CA connectors do not need thread security in order to propagate the current thread identity through to IMS and CICS.

In order to use thread identity and thread security the J2CA connection must be container-managed. That means that your resource references in the EJB deployment descriptor must set Authentication=Container (<res-auth>Container</res-auth> in ejb.xml) for these functions to work.

Figure 8-8 on page 206 shows a display from WebSphere Studio Application Developer where the Authentication attribute is set. There is a resource reference called /eis/db2contJCA1 which has a JNDI name of /eis/db2sampleJCA1. The *cont* in the resource reference name is used when we set the Authentication to Container. In this way you can avoid confusion. In the WebSphere administrative console we created a data source for the DB2 JDBC provider called db2sampleJCA1, which had a JNDI name of /eis/db2sampleJCA1 so when we deploy, the JNDI names already match.

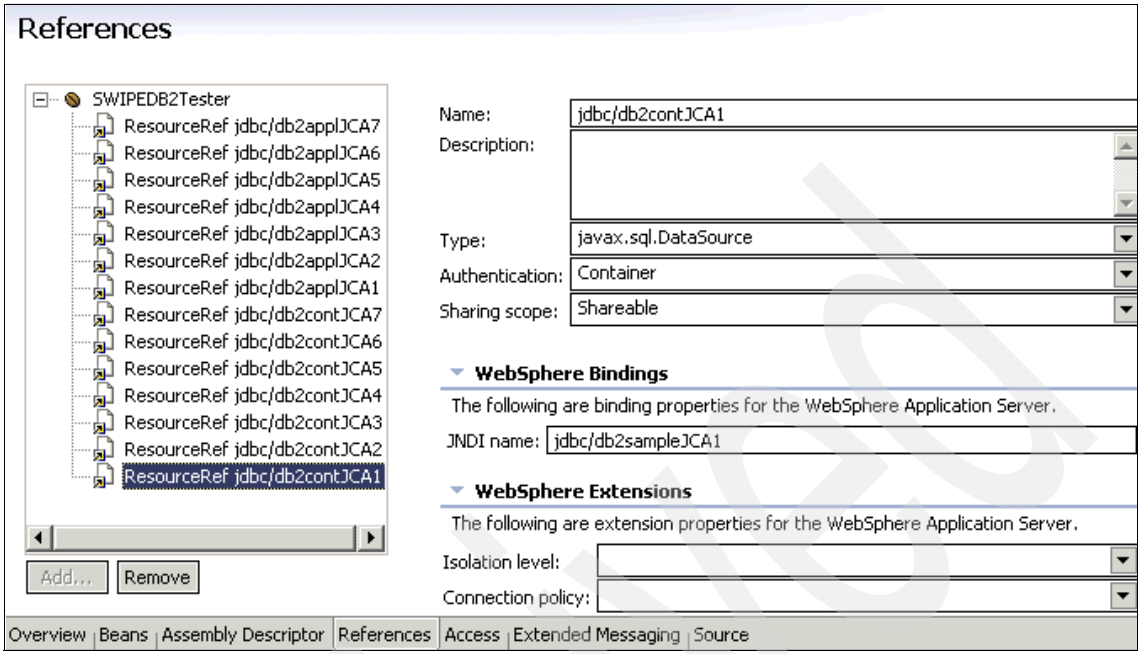


Figure 8-8 J2EE resource reference to data source, Authentication=Container

The J2CA connector *declares* to the WebSphere container whether it supports or allows thread identity, or supports or requires thread security, so you do not have to set anything in the deployment descriptor to enable these as long as you are using Authentication=Container. However, thread security is only enabled if your WebSphere cell has global security enabled.

If you want to propagate the current thread identity to IMS and DB2 using the IMS and DB2 JDBC providers, you need to use the WebSphere administrative console. Navigate to **Security** → **Global Security**, scroll down and select **z/OS Security Options**, and then select the **Connection Manager RunAs Identity Enabled** option.

8.2 Define J2CA connection factories and data sources

The J2CA connection factories and data sources must be defined before you install the SWIPE/EIS J2EE applications. We recommend that you define at least one J2CA connection factory or data source for each EIS being used after you decide what scenario is suitable for your installation. If you are not sure how to define J2CA connection factories and data sources, see *WebSphere for z/OS V5*

Connectivity Handbook, SG24-7064. Then look at the way we did it in the following sections.

8.2.1 Suggested scenarios for security verification

In our tests we defined J2CA connection factories and data sources to allow us to test many different scenarios. Numerous scenarios were needed because the user ID that is flowed to the EIS depends on many variables:

- Local mode versus remote mode
- JAAS authentication alias defined
- Thread identity and thread security enabled or not

Additionally, we performed tests using an invalid user and password in the JAAS alias entry:

- Invalid user ID set for the JAAS authentication alias
- Invalid password set for the JAAS authentication alias

You might also want to check other aspects, such as SSL:

- SSL used for the connection to a remote EIS.

8.2.2 Set up JAAS authentication aliases

If you want to test the effect of using a JAAS authentication alias, you need to create one or more aliases *before* you create the connection factories because you need to refer to the JAAS authentication alias from the connection factory.

Therefore, it makes sense to prepare some different JAAS authentication aliases first. Table 8-2 shows the JAAS authentication aliases we created. You must have created user IDs USERA and USERB earlier (“Defining RACF SWIPEEIS test users in a group” on page 202).

Use the WebSphere administrative console and navigate to **Security → JAAS Configuration → J2C Authentication Data**. Then click **New** and complete the next panel. Note that JAAS authentication alias and J2C (Java 2 Connector) Coauthentication data entry are different terms for the same thing.

Table 8-2 JAAS alias entries for SWIPE CICS, IMS, and DB2

JAAS authentication alias name	User ID	Password
CICS	USERA	
DB2	USERA	USERA (OK)

JAAS authentication alias name	User ID	Password
IMS	USERB	USERB (OK)
WrongUser	WRONGUSR	WRONGPSW
WrongPsw	USERA	WRONGPSW

8.2.3 Set up connection factories and data sources for SWIPE/EIS

The testing you perform for each EIS will probably vary a little. We set up connection factories and data sources to perform similar testing of each J2CA connector but there are some differences in the setup for each EIS.

SWIPE for CICS

The SWIPE for CICS application uses the CICS ECI Resource Adapter, which is supplied as part of the CICS Transaction Gateway for z/OS V5.01.

For instructions on how to install the CICS ECI Resource Adapter and how to configure the J2CA connection factories, see *WebSphere for z/OS V5 Connectivity Handbook,* SG24-7064.

Connection factories were created, as shown in Table 8-5 on page 210. SWIPECICS used the JAAS authentication aliases, CICS, WRONGUSR, and WRONGPSW, as shown in Table 8-2 on page 207.

Table 8-3 SWIPE CICS connection factories

Connection factory name	JNDI name	JAAS authentication alias?
cicslocalJCA1	eis/cicslocalJCA1	----------
cicslocalJCA2	eis/cicslocalJCA2	CICS
cicslocalJCA3	eis/cicslocalJCA3	WrongUser
cicslocalJCA4	eis/cicslocalJCA4	WrongPsw
cicsremoteJCA1	eis/cicsremoteJCA1	----------
cicsremoteJCA2	eis/cicsremoteJCA2	CICS
cicsremoteJCA3	eis/cicsremoteJCA3	WrongUser
cicsremoteJCA4	eis/cicsremoteJCA4	WrongPsw
cicsremoteJCASSL	eis/cicsremoteJCASSL	----------

Then in the EJB deployment descriptor of SWIPECICS we created resource references, which mapped to the connection factories.

The SWIPE CICS resource reference bindings are shown in Table 8-4.

Table 8-4 SWIPE CICS WebSphere bindings

EJB deployment descriptor resource reference name	Authentication	JNDI name
eis/cicslocalcontJCA1	Container	eis/cicslocalJCA1
eis/cicslocalcontJCA2	Container	eis/cicslocalJCA2
eis/cicslocalcontJCA3	Container	eis/cicslocalJCA3
eis/cicslocalcontJCA4	Container	eis/cicslocalJCA4
eis/cicslocalapplJCA1	Application	eis/cicslocalJCA1
eis/cicslocalapplJCA2	Application	eis/cicslocalJCA2
eis/cicslocalapplJCA3	Application	eis/cicslocalJCA3
eis/cicslocalapplJCA4	Application	eis/cicslocalJCA4
eis/cicsremotecontJCA1	Container	eis/cicsremoteJCA1
eis/cicsremotecontJCA2	Container	eis/cicsremoteJCA2
eis/cicsremotecontJCA3	Container	eis/cicsremoteJCA3
eis/cicsremotecontJCA4	Container	eis/cicsremoteJCA4
eis/cicsremoteapplJCA1	Application	eis/cicsremoteJCA1
eis/cicsremoteapplJCA2	Application	eis/cicsremoteJCA2
eis/cicsremoteapplJCA3	Application	eis/cicsremoteJCA3
eis/cicsremoteapplJCA4	Application	eis/cicsremoteJCA4
eis/cicsremotecontJCASSL	Container	eis/cicsremoteJCASSL

SWIPE for IMS

The SWIPE for IMS application uses the IMS Connector for Java. For instructions on how to install this Resource Adapter for IMS and how to configure the J2CA connection factories, see *WebSphere for z/OS V5 Connectivity Handbook,* SG24-7064.

Connection factories were created as shown in Table 8-5 on page 210. SWIPE IMS used the JAAS authentication aliases, IMS, WRONGUSR, and WRONGPSW, as shown in Table 8-2 on page 207.

Table 8-5 SWIPE IMS J2CA connection factories

Connection factory name	JNDI	JAAS auth alias (component)	JAAS auth alias (container)
imselocal1_cf	eis/imssampleJCA1	----	---
imselocal2_cf	eis/imssampleJCA2	IMS	IMS
imselocal3_cf	eis/imssampleJCA3	IMS	---
imselocal4_cf	eis/imssampleJCA4	WrongUser	WrongUser
imselocal5_cf	eis/imssampleJCA5	WrongPsw	WrongPsw
imselocal6_cf	eis/imssampleJCA6	----	IMS
imsermt1_cf	eis/imssampleJCA1	----	---
imsermtl2_cf	eis/imssampleJCA2	IMS	IMS
imsermtl3_cf	eis/imssampleJCA3	IMS	---
imsermt4_cf	eis/imssampleJCA4	WrongUser	WrongUser
imsermtl5_cf	eis/imssampleJCA5	WrongPsw	WrongPsw
imsermtl6_cf	eis/imssampleJCA6	----	IMS

Table 8-6 shows how we configured the WebSphere bindings in the EJB deployment descriptor to associate the resource references with real JNDI names.

Table 8-6 SWIPE IMS WebSphere bindings

EJB deployment descriptor resource reference name	Authentication	JNDI name
eis/imscontJCA1	Container	eis/imssampleJCA1
eis/imscontJCA2	Container	eis/imssampleJCA2
eis/imscontJCA3	Container	eis/imssampleJCA3
eis/imscontJCA4	Container	eis/imssampleJCA4
eis/imscontJCA5	Container	eis/imssampleJCA5
eis/imscontJCA6	Container	eis/imssampleJCA6
eis/imsapplJCA1	Application	eis/imssampleJCA1

EJB deployment descriptor resource reference name	Authentication	JNDI name
eis/imsapplJCA2	Application	eis/imssampleJCA2
eis/imsapplJCA3	Application	eis/imssampleJCA3
eis/imsapplJCA4	Application	eis/imssampleJCA4
eis/imsapplJCA5	Application	eis/imssampleJCA5
eis/imsapplJCA6	Application	eis/imssampleJCA6

SWIPE for DB2

The SWIPE for DB2 application uses the DB2 390 Local JDBC Provider (RRS) already installed into WebSphere. For more details about how to configure the DB2 JDBC provider and data sources, see *WebSphere for z/OS V5 Connectivity Handbook,* SG24-7064.

Restriction: SWIPE for DB2 is a J2EE 1.3-compliant application. The only option supported for J2EE 1.3 is the data source V5 style (J2CA connector). data source V4 style does not work for J2EE 1.3 applications (EJB 2.0 and Servlet 2.3 specifications).

Table 8-7 and Table 8-8 on page 212 were used by our SWIPE DB2 tests. SWIPE DB2 used the JAAS authentication aliases, DB2, IMS, WRONGUSR, and WRONGPSW, as shown in Table 8-2 on page 207.

Table 8-7 Data sources entries for SWIPE DB2

Data source	JNDI	JAAS auth alias (component)	JAAS auth alias (container)
db2sampleJCA1	jdbc/db2sampleJCA2	----	-----
db2sampleJCA2	jdbc/db2sampleJCA1	DB2	DB2
db2sampleJCA3	jdbc/db2sampleJCA3	DB2	---
db2sampleJCA4	jdbc/db2sampleJCA4	IMS	---
db2sampleJCA5	jdbc/db2sampleJCA5	WrongUser	WrongUser
db2sampleJCA6	jdbc/db2sampleJCA6	WrongPsw	WrongPsw
db2sampleJCA7	jdbc/db2sampleJCA7	---	DB2

Table 8-8 shows how we configured the WebSphere bindings in the EJB deployment descriptor to associate the resource references with real JNDI names.

Table 8-8 DB2 WebSphere bindings

J2EE res. ref.	Authentication	JNDI name
jdbc/db2contJCA1	Container	jdbc/db2sampleJCA1
jdbc/db2contJCA2	Container	jdbc/db2sampleJCA2
jdbc/db2contJCA3	Container	jdbc/db2sampleJCA3
jdbc/db2contJCA4	Container	jdbc/db2sampleJCA4
jdbc/db2contJCA5	Container	jdbc/db2sampleJCA5
jdbc/db2contJCA6	Container	jdbc/db2sampleJCA6
jdbc/db2contJCA7	Container	jdbc/db2sampleJCA7
jdbc/db2applJCA1	Application	jdbc/db2sampleJCA1
jdbc/db2applJCA2	Application	jdbc/db2sampleJCA2
jdbc/db2applJCA3	Application	jdbc/db2sampleJCA3
jdbc/db2applJCA4	Application	jdbc/db2sampleJCA4
jdbc/db2applJCA5	Application	jdbc/db2sampleJCA5
jdbc/db2applJCA6	Application	jdbc/db2sampleJCA6
jdbc/db2applJCA7	Application	jdbc/db2sampleJCA7

8.3 Install SWIPE for CICS, IMS, and DB2

The SWIPE/EIS applications use data source references for DB2 and J2CA Connection references for CICS and IMS. During installation of the .ear files you must associate these references with the JNDI names defined in data source (V5 style) or J2CA connection factory.

Before you install the SWIPE/EIS applications you must define at least one of these resources, but you should probably consider setting up a series of scenarios as we described in 8.2.3, “Set up connection factories and data sources for SWIPE/EIS” on page 208.

The WebSphere administrative console or wsadmin script tool can be used to install the applications. We used the administrative console and performed the following tasks:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications** → **Install New Application**.

   A dialog for installing the application opens.

2. Type the path to the SWIPEeis.ear file or use the *Browse* button to find it in your local path. See Figure 8-9. In the following instructions we will use the swipedb2.ear as an example, but the installation process is very similar for the SWIPECICS.ear and SWIPEIMS.ear, except in step 4 (Map resource references to resources) where each EIS has a different subset of J2EE resource references. If you are using the same naming convention and scenarios that we used, you could use the tables shown in 8.2.1, “Suggested scenarios for security verification” on page 207 to verify that the resource reference to resource mapping is correct.

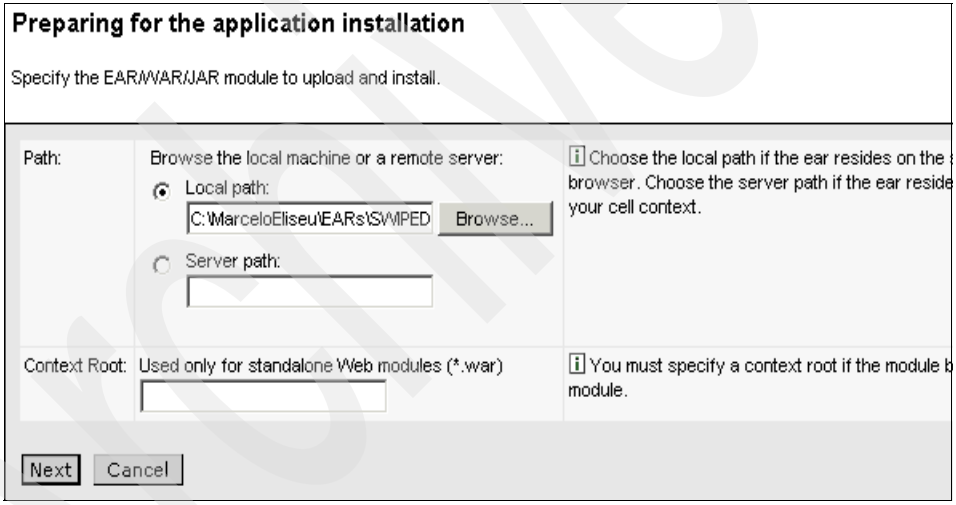


Figure 8-9 SWIPEeis.ear install application

3. Click **Next**.

   A dialog for generating default bindings opens.

4. Accept the defaults and click **Next**.

   “Step 1: Provide options to perform the installation” wizard opens.

5. Click **Next**.

   “Step 2: Provide JNDI Names for Beans” wizard opens.

6. Click **Next**.

   “Step 3: Map EJB references to beans” wizard opens.

7. Click **Next**.

   “Step 4: Map resource references to resources” opens. At this point you have to associate the resource references in the EJB deployment descriptor with the JNDI names defined in your J2CA connection factory or data source (V5 style). If you chose the names carefully, the JNDI names should match and you will not need to do anything here.

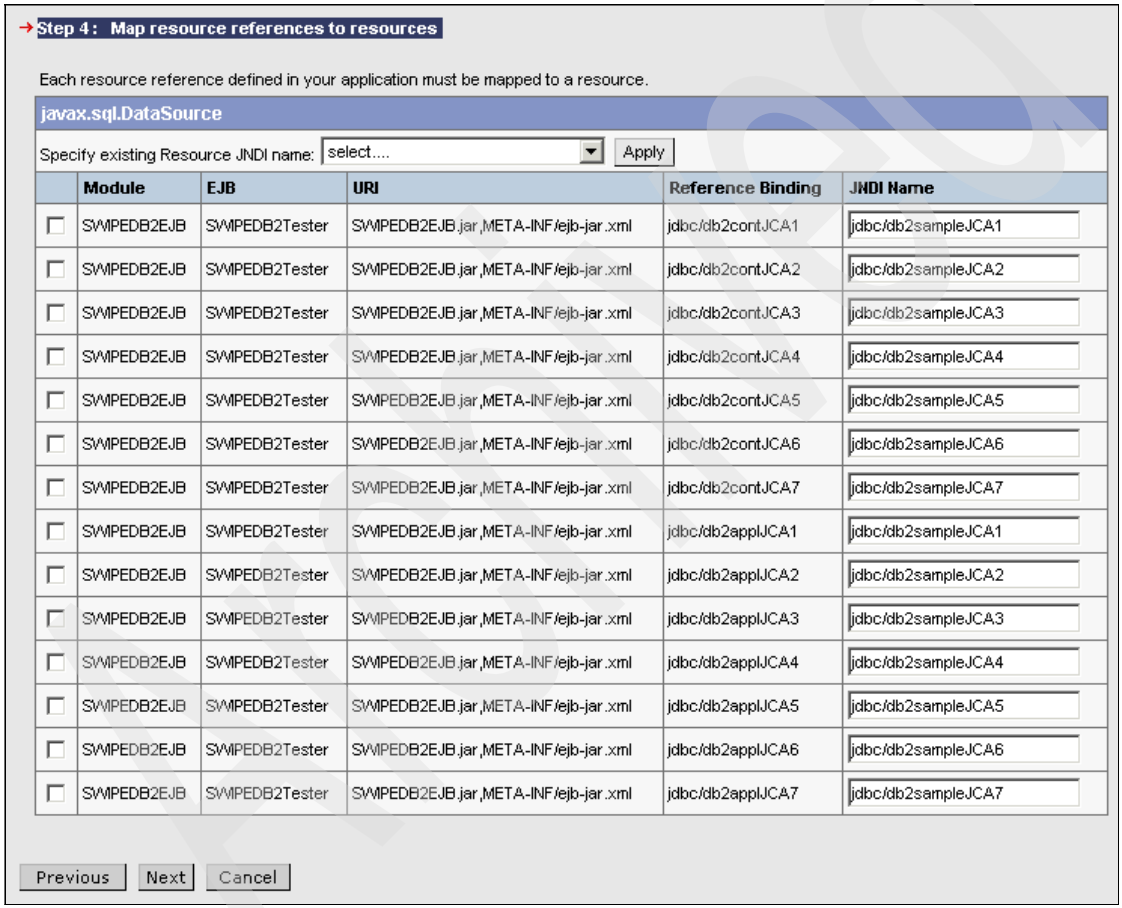


Figure 8-10 Map resource reference to JNDI name during application install

When the application is installed, type the resource reference name you want to use for the test on the index.jsp panel. The SWIPE applications already contain resource references with the JNDI names shown in WebSphere

bindings, Table 8-4 on page 209, Table 8-6 on page 210 and Table 8-8 on page 212.

Tip: There is a trick in this panel if you want to use it to correct the mapping of resource reference to JNDI resource name. First, select the check box to the left of the Module you want to correct, and then click the drop-down list box that contains the text “select...”. Select the JNDI name you need to use, and then click **Apply** to the right of the “select...” box. You should now see the correct JNDI name in the right column, “JNDI Name”.

8. Click **Next**.

   “Step 5: Map virtual hosts for web modules” opens.

9. Click **Next**.

   “Step 6: Map modules to application servers” opens.

10. Click **Next**.

    “Step 7: Map security roles to users/groups” opens. Select the check box for **All authenticated?** and click **Next**.

Note: When using RACF to perform EJBROLE authorization and role-to-user ID mapping, anything you set in Step 7 will be ignored.

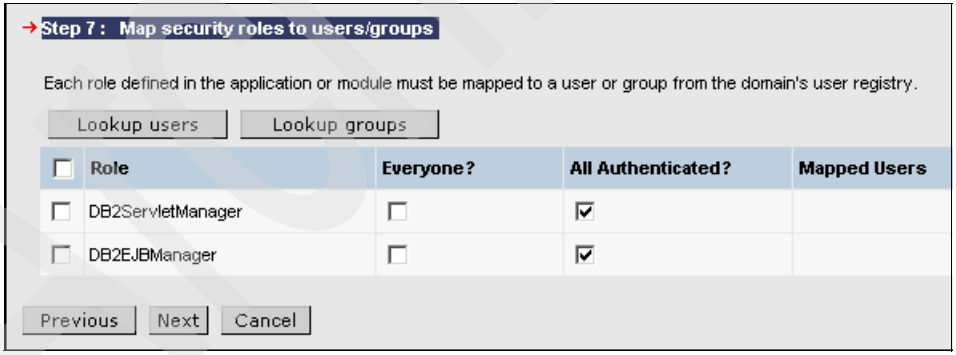


Figure 8-11 Map security roles to users/groups step during application install

11. Click **Next**.

    “Step 8: Summary” opens. Click **Finish** and **Save to Master Configuration** on the next panel.

Tip: If you use these applications to test many different scenarios, you could be redeploying them a number of times, and after the resource reference to resource mapping is correct (Step 4) you will probably just need to use **Next** for each step. Note that after you have clicked **Next** in Step1, you can scroll down and jump right to the last step and click **Next** → **Finish**.

8.4 Install the CICS components of SWIPECICS

The SWIPECICS J2EE application is designed to call a program called DPLC in CICS. The program source is supplied in the SWIPECICS.zip file together with a README.txt file that explains how to compile the CICS components and install them into your CICS system. There are also some programs for driving DPLC from a CICS 3270 terminal.

When you have compiled and installed these components in CICS you should be able to test the function by logging onto to CICS from a 3270 terminal and issuing the DPLC transaction. If this returns the screen shown in Figure 8-12then you have correctly installed the CICS components.

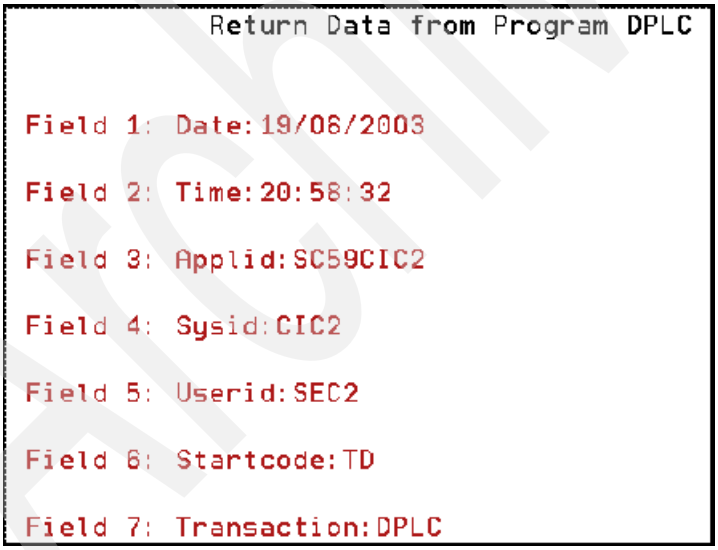


Figure 8-12 Example of a working 3270 DPLC transaction in CICS

Check the RACF profiles protecting CICS resources

Before using the SWIPECICS J2EE application to drive the DPLC program, review the RACF profiles protecting CICS resources to make sure that the test user IDs you defined in Example 8-1 on page 202 have access. See *WebSphere*

for z/OS V5 Connectivity Handbook, SG24-7064 for information about the RACF profiles controlling access to CICS.

8.5 Start SWIPE for CICS, IMS, and DB2

After installation, start the application in WebSphere.

Using the administrative console, navigate to **Applications** → **Enterprise Applications**, select the check box left to SWIPE*eis* (the status should show "stopped," indicated by the red x figure), and click **Start**. Then wait for a screen full of messages, among which is the message shown in Figure 8-13

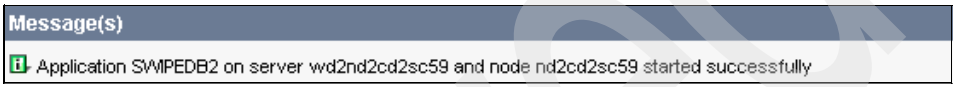


Figure 8-13 Message started successfully for application install

Tip: After redeploying applications, if you are using a WebSphere base cluster configuration (as opposed to Network Deployment), you might have to stop and start the WebSphere server. Also, if you change EJBROLE profiles, these are cached in the WebSphere server, so even if you refresh them with SETR RACLIST(EJBROLE) REFRESH, WebSphere might use the earlier versions.

When redeploying you can use the Update option rather than Stop, UnInstall, and then Install and Start. In general, Update seems to work very well if the changes are only to the deployment descriptor, but if you change JSPs you might find you have to Uninstall the Install to pick up the new JSP.

8.6 Run SWIPE for CICS, IMS, and DB2

Use one of the following URLs to run the SWIPE application:

```
http://websphere_host:port/SWIPEDB2Web/
http://websphere_host:port/SWIPECICSWeb
http://websphere_host:port/SWIPEIMSWeb
```

Where `websphere_host` is the host name where you installed the SWIPE application, and `port` is the port number of the WebSphere transport handler or Web server.

You should see the index.jsp form as shown in the next figures.

SWIPEDB2 index.jsp form

Figure 8-14 shows the SWIPEDB2 index.jsp form.

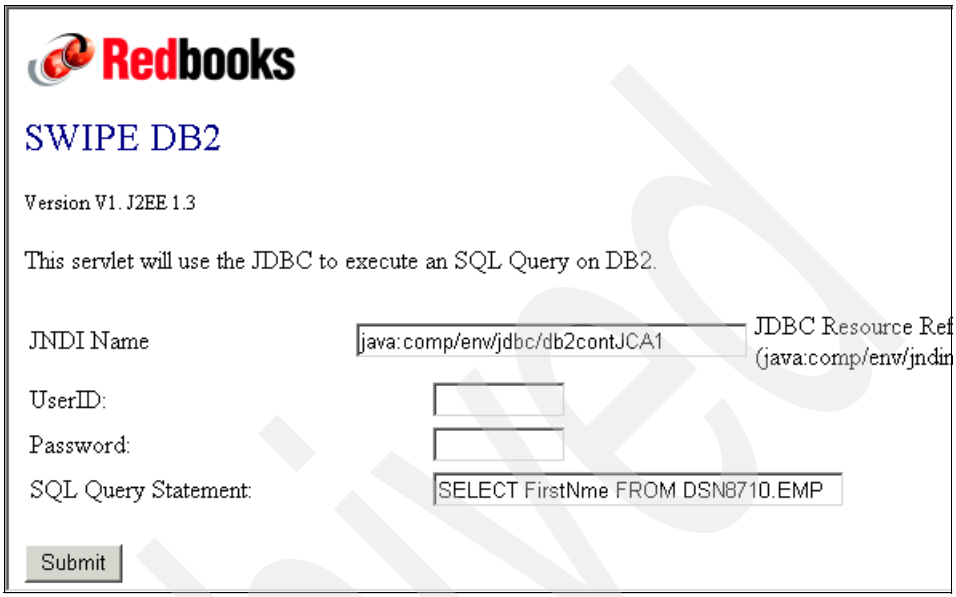


Figure 8-14 SWIPE for DB2 input form

You need to select the resource reference JNDI name to select the connection factory you want to test.

If you set UserId and Password on the screen, these will be passed to DB2 when you use a resource reference that specifies Authentication=Application.

You can code any SQL statement you want in the SQL Query Statement box but the user ID you pass to DB2 must be authorized to the DSNJDBC plan and to the tables you want to access.

SWIPECICS index.jsp form

Figure 8-15 on page 219 shows the SWIPECICS index.jsp form.

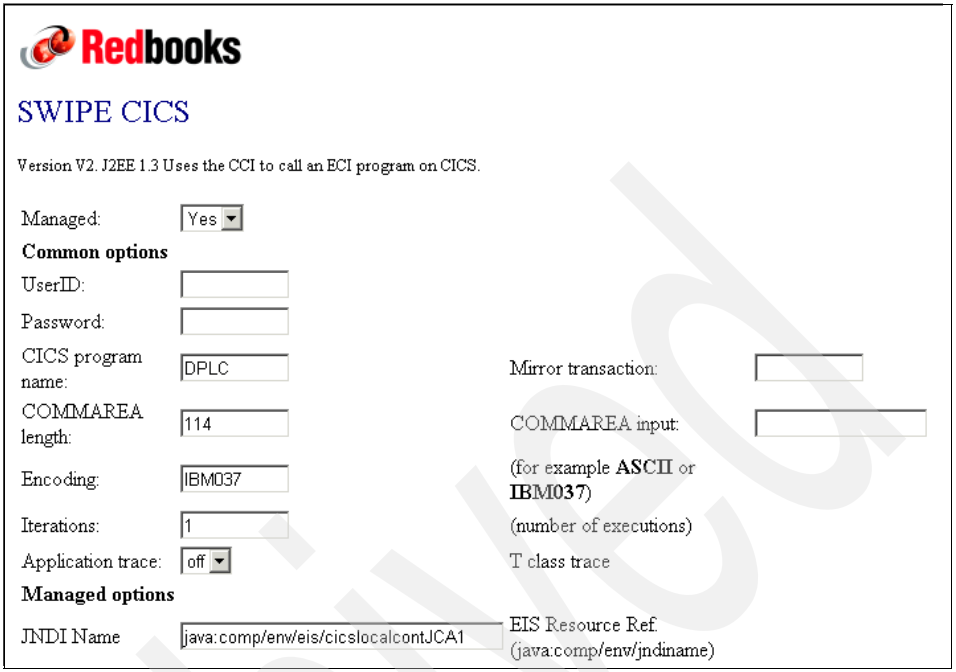


Figure 8-15 Top part of the SWIPECICS input form

SWIPECICS is an enhanced version of the sample program CTGTesterECI, which was originally provided and documented in *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133. Consult that redbook for detailed information about the structure and use of the application. Here we briefly describe the index/jsp input fields.

The top part of the SWIPECICS panel shows the options for a WebSphere Managed connection, while the bottom part in Figure 8-16 on page 221 shows the options for a Non-Managed connection and the Submit button.

The first drop-down list at the top of the form is where you can select Managed or Non-Managed.

Next there are fields common to both Managed and Non-Managed connections.

If you enter a user ID and password, these will be set on the ConnectionSpec object and passed on the getConnection. Whether the user ID you enter is used depends on whether you have a JAAS authentication alias set on the resource reference (JNDI name) and whether the resource reference uses Authentication=Application or Container.

By default, SWIPECICS executes a simple C language program in CICS called DPLC. This expects to return a COMMAREA, which is a string of 114 bytes. You do not need to supply any data in the COMMAREA input field, but if you set some input data in the COMMEARA you can easily check that ASCII/EBCDIC translation is working properly by looking at the CICS Transaction Gateway trace entries or by running the CICS debug transaction, CEDX, which allows you to see the COMMAREA data passed to CICS.

On the right is a field where you can specify the transaction code of the mirror transaction you want to use in CICS. The value you specify will be passed as TPName on the InteractionSpec object. To use a transaction other than the default of CSMI, you would need to use the CICS CEDA transaction to copy the supplied CSMI transaction definition to create a new transaction (MIRR for example), and then install the MIRR transaction in CICS.

The SWIPECICS application performs ASCII/EBCDIC conversion itself, so you should leave the Encoding field to IBM037. If CICS were performing ASCII/EBCDIC conversion, the DPLC program would need to be defined in the CICS DFHCNV table with macros to state how the different fields of the COMMAREA should be translated. When CICS performs translation you could set ASCII for the Encoding option.

You can use the Iterations field to issue multiple requests and do some limited performance testing and testing of Connection Pooling.

If you set the Application Trace to On, you will get the CICS ECI Resource Adapter T trace in the WebSphere servant JOBLOG. This is very useful for diagnosis.

When using a managed connection, you set the name of the resource reference for the connection factory you want to use in the JNDI Name field. This currently defaults to java:comp/env/eis/cicslocalcontJCA1, which was designed to map to the connection factory called cicslocalJCA1.

If you select a Non-Managed connection, the JNDI Name field is ignored (because by definition a Managed connection is one where you look up and get a connection from a connection factory.

When you use a Non-Managed connection, SWIPECICS builds the connection itself. You must specify something for the ConnectionURL and the CICS server name (which is the VTAM® APPLID of the CICS region to which you want to connect).

If ConnectionURL is set to local, you need to set the CICS Server name; port is ignored.

If you want to connect to a remote CICS Transaction Gateway daemon, set a URL with a network protocol such as TCP, SSL, HTTP, or HTTPS and the domain name or IP address of the CICS Transaction Gateway daemon. For example: tcp://wtsc59.itso.ibm.com.

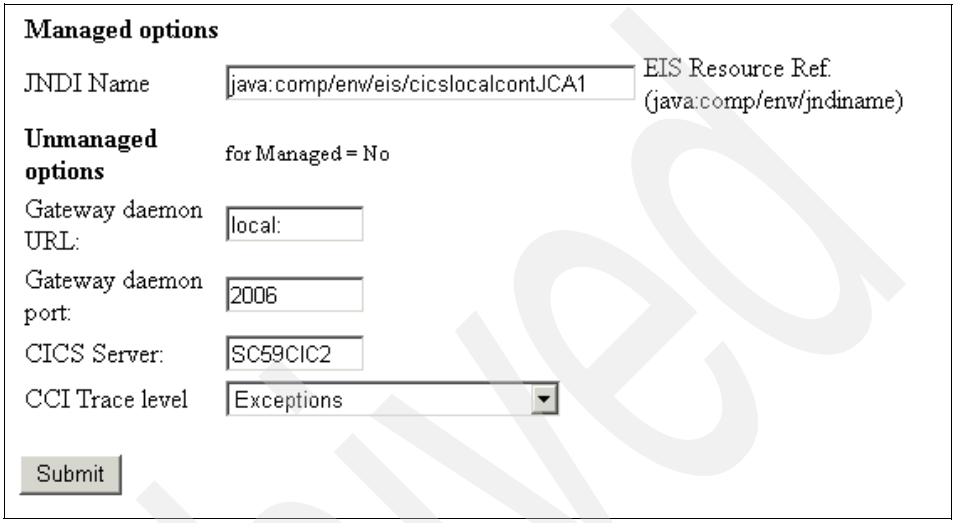


Figure 8-16 Bottom part of the SWIPECICS input form

After submitting, you should get a COMMAREA back from CICS that tells you the user ID that CICS used to execute your request, and the transaction code of the mirror transaction that services the request. For example, see Figure 8-17 on page 222.

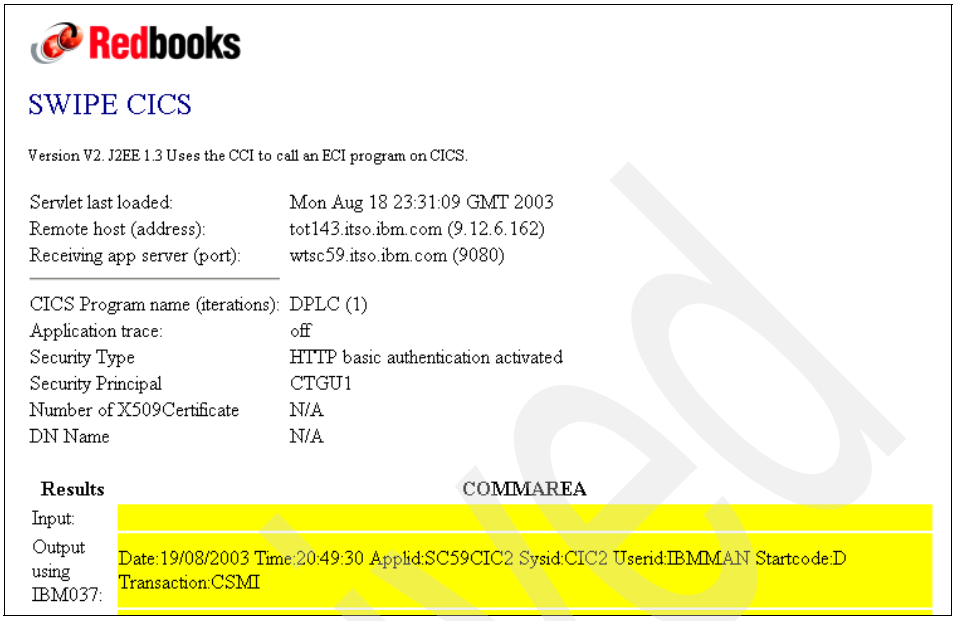


Figure 8-17 Example of the results.jsp for a request to CICS

SWIPE IMS index.jsp form

Figure 8-18 on page 223 shows the SWIPE IMS index.jsp form.

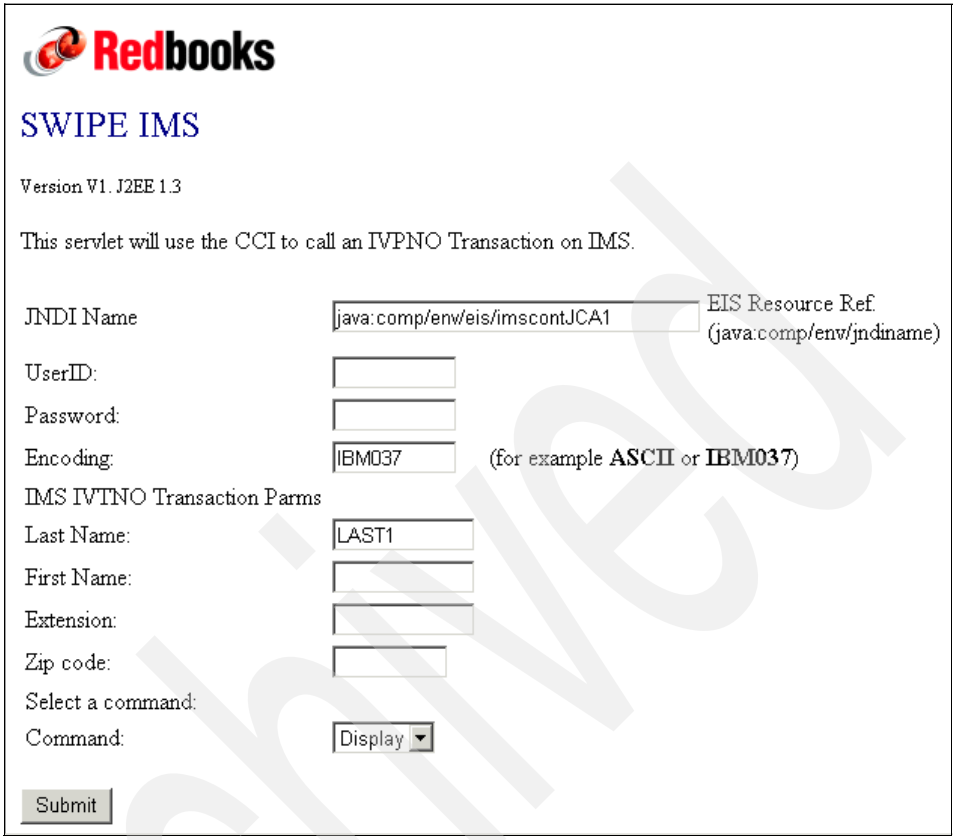


Figure 8-18 SWIPE for IMS input form

As with SWIPEDB2, with SWIPEIMS you choose the JNDI resource reference name in the first box.

You can set a user ID and password and these will be set on the ConnectionSpec object and passed to IMS on the getConnection. They will be used when the resource reference you use specifies Authentication=Application. If you use a resource reference with Authentication=Container, the user ID and password you enter on the panel should be ignored.

The Encoding box should stay at IBM037 because we do ASCII/EBCDIC translation in SWIPEIMS.

The other boxes are input that you can pass to the IMS IVTNO program.

8.7 Debug SWIPE for CICS, IMS, and DB2

The SWIPE/EIS applications report any errors on the error.jsp page.

If the error is a Java naming exception, you probably have an error in the mapping of resource references to J2CA connection factory JNDI names. A common user error is making a typing error in the JNDI Name input field on the input form, so check this first.

Look in the job log of the WebSphere servant and on SYSLOG for any error messages relating to the errors. There could be a problem with access to resources protected by RACF, so look for ICH408I messages.

Check the JOBLOG for the EIS system you are connecting to.

If you make a change (to the deployment descriptor or RACF profiles, for example) that you think should alter the behavior and you find that the behavior has not changed, try uninstalling and installing the application and stop/starting the WebSphere server, especially in a base cluster configuration, to make sure you are not using old cached copies of RACF profiles or earlier versions of JSPs.

If you cannot find the problem by checking your input, the deployment descriptor or the RACF profiles, you will probably need a trace.

Enable J2CA tracing as follows:

1. Using the WebSphere administrative console, navigate to Resources, click your Resource Adapter, scroll down, and click **J2C Connection Factories**.
2. Locate the connection factory you are using and click **Custom Properties**.
3. Set the TraceLevel to 3, click **OK**, and save the change.
4. Now enable J2CA trace recording in WebSphere. You can set this in the jvm.properties file of your servant, but this will require a restart of the server, so it is better to use the MVS MODIFY command, as follows:

   F <serverCR>,tracetosysprint=yes
   F <serverCR>,tracejava='com.ibm.ejs.j2c.*=all=enabled'
   F <serverCR>,tracejava='com.ibm.connector2.***eis***.*=all=enabled'
   F <serverCR>,tracejava='org.apache.wsif.*=all=enabled'

   Substitute ***eis*** for ims, cics, or db2. Pay attention to the quotes in these commands.

See the *WebSphere Application Server for z/OS Information Center* for more details about using the MODIFY command with WebSphere Application Server for z/OS.

If you suspect a problem in CICS, you can log on to CICS from a 3270 terminal and use the CEDX transaction to get an online debug of the mirror transaction (CEDX CSMI to start the debug).

8.8 The SWIPE application for JMS

The SWIPE application to demonstrate security with regard to JMS was developed some time after the CICS, IMS, and JDBC samples, and as such is described separately.

However, the basic approach and concepts are similar to those described for the other samples.

8.8.1 Invoke the JMS sample

The JMS sample can be invoked with or without authentication. Authentication is done through basic authentication only in the example.

8.8.2 SWIPE application for JMS contents

This sample J2EE application is packaged in a file called SWIP*EJMS*.ear, which contains:

- itsoSWIP*EJms*EJB.jar: Contains the session EJB class.
- itsoSWIP*EJms*Web.war: Contains the servlet class, configured to use basic authentication.
- itsoBean.jar: Contains common classes used by both the servlet and EJB.

The SWIPE application for JMS has a main servlet that can be invoked with or without basic authentication. The purpose of these samples is not to test every kind of authentication method. That is covered by the main SWIPE application. The aim here is to investigate the behavior of JMS and security settings.

URL

The URL to invoke the SWIPE JMS application without authentication is:

```
http://<domainName:port>/itsoSwipeJmsWeb/JmsServlet
```

The URL to invoke the SWIPE JMS application with authentication is:

```
http://<domainName:port>/itsoSwipeJmsWeb/secure/JmsServlet
```

8.8.3 Security roles in the samples

The SWIPE JMS sample application uses one EJBROLE called Worker, which is also used in the SWIPE application described in Chapter 7, “The security investigation application” on page 121.

To be able to run the servlet when using the URL that invokes authentication, the user ID entered must be authorized to use the EJBROLE Worker.

Methods in the EJB in the SWIPE JMS application are not protected by any EJBROLE.

8.8.4 WebSphere MQ

The J2EE specification describes the JMS Java APIs, which are the APIs to be used by applications that want to perform messaging type functions. The J2EE 1.3 specification states that an application server to be compliant with the J2EE 1.3 specification must provide the underlying functionality to support the JMS APIs.

WebSphere ships with the IBM WebSphere MQ product to provide the functionality that is required for WebSphere to be compliant with the J2EE 1.3 specification. If you have the full IBM WebSphere MQ product, you can configure WebSphere to use that rather than the embedded JMS provider.

We tested with the full WebSphere MQ product, V5.3.1. Additionally, we had a test fix for APAR PQ73210 applied.

8.8.5 Prepare WebSphere security to run the samples

You need to have WebSphere V5 at maintenance level W501000 or better to get correct behavior of JMS security. This was the level we tested with for the SWIPE JMS application.

The deployment descriptor for the SWIPE/JMS sample has no role in user mapping. We tested only with WebSphere configured to use the Local OS registry, and thus defined the Worker EJBROLE to RACF.

8.8.6 WebSphere MQ: Queue definitions

For our testing we used the queue SYSTEM.DEFAULT.LOCAL.QUEUE as the target queue we wanted to be able to do PUT and GET operations on. This queue is generally set up when a queue manager is set up.

We also defined a queue named ITSO.SWIPE.NOACCESS.QUEUE in the queue manager.

8.8.7 WebSphere MQ: RACF resource profiles

To investigate the security behavior when applications in WebSphere V5 use JMS to access WebSphere MQ resources, you must enable security checking for your WebSphere MQ queue manager.

The name of the queue manager on our system was MQ59. To allow any application in WebSphere to be able to connect to the WebSphere MQ queue manger, we set up, in the MQCONN class, a rule named MQ59.BATCH with UACC set to READ.

We created a rule in the MQQUEUE class called MQ59.ITSO.SWIPE.NOACCESS.QUEUE, with UACC set to NONE. We did not specify any user IDs that had access to this queue.

8.8.8 J2C authentication data entries

Prior to defining resources in WebSphere, we need to set up two J2C authentication data entries, because these definitions will be used when defining the JMS resources.

Using the administrative console, we created the two entries shown in Table 8-9.

Table 8-9 J2C authentication data entries

J2C authentication data entry	Associated user ID
JMS-ID1	USRWORK
JMS-ID2	USRMGR

8.8.9 JMS queue connection factory definitions

To test what effect different settings have, we used the administrative console to define four queue connection factory definitions, shown Table 8-10 on page 228.

Table 8-10 Queue connection factory definitions

Name	JNDI name	Component-managed authentication alias	Container-managed authentication alias
qcfSampJCA1	jms/qcfSampJCA1	Not set	Not set
qcfSampJCA2	jms/qcfSampJCA2	JMS-ID1	Not set
qcfSampJCA3	jms/qcfSampJCA3	Not set	JMS-ID2
qcfSampJCA4	jms/qcfSampJCA4	JMS-ID1	JMS-ID2

8.8.10 Queue destination definitions

We used the administrative console to define three queue destination definitions as shown in Table 8-11.

Table 8-11 Queue destination definitions

Name	JNDI name	Queue name
queueCanAccess	jms/queueCanAccess	SYSTEM.DEFAULT.LOCAL.QUEUE
queueNoAccess	jms/queueNoAccess	ITSO.SWIPE.NOACCESS.QUEUE
queueNotThere	jms/queueNotThere	ITSO.SWIPE.NOTTHERE.QUEUE

The intent of these three definitions is to:

1. Provide a queue destination, queueCanAccess, that the SWIPE JMS application should be able to do a PUT and GET on
2. Provide a queue destination, queueNoAccess, that the SWIPE JMS application will *not* be able to do a PUT or GET to.
3. Provide a queue destination, queueNotThere, just to test what happens if you try to access a queue destination for which the specified queue does not exist.

8.8.11 SWIPE JMS: Logical resources

In the SWIPE JMS application, logical resources for the queue connection factories and queue destinations where defined. The same definitions where used in the web.xml file, which is used by the servlet, and in the ejb-jar.xml file, which is used by the EJB. The definitions defined are shown in Table 8-12 on page 229.

Table 8-12 SWIPE JMS: Logical resources

Resource reference	JNDI name	Authentication
jms/qcfContJCA1	jms/qcfSampJCA1	Container
jms/qcfContJCA2	jms/qcfSampJCA2	Container
jms/qcfContJCA3	jms/qcfSampJCA3	Container
jms/qcfContJCA4	jms/qcfSampJCA4	Container
jms/qcfAppJCA1	jms/qcfSampJCA1	Application
jms/qcfAppJCA2	jms/qcfSampJCA2	Application
jms/qcfAppJCA3	jms/qcfSampJCA3	Application
jms/qcfAppJCA4	jms/qcfSampJCA4	Application
jms/queueCanAccess	jms/queueCanAccess	Not Applicable
jms/queueNoAccess	jms/queueNoAccess	Not Applicable
jms/queueNotThere	jms/queueNotThere	Not Applicable

The JNDI names set for the logical resources match the values specified when defining the queue connection factories and queue destinations, which simplifies the deployment process.

8.8.12 Install the SWIPE JMS application

The process for installing the SWIPE JMS application into WebSphere V5 is similar to that described in 8.3, “Install SWIPE for CICS, IMS, and DB2” on page 212.

8.8.13 Run the SWIPE JMS application

When the SWIPE JMS application is run, the initial output is as shown in Figure 8-19 on page 230.

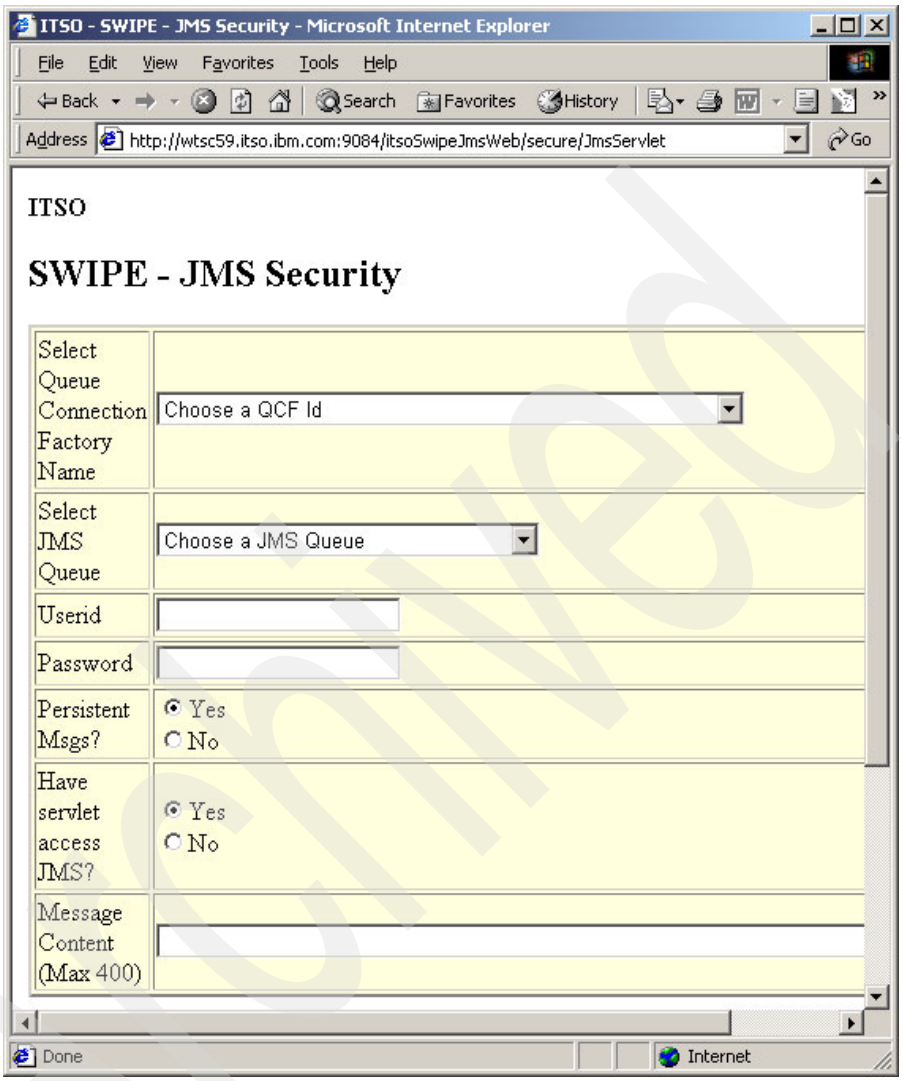


Figure 8-19 Initial SWIPE JMS application display

Not shown in Figure 8-19 are two buttons labelled Put Message and Get Message.

Enter some text in the field labelled Message Content, which will be the content of the message written to a queue if you do a PUT to the queue. If you do a GET from a queue, the contents will be displayed in this field.

When one of the action buttons is pressed, the servlet will attempt the PUT or GET operation first, and then invoke the session EJB to also perform the operation. There is a flag, the Have Servlet access JMS field, you can set on the form to stop the servlet performing the PUT or GEToperation, if desired.

To test a security combination, select a value from the drop-down boxes for Queue Connection Factory Name shown in Figure 8-20, and Select JMS Queue shown in Figure 8-21 on page 232.

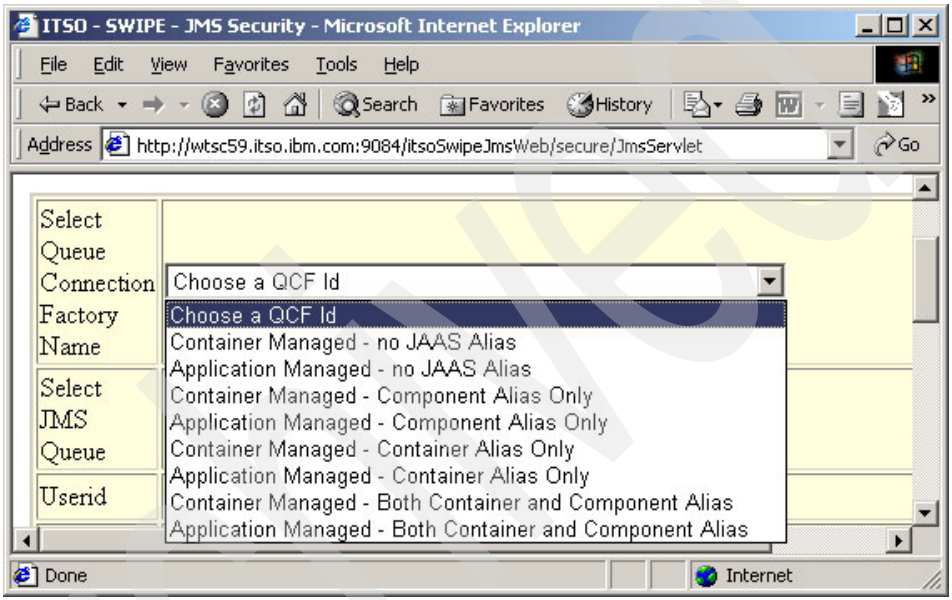


Figure 8-20 SWIPE JMS: Selecting a Queue Connection Factory

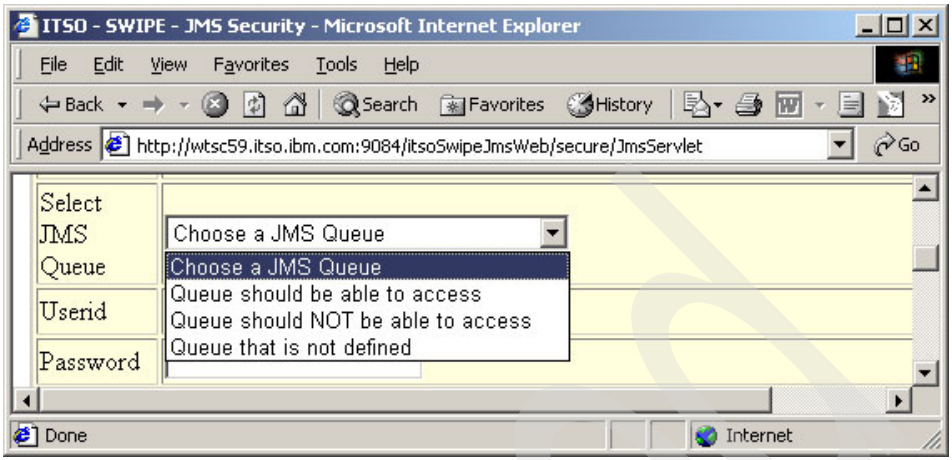


Figure 8-21 SWIPE JMS: Selecting a JMS Queue Destination

Figure 8-19 on page 230 also shows that there are two input fields called Userid and Password. You can enter values here that will then be used when the queue connection factory is created by the SWIPE JMS application, to test what effect this has.

Results

When you press **Put Message** or **Get Message**, results of what happened are displayed below the input form. If you let the servlet perform the PUT or GET operation, an example of a successful PUT operation by the servlet is shown in Figure 8-22 on page 233.

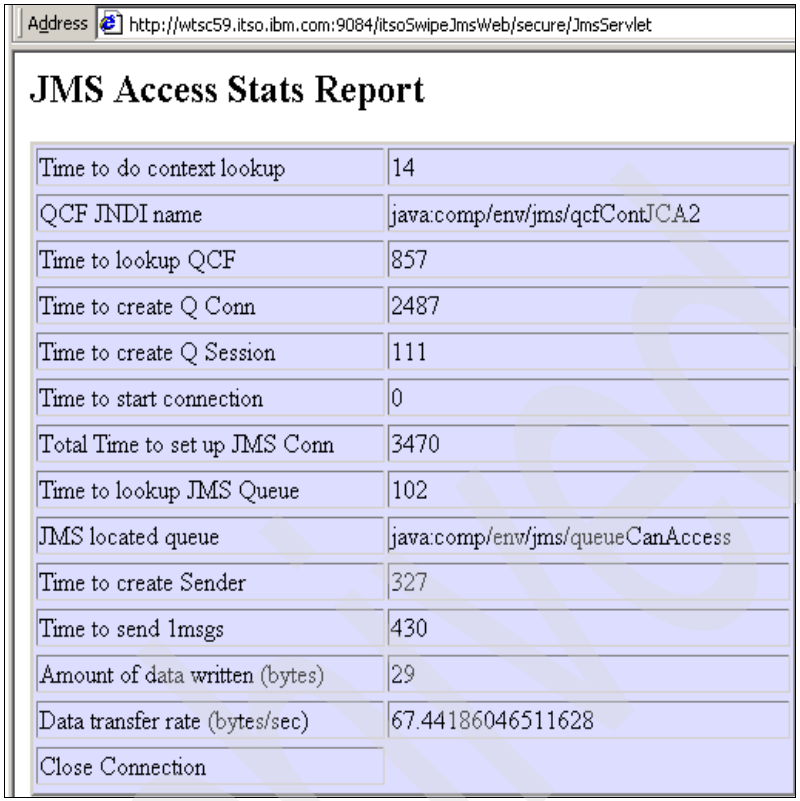
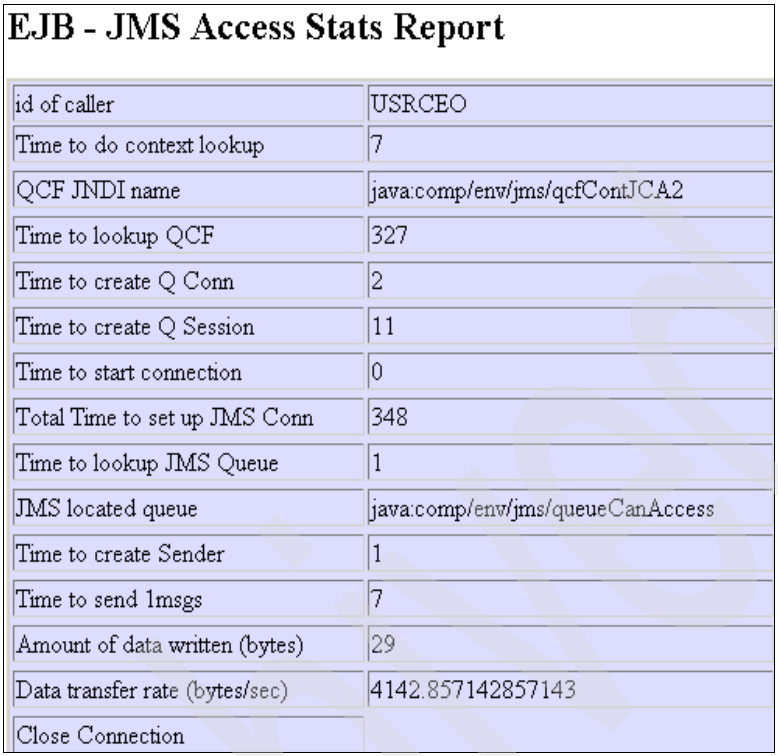


Figure 8-22 SWIPE JMS: Result of a servlet performing a JMS operation

An example of the EJB performing a successful PUT operation by the servlet is shown in Figure 8-23 on page 234.



EJB - JMS Access Stats Report

id of caller	USRCEO
Time to do context lookup	7
QCF JNDI name	java:comp/env/jms/qcfContJCA2
Time to lookup QCF	327
Time to create Q Conn	2
Time to create Q Session	11
Time to start connection	0
Total Time to set up JMS Conn	348
Time to lookup JMS Queue	1
JMS located queue	java:comp/env/jms/queueCanAccess
Time to create Sender	1
Time to send 1msgs	7
Amount of data written (bytes)	29
Data transfer rate (bytes/sec)	4142.857142857143
Close Connection	

Figure 8-23 SWIPE JMS: Result of an EJB performing a JMS operation

8.8.14 RACF messages

When you run the SWIPE JMS application (and assuming you are using RACF on your system), when the SWIPE JMS application is run in WebSphere and it tries to connect to WebSphere MQ and perform the PUT or GET operation, then if there is a security violation, a RACF security violation message will be written to the log.

For example, attempting to access the queue that has been set up to not allow PUT or GET produces a message in the WebSphere MQ MSTR started task similar to that shown in Example 8-3.

Example 8-3 RACF violation message

```
ICH408I USER(USRCEO) GROUP(SYS1  ) NAME(CEO - COMPANY HEAD)
  MQ59.ITSO.SWIPE.NOACCESS.QUEUE CL(MQQUEUE )
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(UPDATE )  ACCESS ALLOWED(NONE )
```

8.8.15 Check the user ID that flows to WebSphere MQ

When using this SWIPE JMS application to test out security, an important part is to determine what user ID is connecting to WebSphere MQ from WebSphere.

There are a few approaches that can be used. One is to select the Queue Destination called “Queue should NOT be able to access”, and ensure that you have set up a RACF rule to prevent any application from being able to write to the queue. This will generate a RACF violation message that will show the user ID that is attempting to update the queue, as shown in Example 8-3 on page 234.

Another way is to write the message to SYSTEM.DEFAULT.LOCAL.QUEUE, and then check the user ID set in the message header part of the message on the queue. There is a SupportPac called MA10 that can be downloaded from:

http://www.ibm.com/software/integration/support/supportpacs/

This support pac provides an ISPF panel that can display messages, including the message header, that are on a queue. Figure 8-24 shows a sample display from this ISPF panel, showing the field in the message header that contains the user ID.

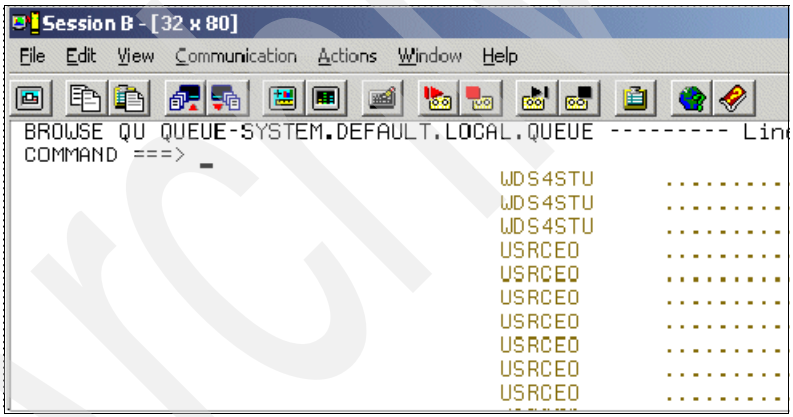


Figure 8-24 MA10 SupportPac: Browsing a queue

Part 3

Cryptography

In this part, we discuss cryptography. Cryptography is central to the correct functioning of a production WebSphere Application Server for z/OS Version 5 environment. Cryptography is not only about providing secure communications across a network. WebSphere Application Server for z/OS Version 5 uses cryptography in other ways, such as when creating tokens for use in support of single sign-on or session affinity.

The zSeries hardware provides unique hardware encryption capabilities. A production WebSphere Application Server for z/OS Version 5 environment will almost certainly need to exploit this because of the heavy use of cryptography in a production environment. Therefore, as part of your security design you should plan the implementation of hardware encryption and how to create and manage keys, *before* you start to implement security features in WebSphere Application Server for z/OS Version 5.

9

Using cryptographic services

This chapter describes the cryptographic services offered and used by WebSphere Application Server for z/OS and OS/390.

9.1 Cryptographic support

Not all information can be protected using techniques such as physical security and access control. Although we can use physical and access control to protect information at both the sender and receiver sites, we need additional methods to protect information passing from one site to another across public communications networks. While the information is in this unprotected environment, a malicious hacker or anybody else with the right equipment and expertise could intercept it. Although it is difficult to prevent this from happening, the actual data can be concealed so that even if it is intercepted, it is unintelligible to the person who intercepted it. This is what cryptography does.

Cryptography includes a set of techniques and algorithms for scrambling, or enciphering, data based on a key so that it is available only to someone who can use the algorithm and a complementary key to restore the data to its original form. Cryptography plays a major role in security, not just on the zSeries engines, but also for the Internet.

Most cryptographic systems combine two elements: A process or algorithm that is a set of rules specifying the mathematical steps needed to encipher or decipher data, and a cryptographic key (a string of characters or numbers) or keys that the algorithm uses to select one relationship between plaintext and ciphertext out of the many possible relationships the algorithm provides.

Cryptographic terms and functions which are used for this chapter are described Table 9-1.

Table 9-1 Cryptographic functions

Term	Description
Enciphering	Converting plaintext (which is intelligible) to ciphertext (which is not intelligible). Enciphering is also called encrypting.
Deciphering	Converting ciphertext back into plaintext. Deciphering is also known as decrypting.
Hashing	Using a one-way calculation to condense a long message into a compact bit string or message digest.
Generating and verifying digital signatures	Encrypting a message digest with a private key to create the electronic equivalent of a hand-written signature, an unforgettable piece of data that verifies the identity of the signer.

In current computer systems, cryptography provides a strong, economical basis for keeping data confidential and for verifying data integrity. With the growth of distributed systems and the increasing use of the Internet, cryptography is

already playing a critical and expanding role in providing confidentiality and integrity of data for a large number of financial transactions between companies. It also protects the confidentiality of personal identification numbers (PINs) for automated teller machines, the ability to identify and authenticate, and other areas that require secure data transmission and the authentication of the sender, such as generating electronic signatures. Digital signatures also serve to ensure that the signed document has not been altered since it was signed. Secure Sockets Layer (SSL) uses cryptography both to authenticate clients and servers and to encrypt data. 9.8, “Set up Secure Sockets Layer (SSL) for WebSphere for z/OS” on page 265 describes the SSL enablement itself.

Note: The zSeries 990 and 890 are *not* shipped with the Cryptographic Coprocessor Facility (CCF) that came standard with earlier systems. The PCI Cryptographic Coprocessor (PCICC) is also no longer an option. New cryptographic hardware functions are now supported that provide increased performance and features. Refer to 9.3, “Hardware cryptography support for zSeries 2084 or 2086 engines” on page 245 for further information and differences.

z/OS supports three different cryptographic hardware engines:

- The Cryptographic Coprocessor Feature (CCF), which has been available since 1994
- The Peripheral Component Interface Cryptographic Coprocessor (PCICC)
- The PCI Cryptographic Accelerator (PCICA), introduced in April 2002 with the z900 Turbo models

The CCF is purely a hardware implementation; no microcode executes within the secure cryptographic boundary. It provides very fast Data Encryption Standard (DES) encryption, has a very fast secure RSA (Rivest, Shamir Adleman) signature and key distribution capability. These functions are becoming more and more critical to the enablement of SSL on z/OS.

PCICC, an optional feature on zSeries systems, can more rapidly support new functions within the secure hardware boundary than can CMOS, which requires longer engineering and development cycles. The PCICC feature can support the growing demands of SSL with special enhancements and adaptation packaging. The PCICC feature card has an operating system and a C programming environment for the implementation of new security functions.

With the PCICC card you can write your own functions within the PCICC card and have these functions accessed by Integrated Cryptographic Service Facility (ICSF), the application programming interface to the cryptographic hardware. PCICC has excellent RSA performance, especially for the RSA private key

operation that is used in the SSL handshake. With a single PCICC card you can perform about twice as many SSL handshakes per second than with a CCF engine.

The PCICA has further increased SSL performance capabilities on the z900 Turbo models. Each PCICA feature contains two cryptographic accelerator cards and can support up to 2100 RSA operations per second.

9.2 How WebSphere fits in z/OS and zSeries cryptographic infrastructure

zSeries provides various hardware solutions for cryptographic support. z/OS itself delivers the operating system interfaces to the hardware (ICSF) and also the corresponding APIs. zSeries cryptographic support is integrated into the WebSphere implementation to support the SDK and J2EE security specifications and APIs.

9.2.1 Supported J2EE APIs

WebSphere and its J2EE applications use these services in different ways:

- They use the z/OS System SSL (SSSL) repertoire type for Web container and ORB transport. z/OS System SSL fully supports all of the available hardware features of the zSeries platform.
- The Java Security Socket Extension (JSSE) implementation is mainly used for JMX and SOAP based requests and communication between deployment manager and node agents. Java Secure Socket Extension (JSSE) provides an API within the J2EE programming model for invoking a Java version of SSL and Transport Layer Security (TLS) from within a component. Developers can provide authentication of clients and servers, non-repudiation and data integrity for communication with remote (especially non-J2EE) applications over many standard TCP/IP-based protocols (such as HTTP, FTP, and Telnet).
- Java Cryptography Extension (JCE) provides an API within the J2EE programming model for direct application access to cryptographic functions such as data encryption, key generation and key agreement and Message Authentication Code (MAC) generation and verification. This might prove useful for encrypting or digitally signing portions of data within a message or performing other cryptographic functions not automatically provided in other layers of the programming model or through the security APIs.

Tip: At the time we set up keyrings for JSSE, there was a need to install them into the HFS file system. With the WebSphere V5.01 service level, this requirement is gone and you can use RACF keyrings even for JSSE-based SSL handshakes.

9.2.2 SSL overview

Secure Sockets Layer (SSL) is one of the low-level services supported by z/OS cryptographic services. SSL is a pervasive encryption and authentication protocol on the Internet, where it runs over TCP/IP. SSL, as configured by WebSphere SSL repertoires, supports all encryption and low-level authentication services for WebSphere HTTP and IIOP transports.

An overview of the SSL protocol is useful for understanding the configuration of the WebSphere repertoires and associated HTTP and IIOP transports. In particular, this explains the role of certificates, which must be made available to WebSphere servers and to the clients and servers with which WebSphere interacts; management of the certificates is one of the most significant interoperability issues when setting up WebSphere SSL.

Certificates

Certificates are an essential part of configuring SSL. Other aspects of SSL configuration are generally boilerplate, for example, designating timeouts and supported encryption algorithms; these will come as a default setup with the product that you are configuring, and you will not generally have to configure them. However, certificates must be carefully configured.

A certificate (more formally referred to as a “digital certificate”) is a set of data, organized into one of several industry standard formats, that contains the following information, in addition to other information that is less critical to this discussion:

- Issuer’s distinguished name
- Certificate owner’s distinguished name
- Validity dates
- Public key
- A signature created by the issuer

The certificate is served to anyone who initiates an SSL connection to the owner’s system, where there is also an associated private key, which is never issued to anyone. The public key on the certificate and the private key on the certificate owner’s system form the basis for the SSL “handshake,” which allows

the client to authenticate the server, the server to authenticate the client, and for all data flowing over the connection established by the handshake to be private (not readable by anyone other than the client and the server). The important aspect of a public and private key pair is that if one of the keys is used to encrypt a message, only the other key can be used to decrypt the message.

Further details about certificates and the SSL protocol are readily available on the Internet, and there are many related standards that apply.

SSL handshake

An SSL handshake is a part of the SSL standard and is used to establish an authenticated, secure, private connection between server and client. The handshake is initiated by a client, typically an Internet browser. The flow of an SSL handshake is shown in Figure 9-1, including the sharing of server and client certificates.

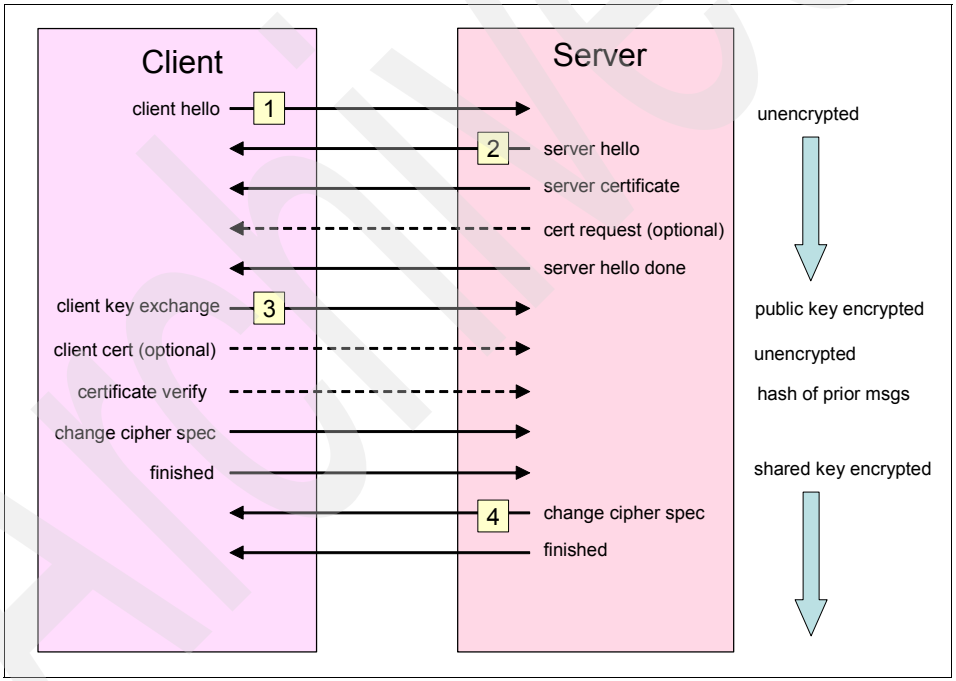


Figure 9-1 SSL V.3 handshake

The item numbers in the following list correspond with the numbers in Figure 9-1:

1. The client begins the SSL handshake with a “hello” message.

2. The server sends a server “hello” message, the server certificate, and optionally a certificate request message to indicate to the client that a certificate is required. The server then sends the server “hello done” message and waits for the client’s response.
3. The client first sends a “client key exchange” message, which is encrypted with the server’s public key; the message contains enough information to enable the client and server to construct private keys that can be used for data following the completion of the handshake. This is a very important message to understand; only the server can decrypt this message to get at the private information that the client has sent, because only the server has access to the private key associated with the public key.

   If the server requested a certificate, the client sends a certificate if available. Otherwise, it sends a “no certificate” alert message. In the case of a Web browser, the browser either sends a preselected certificate or prompts the user to select a certificate from the list of installed certificates and to acknowledge sending the certificate. When a client certificate is sent, it is followed by a “certificate verify” message, which contains a signature that allows the server to authenticate that only the client could have sent the series of messages up to this message and including this message.

   The “change cipher spec” message contains no information, but informs the server that the client will use the negotiated private keys and encryption algorithm for all further messages, including the finished message.
4. The server verifies the integrity of the preceding client messages by using the “certificate verify” message, and so authenticates the client. The server then sends back the “change cipher spec” and “finished” messages.

   The client uses the server’s encrypted finish message to complete its authentication of the server. The fact that the server was able to construct a readable, encrypted “finish” message means that the server must have had the private key to decrypt the client’s prior “client key exchange” message.

All further data passed back and forth between client and server is encrypted using the negotiated private keys.

9.3 Hardware cryptography support for zSeries 2084 or 2086 engines

The introduction of the IBM @server zSeries 990/890 server introduces a new set of cryptographic hardware.

The standard Cryptographic Coprocessor Facility (CCF) has been removed. The cryptographic hardware built in is the Central Processor Assist for Cryptographic

Functions, or CPACF. The implementation and function of the CPACF is similar to the CCF and to the PCICA. The similarity to CCF is that a configuration must be loaded to activate the use of the CPACF. The configuration or enablement diskette is an optional, charged feature. The similarity to the PCICA is that there are no master keys associated with CPACF. The two major implementation and usage differences between the CPACF and the CCF is that the:

- CPACF performs a restrictive set of functions, limited to use with clear key values and supporting only the DES and Triple-DES algorithms.
- Enablement diskette loading is a nondisruptive task.

The existing PCI Cryptographic Accelerator (PCICA) is available for use and order.

A second new cryptographic hardware feature is available for shipment. This new feature is the PCIXCC. This feature is the replacement hardware for most of the secure key functions previously available with the CCF and PCICC hardware features. Implementation is similar to that performed for the PCICC. There are two major implementation differences between the PCIXCC and the PCICC:

- PCIXCC has no enablement diskette or FCV load requirement. The configuration information will be obtained from the CPACF, which is a prerequisite feature.
- PCIXCC has no requirement for CHPID.

ICSF is still the interface to the cryptographic hardware. Additional changes to the z/OS V1.4 operating system code and ICSF code have been made to recognize and interface with the new hardware. This code is available as a Web deliverable and referred to as “Compatibility Code.” Back releases of z/OS and OS/390 V2R10 also have available updates. See the following Web site for more information about z990/z890 compatibility for selected releases:

`http://www.ibm.com/servers/eserver/zseries/zos/downloads/`

The following list summarizes WebSphere Application Server for z/OS and OS/390 use of z990/z890 cryptographic hardware:

- WebSphere Application Server Versions 4 and 5 use System SSL, which takes advantage of the CPACF and PCICA features for hardware support of SSL. If no cryptographic hardware is installed, System SSL uses general-purpose CPU cycles.
- WebSphere Application Server Version 4 *requires* the PCIXCC feature if *form-based authentication* is used by applications deployed on the server.
- WebSphere Application Server Version 5 *requires* the PCIXCC feature if the authentication mechanism selected is *ICSF*. If the authentication mechanism is LTPA, the PCIXCC is not required.

Where differences exist in the setup, notes and special instructions will point them out.

9.4 Activation of hardware cryptography support for zSeries 2084, 2086, 9672, 2064, 2066, or 7060 engines

These instructions are for activating the cryptographic hardware and software in support of SSL and form-based authentication as used by the WebSphere Application Server V4 or V5, the IBM HTTP Server for OS/390 and z/OS, the SSL-enabled TN3270 server, LDAP server, FTP server, and so on. Optional cryptographic features such as the Trusted Key Entry workstation, the PCI Cryptographic Coprocessor, and the PCI Cryptographic Accelerator are not discussed.

While these instructions should be adequate for activating cryptographic services for proof-of-concept or testing, additional training is required on the management of cryptographic services in a production environment.

You are strongly encouraged to attend the IBM Learning Services class, S/390 & zSeries Crypto Hardware, ICSF, TKE Installation and Overview Workshop. The course number is ES801 (ES800 in Canada). Call (800) IBM-TEACH for more information or to register.

The following steps are necessary to enable cryptographic hardware support for the listed zSeries engines:

1. Verify that your processor model has Cryptographic Coprocessor(s).
2. Obtain the configuration enablement diskette(s) for your processor.
3. Load the configuration enablement diskette(s).
4. Assign Cryptographic Coprocessor(s) to LPARs.
5. Install and initialize ICSF, the Integrated Cryptographic Service Facility.
6. Start ICSF.
7. Initialize the CKDS and PKDS and load your master key.
8. Configure WebSphere to use hardware cryptographic services.

The following eight sections cover each of the above tasks in detail.

9.4.1 Verify that your processor has Cryptographic Coprocessor

The Cryptographic Coprocessor is a hardware feature available on current IBM S/390 and zSeries processors. It provides special purpose hardware for improved cryptographic performance and key storage.

This document refers to the processor families by their machine types. As a reminder:

- The z990 is the 2084.
- The z890 is the 2086.
- The z900 is the 2064.
- The z800 is the 2066.
- The S/390 Parallel Enterprise Server™ is the 9672.
- The Multiprise® 3000 is the 7060.

Note: The 2084 and 2086 models no longer offer the Cryptographic Coprocessor Features (CCFs). Functionality is available in other forms. Refer to 9.3, “Hardware cryptography support for zSeries 2084 or 2086 engines” on page 245 for more information. Current 9672 models (G4 and later) and all 2064 models have one or two Cryptographic Coprocessor Features (CCFs) standard. The 7060 and 2066 models do not have CCFs standard, but they can be ordered as optional features. The presence of one or two CCFs is indicated by feature code 0800.

9.4.2 Obtain the correct configuration enablement diskette or diskettes for your processor

The Cryptographic Coprocessor Feature (CCF) (and the CPACF in 2084 and 2086 machines) is shipped disabled to comply with U.S. Export Regulations. They are enabled by loading one or two configuration enablement diskettes, which are not included with feature code 0800, but have an optional feature code of their own. They will have been shipped with your processor (if they were ordered), or they can be ordered using a no-charge MES later. Note that earlier processors shipped an enablement diskette for each cryptographic coprocessor (1 or 2) on the processor. Recently, processors have shipped with one or both enablement files on a single diskette.

Configuration enablement diskettes are unique; only the diskette or diskettes created for your machine serial number will work in your processor.

After you have confirmed that your processor has cryptographic coprocessor or coprocessors, you will need to determine the following:

- Whether your processor was ordered with configuration enablement diskettes
- Whether they were ever loaded
- Where they are currently located

Check the configuration data for your processor to see whether the configuration enablement diskettes were ordered. The feature code varies with processor family, but will be in the 08xx range, and the description will probably be TDES with PKA or TDES. If your processor configuration includes the Trusted Key Entry workstation, make sure you have the configuration enablement diskettes that indicate this (for example, the description will read “TDES with PKA and TKE”).

As of November 2002 (see announcement letter 902-178), the two versions of configuration enablement diskettes (that is, “with TKE” and “without TKE”) have been consolidated into a single version, to simplify the ordering process.

The sales manual (online at the IBMLink™ Web site) will list the correct feature code for your machine/model. If the feature code was not ordered, place a no-charge MES for it as soon as possible.

If the configuration enablement diskettes were ordered, you will need to locate them. To start, check with your IBM Customer Engineer (CE). The diskettes are sometimes stored in the processor cabinet or other storage area accessible by the CE. If they were originally ordered but are missing, the CE can obtain replacements by contacting the IBM Quality Hotline.

9.4.3 Load the configuration enablement diskette(s)

Loading the configuration enablement diskettes requires that you have a user ID and password, as well as physical access to the Hardware Management Console (HMC) or Support Element. This enables you to insert the diskettes into the HMC or Support Element diskette drive. For processors with an integrated Support Element (such as 9672 G5 and G6) or any other processor where the Support Element is locked inside the processor cabinet, you should use the HMC. For safety reasons, only the Customer Engineer is authorized to open the processor cabinet. If you have not worked with the HMC or the Support Element, you should work with someone who is familiar with it. After loading the configuration enablement diskettes, a power-on reset (POR) will be required on your processor.

If the configuration enablement diskettes were ordered with your processor, it’s likely they were loaded by the CE when your processor was installed. If you order

the configuration enablement diskettes after the processor has been installed, you will need to schedule some time for a POR.

Reference information describing how to load the Configuration Enablement Diskettes is available in the *Support Element Operations Guide* for your processor. Accessing Support Element functions through the HMC is described in the *Hardware Management Console Operation*s guide for your processor. These books are available at:

http://www.ibm.com/servers/s390/os390/bkserv/hw/disc1_srch.html#titles

Or:

http://www.ibm.com/servers/s390/os390/bkserv/hwpdf/z900.html

Here are the instructions for loading the configuration enablement diskettes, pulled together from various sections of the Support Element and HMC guides:

1. Sign on to the HMC using the system programmer or service representative user ID.
2. Establish a Support Element session from the HMC, as follows:
   a. Open the **Task List** from the Views area. (Note on terminology: the Views area is the colored area at the top half of the screen. The Work area is the area at the bottom half of the screen. The Tasks area is the vertical area on the right side of the screen.)
   b. Open **CPC Recovery** from the Task List Work Area.
   c. Open **Groups** from the Views area.
   d. Open **Defined CPCs** from the Groups Work Area.
   e. Identify the CPC with the Support Element that you want to connect to.
   f. Drag and drop (right-click and hold) the selected CPC to Single Object Operations in the CPC Recovery tasks area. The Single Object Operations Task Confirmation window opens. Click **Yes** to continue establishing a session with the Support Element. The Support Element Workplace session window opens, and your Support Element session has begun.
3. Continuing in the Support Element Workplace window, open the **Task List** view.
4. Open the **CPC Configuration** task in the Task List Work Area.
5. Open the **Groups** view.
6. Open the group containing the CPC.

Note: If you are configuring a z890 or z990, now would be a good time to record the information for the installed cryptographic hardware. Drag the selected CPC onto the PCI Cryptographic Management icon in the Tasks area. Any installed PKICA and PCIXCC features will be listed. PCICA features will be assigned two PCI cryptographic numbers. *Record* the information on this window for future reference.

7. Drag the CPC onto the Cryptographic Coprocessor Configuration under the CPC Configuration.
8. On the Cryptographic Coprocessor Configuration window, note the Current Configuration Description. Use the slider bar to help. If the Current Configuration Description displays DES/TDES w/PKA, DES/TDES w/PKA & TKE, or something similar, the cryptographic enablement diskettes have already been loaded. You can skip to the next section, 9.4.4, "Assign Cryptographic Coprocessors to LPARs" on page 252. If the Current Configuration Description displays "No Configuration chosen," the cryptographic enablement diskettes have not been loaded.
9. If the cryptographic enablement diskettes have not been loaded, click the line for coprocessor 0 to highlight it, and then click **Import**. A pop-up window will request that you insert the correct Enablement Diskette into the diskette reader of the HMC. The diskettes are labeled by cryptographic coprocessor module identifier and the pop-up window will indicate by identifier which diskette to insert. Insert the diskette and click **Import**. You will receive a warning message that you are about to import the diskette. Click **Enter** to continue. A pop-up window will indicate that the import was successful. Click **OK** to continue.
10. Now, with coprocessor 0 highlighted on the Cryptographic Coprocessor Configuration screen, click **Select for next activation**. On the Select Cryptographic Coprocessor for Next Activation screen, in the Next Configuration list box, highlight the description of the cryptographic enablement diskette you just imported. Select **Auto-initialize** and click **Save**. You will receive a warning message that the coprocessor will be initialized on the next activation. Click **Enter** to continue. You will receive a message to remove the enablement diskette. Remove the diskette and click **OK**.
11. If your processor also has a coprocessor 1, complete steps 9 and 10 (above) to import the cryptographic enablement diskette for coprocessor 1.
12. A power-on reset (POR) must occur for the cryptographic enablement to take effect. However, if you have not already done so, you should complete the next section (9.4.4, "Assign Cryptographic Coprocessors to LPARs" on page 252) before you perform the POR. The POR will then activate the changes made in both steps.

9.4.4 Assign Cryptographic Coprocessors to LPARs

If your processor is operating in LPAR mode, the cryptographic coprocessors must be assigned to specific LPARs so that they can be accessed. This is done by customizing either a reset profile or an image profile using the Hardware Management Console (HMC) to access the Support Element.

If your processor is operating in basic mode, all cryptographic coprocessors are always available to the processor complex. There's nothing for you to do in this step. You should skip all the tasks listed here in this section and proceed with the POR as described in the last task of the previous section, 9.4.3, "Load the configuration enablement diskette(s)" on page 249. Then proceed with the next section, 9.4.6, "Install and initialize Integrated Cryptographic Service Facility" on page 256.

Reference information about how to assign the cryptographic coprocessors is available in the *Support Element Operations Guide* for your processor. Accessing Support Element functions through the HMC is described in the *Hardware Management Console Operations Guide* for your processor. These books can be found at one of the following sites:

`http://www.ibm.com/servers/s390/os390/bkserv/hw/disc1_srch.html#titles`

Or:

`http://www.ibm.com/servers/s390/os390/bkserv/hwpdf/z900.html`

Important: The instructions below assume you are assigning the cryptographic coprocessors to existing partitions on a central processor complex operating in LPAR mode.

For each partition you want to have access to the cryptographic coprocessors, customize that partition's image profile as follows:

1. If you are continuing from the previous section, 9.4.3, "Load the configuration enablement diskette(s)" on page 249, you will already have started a Support Element session on your HMC. If not, start a Support Element session on your HMC as follows:
   a. Sign on to the HMC using the system programmer or service representative user ID.
   b. Open **Task List** from the Views area. (Note on terminology: the Views area is the colored area at the top half of the screen. The Work area is the area at the bottom half of the screen. The Tasks area is the vertical area on the right side of the screen.)
   c. Open **CPC Recovery** from the Task List Work Area.

d. Open **Groups** from the Views area.

e. Open **Defined CPCs** from the Groups Work Area

f. Identify the CPC with the Support Element that you want to connect to.

g. Drag and drop (right-click and hold) the selected CPC to Single Object Operations in the CPC Recovery tasks area. The Single Object Operations Task Confirmation window opens. Click **Yes** to continue establishing a session with the Support Element. The Support Element Workplace session window opens, and your Support Element session has begun.

2. From the Support Element Workplace window, open the **Task List** view.
3. Open the **CPC Operational Customization** task in the Task List Work Area.
4. Open **Groups** from the Views area.
5. Open the **Images** group from the Groups Work Area.
6. Locate the image with the same name as the logical partition.
7. Drag and drop (right-click and hold) the logical partition on the Customize/Delete Activation Profiles task to start it. This opens the image profile and the list of load profiles that you want to customize. When the list is initially displayed, the highlighted profile is the currently assigned profile for the partition.
8. Select from the list the name of the image profile you want to customize.
9. Click **Customize**. If you are running on a z890 or z990 system, refer to 9.4.5, “Additional instruction for assigning the PCI crypto features to LPARs with a 2084 or 2086 engine” on page 254 for a continuation of the instructions.
10. Click the **Processor** tab of your image profile. (If you are in basic mode, there is no Processor tab.) Highlight one or both cryptographic coprocessors to make them available to that partition.

    Note: If central processors are dedicated to a specific partition, the cryptographic coprocessors are dedicated as well. They will not be available to any other partitions.

    When you highlight one or both cryptographic coprocessors, a Crypto tab (as well as a PCI Crypto tab) is added at the bottom of your image profile.

11. Click the **Crypto** tab of your image profile. (If you are in basic mode, there is no Crypto tab.) Ensure that a check mark is present in the following check boxes on the Crypto page:
    - Enable public key algorithm (PKA) facility.
    - Enable cryptographic functions.

- Enable special security mode.
- Enable integrated cryptographic facility (ICRF) key entry.
- Enable public key secure cable (PKSC) and integrated cryptographic service facility (ICSF).
- The rest of the check boxes can be left blank.
- For the Control domain index and the Usage domain index, highlight the value that corresponds with your LPAR number, from 00 to 15. Write the LPAR number down for use later.

12. When done updating the Crypto page, click the **save** button. You might have to use the scroll bar on the right side of your screen to scroll down to the **save** button.
13. Complete steps 6 through 12 for each partition you activate Crypto on.
14. At this point each partition you modified must be deactivated and reactivated for the changes to take effect.
15. Log off of the Support Element session on your HMC by pressing and holding the Alt key and pressing F4.
16. Log off of the HMC by pressing and holding the Alt key and pressing F4.

9.4.5 Additional instruction for assigning the PCI crypto features to LPARs with a 2084 or 2086 engine

This section assumes that you:

- Are configuring a z890 or z990
- Have successfully followed the instructions in 9.4.4, “Assign Cryptographic Coprocessors to LPARs” on page 252 up to and including step 9 on page 253

For each partition, customize the image profile with the usage domain index, control domain index, PCI crypto candidate list, and PCI crypto online list.

The *usage domain index* is the domain number used by ICSF. The domain number is an index to which one of the 16 master key registers this partition will use as its master key register. The same usage domain number is specified in the image profile and in the ICSF options data set used by that partition.

The *control domain index* specifies the domains that the usage domain can control using the Trusted Key Entry (TKE) workstation. If you are not using the TKE workstation, set the control domain index to the same value as the usage domain index.

The *PCI crypto candidate list* enables you make PCICA or PCIXCC crypto features, or both, available to the partition. The values (from 0 to 16) represent the PCI Cryptographic numbers assigned to the processors on the PCICA and PCIXCC feature cards. These are the PCI Cryptographic numbers you recorded in 9.4.3, “Load the configuration enablement diskette(s)” on page 249.

The *PCI crypto online list* enables you specify which of the PCI Cryptographic numbers in the PCI crypto candidate list will be brought online during logical partition activation.

In most environments, it is desirable to share all crypto cards between all partitions. This provides the most configuration flexibility. Some exceptions are if the OS running on the partition does not support the crypto feature, or where a partition must have exclusive access to a crypto feature for performance reasons. If you plan to share all crypto cards between all partitions, you can assign all possible numbers on the PCI crypto candidate list and PCI crypto online list to each image. Even PCI Cryptographic numbers for which there are no installed crypto feature can be assigned, without causing error messages at activation or IPL time. The advantage of preassigning numbers is that it will save activations when PCI cryptographic features are added at a later date.

The image customization for the image you selected should be open. You can now assign the crypto features to the LPAR:

1. Click the PCI Crypto tab. Here, you will see the usage domain index, control domain index, PCI crypto candidate list, and PCI crypto online list.
2. Set the Usage domain index to the domain number you specified in the ICSF options data set for this partition. Use the slider bar to locate the domain index number and click the number to select it. You should only select one domain number, because OS/390 and z/OS can only use one.
3. Set the Control domain index to the same value you selected for the Usage domain index.
4. Set the PCI Cryptographic Candidate List to include those PCI Cryptographic numbers you want to be accessible from this partition. They do not have to be installed to be selected.
5. Set the PCI Cryptographic Online List to include those PCI Cryptographic numbers you want to be brought online to this partition at activation time.
6. Click the **Save** button to save the image profile.
7. Repeat these steps for the other image profiles on your z890/z990.
8. At this point, each partition you modified must be deactivated and reactivated for the changes to take effect.

9. Log off of the Support Element session on your HMC by pressing and holding the Alt key and pressing F4.
10. Log off of the HMC by pressing and holding the Alt key and pressing F4.

9.4.6 Install and initialize Integrated Cryptographic Service Facility

Integrated Cryptographic Service Facility (ICSF) runs as a started procedure, and each partition that requires cryptographic services must have ICSF running. ICSF System Programmer's Guide and other helpful ICSF books can be found online at the following locations. Search on books with titles containing "ICSF".

http://www.ibm.com/servers/s390/os390/bkserv/os390/bop10_srch.html#titles

Or:

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zos/zbop3_srch.html#titles

The following is a summary of the steps described in the ICSF System's Programmer Guide. You should perform all of these steps.

1. Customize SYS1.PARMLIB.
2. Create the Cryptographic Key Data Set (CKDS) and Public Key Data Set (PKDS).
3. Create the installation options data set.
4. Create the ICSF startup procedure.
5. Provide access to the ICSF panels.

Notes: Each partition that requires cryptographic services must have ICSF running. Each ICSF must have its own installation options data set. The sample installation options values specified in the ICSF System Programmer Guide will work, with the following comments/exceptions:

- The installation options data set value for SSM should be YES.
- A PKDS data set is required as of OS/390 V2R9. It's a good idea to define it even if you are on an earlier release.
- The "domain" value specified in the installation options data set must match the "Usage Domain" you highlighted on the Crypto tab for the LPAR in task 11 in 9.4.4, "Assign Cryptographic Coprocessors to LPARs" on page 252. Each partition must have a unique domain value. The hardware won't allow them to share. This means that each partition which requires cryptographic services must have its own installation options data set.
- If your processor is operating in basic mode, pick a whole number from 0 to 15, and use it for your domain value.
- Partitions can share CKDS and PKDS data sets, if the same master keys are loaded into each partition.

9.4.7 Initialize the CKDS and PKDS and load your master key

Notes: For the *z890* and *z990*:

If no PCIXCC features are assigned to a partition, you have completed the activation of cryptographic services for that partition. It is not possible (or necessary) to load master keys or initialize the CKDS or PKDS for a partition with no PCIXCC assigned. ICSF will support the CPACF and any PCICA features, and cryptographic services will be active with an uninitialized CKDS and PKDS and no master keys.

The rest of this step assumes that you have at least one PCIXCC feature installed.

If you are migrating a CKDS and PKDS from a previous ICSF implementation, it is necessary to load the same master keys that were used on the previous implementation. This is described in Chapter 6, "Managing Master Keys - PCI X Cryptographic Coprocessor" of the *z/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521. Entering clear master keys is outside the scope of this document.

If you have created a new CKDS and PKDS for a partition, the easiest way to load the master keys is to use the ICSF Pass Phrase Initialization Utility described in this section.

After the ICSF address space is started, you should be able to access the ICSF administrative panels by entering ICSF from the ISPF main menu. At this point, you must initialize the CKDS and PKDS and load your master keys using the ICSF panels. You will do this once on each partition where cryptographic services are required. The easiest way to do this is to use the Pass Phrase Initialization Utility, which is well documented in Chapter 4 of *Iz/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521.

Pass Phrase initialization tips

Consider the following tips:

- The following information is necessary on the Pass Phrase Initialization panel:
  - Pass Phrase: A 16- to 64-character phrase that is the basis for your new Master Key.
  - CKDS: The data set name of your VSAM Cryptographic Key Data Set.
  - Initialize the CKDS? (Y/N): If you will not be sharing the CKDS and PKDS with another partition, specify `Y` for the question "Initialize the CKDS?".
  - Signature MK = Key Management MK?: Specify `Y`.
- The Pass Phrase Initialization panel can be found on the ICSF main menu. It is option PPINIT.
- Be sure to write down your Pass Phrase and store it in a safe place.
- It is not necessary to enclose your Pass Phrase in quotes. If you do, the quotes will be treated as part of the Pass Phrase.
- You should enclose your CKDS data set name in quotes. Otherwise, TSO will prefix the name with your TSO user ID and won't find the data set.
- If you will be sharing the CKDS and PKDS with another partition, each partition must use the same Pass Phrase. On the first partition where you run Pass Phrase Initialization, specify `Y` for the question "Initialize the CKDS?". On subsequent partitions that use the same CKDS and PKDS, use the same Pass Phrase but specify N for the question "Initialize the CKDS?".

If the process is successful, you will receive a message that initialization has completed, and the system log will receive messages similar to the following:

```
IEE504I CRYPTO(0),ONLINE
IEE504I CRYPTO(1),ONLINE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
```

On subsequent activations of ICSF you will also see the message:

```
CSFM405A AUTHENTICATION CODE IN PKDS HEADER RECORD DOES NOT MATCH COMPUTED
VALUE.
```

This message is due to not having any keys defined in your PKDS data set. It is a normal message, and as long as you have no keys defined in your PKDS, it is no cause for concern.

Tip: If you have trouble with your Pass Phrase initialization process, see “Troubleshooting ICSF Pass Phrase initialization problems” on page 713.

9.5 Configure WebSphere to use hardware cryptographic services

WebSphere can benefit in the performance and security area if hardware cryptographic support is used.

9.5.1 Configure WebSphere to use hardware cryptography for SSL

WebSphere uses the System SSL component of OS/390 or z/OS and JSSE for cryptographic support of SSL connections. If the cryptographic hardware is installed and enabled, and if ICSF is active and a master key is loaded, System SSL will use ICSF and the cryptographic hardware for SSL operations. There is no option in WebSphere to enable or disable this.

9.5.2 Configure WebSphere to use hardware cryptography in support of the ICSF authentication mechanism

When WebSphere global security is enabled, you choose to use either SWAM, LTPA, or ICSF as the active authentication mechanism, as described in 3.5.1, “Authentication mechanism and single sign-on (SSO)” on page 62. If ICSF is chosen, the name of an existing triple-DES data key must be specified for use in the encryption of login tokens (the LTPA token cookie). The cryptographic hardware *must* be enabled, ICSF *must* be active, and a master key *must* be loaded, as described in the previous sections.

The Triple-DES key must be created and stored in the ICSF Cryptographic Key Data Set (CKDS). The following instructions describe the process.

Creating an ICSF Triple-DES key for use by WebSphere

From the ISPF main menu, type ICSF to invoke the ICSF application panels:

1. The ICSF menu options were rearranged and updated at about z/OS V1R2. If you have the earlier ICSF main menu, select 2 to invoke the KGUP panels. If you have the later ICSF main menu, select option 8 to invoke KGUP.

2. From the KGUP menu, select 1 to create key generator control statements.
3. At the Control Dataset Spec panel, for the data set name, type `userid.CSFIN` and press Enter. Substitute your TSO user ID for “userid”.
4. At the allocation panel, press Enter to take the defaults. Your data set will be allocated and you will see the KGUP Control Statement menu.
5. At the KGUP Control Statement menu, select 4 to edit the statement storage data set.
6. You are now editing the empty userid.CSFIN data set. This is where you add the command to generate the Triple-DES key used to encrypt the LTPA token. If you plan to use single sign-on with ICSF keys across multiple WebSphere V4 or V5 for z/OS servers, the ICSF key values used by all the servers must be the same. This is so the LTPA token generated by any WebSphere server can be read by all the others. If all the servers share the same ICSF data sets (CKDS and PKDS), they can all share the same key label in ICSF, and typing in the following command will be sufficient:

   ```
   ADD TYPE(DATA) LENGTH(24) CLEAR DES LAB(LOGINTOKENKEY)
   ```

   This command generates a random Triple-DES symmetric key called LOGINTOKENKEY. The label LOGINTOKENKEY is just a suggestion; you can call the key whatever you like. The command is not executed until you submit it as part of a KGUP batch job. Press PF3 to save the data set and continue with the next step.

   If not all the servers share the same ICSF data sets, it is necessary to specify the actual value of the Triple-DES key (rather than letting KGUP generate a random number), and install the same key on each server. In this case, do not enter the above command, but complete the following steps:

   a. Press PF3 four times to return to the ICSF main menu.
   b. At the ICSF main menu, select **ICSF Utilities**.
   c. At the ICSF Utilities menu, select **Generate a Random Number**.
   d. At the Random Number Generator panel, specify ODD for the Parity Option and press Enter. Three random numbers will be generated. Record the values of Random Number1, Random Number2 and Random Number3.
   e. Press PF3 twice to return to the ICSF main menu.
   f. From the ICSF main menu, select the **KGUP** option.
   g. At the Key Administration menu, select **Create key generator control statements**.
   h. At the Control Dataset Spec panel, you should see the data set name userid.CSFIN. Press Enter.

i. At the KGUP Control Statement menu, select 4 to edit the statement storage data set.

j. Now you're back to editing the empty userid.CSFIN data set. Add the following:

```
ADD TYPE(DATA) CLEAR,
KEY(<Random Number1>,<Random Number2>,<RandomNumber3>),
LAB(LOGINTOKENKEY)
```

Where <Random Number1> is the first random number you collected from the Generate a Random Number panel, and so on. For example:

```
ADD TYPE(DATA) CLEAR,
KEY(D916925BA2912F89,C715D089A15797C7,4F4FA4D66125575B),
LAB(LOGINTOKENKEY)
```

Save a copy of this command because you must enter it on every ICSF system that you want to enable for single sign-on. Press PF3 to save the data set.

7. At the KGUP Control Statement Menu, press PF3.
8. At the Control Dataset Spec panel, press PF3.
9. At the ICSF Key Administration menu, select 2 to specify data sets for processing.
10. At the Specify KGUP Datasets panel, fill in the following data set names, in single quotes:
    - For Cryptographic Keys, type the name of your CKDS data set. If you do not know the name of your CKDS, you can find it by going to the ICSF main menu and selecting the **OPSTAT** option. From there, select the **OPTIONS** option to Display Installation Options. Your CKDS will be listed as the Active CKDS.
    - For Control Statement Input, type `userid.CSFIN`.
    - For Diagnostics, type *.
    - For Key Output, type `userid.CSFKEYS`.
    - For Control Statement Output, type `userid.CSFSTMNT`.
11. Press PF3 to save.
12. At the ICSF Key Administration menu, select 3 to submit your KGUP job.
13. At the Set KGUP JCL Job Card panel, set Special Security Mode (at the bottom of the screen) to YES, and change the job card information to be valid in your environment.
14. On the Set KGUP JCL Job Card panel, at the command prompt, type `S` to submit.

15. Check the KGUP job SYSOUT to make sure you get a zero return code.
16. At the ICSF Key Administration menu (KGUP main menu), select 4 to refresh the cryptographic key data set. At the Refresh in-storage CKDS panel, for New CKDS, enter your CKDS name and press Enter. REFRESH SUCCESSFUL should appear at the upper right corner of the screen.

The Triple-DES key is now installed in ICSF and can be used by WebSphere.

9.6 Securing and maintaining cryptography

Using Integrated Cryptographic Services Facility (ICSF) is recommended, but not a prerequisite for WebSphere, unless you have chosen ICSF to be your authentication mechanism. Because ICSF provides the control interface to the cryptographic hardware, we must, in turn, protect ICSF functions. Here we talk about how we can secure cryptographic services and what steps can be taken to secure the integrity of ICSF:

- Protect the cryptographic key data sets (CKDS), and master key data sets (MKDS). These data sets contain encryption key values and information about the keys. The CACHE copy of the CKDS can only be accessed through ICSF functions such as callable services, the Key Generator Utility Program (KGUP), or through the administrator panels. Therefore, you need to ensure that you grant the right user IDs the right level of access to these services and tools.
- Maintain duplicate cryptographic key data sets. This copy can be used to replace a damaged original or to retrieve a lost key.
- Control access to services and keys. Anyone executing the KGUP can read and modify the CKDS if it is not protected. To prevent unauthorized access from the KGUP, store the program in an APF-authorized library that is protected by RACF. You can use the CSFSERV and CSFKEYS classes in RACF to grant access or deny access, and also to create audit records. This will prevent unauthorized individuals from using the program to access the cryptographic key data sets.
- Change cryptographic keys frequently. To reduce the possibility of exposing a key value, we recommend scheduling changes to the cryptographic keys, including the DES master key. If you are using ICSF, you can change keys without interrupting cryptographic functions.

Control the execution environment while generating keys. While the KGUP is running, clear (unenciphered) cryptographic keys are available in main storage. As other programs can access these clear keys, execute the utility only when there are no other programs running on the system. Protect and save the output

from this utility, which contains the clear value of each key that was processed, in case you need to recreate a key.

9.6.1 RACF protection for ICSF

To perform the RACF setup for ICSF, use the following commands:

1. Started task user ID

   ```
   ADDUSER ICSFSTU DFLTGRP(STG) NOPASSWORD NAME('ICSF Started task user')
   ```

2. ICSF data sets

   These data sets contain Cryptographic Keys (CKDS) and Public Keys (PKDS): ICSF.CKDS and ICSF.PKDS, respectively.

   ```
   AG ICSF SUPGROUP(DATA) OWNER(DATA)
   AD 'ICSF.** OWNER(SECADM) UACC(NONE)
   ```

3. Profile in class STARTED

   ```
   RDEF STARTED ICSF.** STDATA(USER(ICSFSTU) GROUP(STG))
   ```

4. General Resource Class CSFKEYS

   Profiles in this class protect labels for keys in the format:

   ```
   RDEF CSFKEYS label UACC(NONE) AUDIT(ALL)
   PE label CL(CSFKEYS) ID(SECADM) ACC(R)
   ```

   Note: You might need to create one or more job role groups in addition to SECADM for key management.

   You might also decide to create a common generic profile for all key labels:

   ```
   RDEF CSFKEYS ** UACC(NONE) OWNER(SECADM) AUDIT(ALL)
   ```

   and permit one or several job role groups to it.

5. General Resource Class CSFSERV

   Profiles in this class protect cryptographic services in fixed-name format

   ```
   RDEF CSFSERV service-name UACC(NONE) OWNER(SECADM) AUDIT(ALL)
   PE service-name CL(CSFSERV) ID(SECADM) ACC(R)
   ```

   Similar to CSFKEYS, you can define a common generic profile:

   ```
   RDEF CSFSERV ** UACC(NONE) OWNER(SECADM) AUDIT(ALL) or
   RDEF CSFSERV CSF* UACC(NONE) OWNER(SECADM) AUDIT(ALL)
   ```

   For all profile names in class CSFSERV, refer to Chapter 3 of *z/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521.

If you have not activated CSFKEYS and CSFSERV yet, issue the commands:

```
SETR CLASSACT(CSFKEYS CSFSERV)
SETR RACLIST(CSFKEYS CSFSERV)
SETR RACLIST(CSFKEYS CSFSERV) REFR
```

9.6.2 RACF setup to secure OCSF and OCEP

The use of Open Cryptographic Services Facility (OCSF) and Open Cryptographic Enhanced Plug-ins (OCEP) is controlled by fixed-name profiles in class FACILITY:

CDS.CSSM Authorizes access to OCSF.

CDS.CSSM.CRYPTO Authorizes access to Cryptographic Service Provider.

CDS.CSSM.DATALIB Authorizes access to Data Library (DL) Service Provider.

It is likely that all these resources will have the same access list, so only one generic profile may be defined:

```
RDEF FACILITY CDS.** UACC(NONE) OWNER(SECADM)
```

9.7 Create RACF keyrings and certificates

WebSphere uses HFS and RACF keyrings for its cryptographic processing. From WebSphere 5.01 onwards the requirement for storing keyrings in HFS for the JSSE cryptographic provider is gone and we strongly recommend to use RACF keyrings.

JSSE and SSSL configurations (repertoires) are generated for you during system setup.

The RACF keyrings are created and installed using the JOBS BBORAC and BBOCR2FA from the ISPF installation dialog:

- RACF keyrings are generated.
- Certificates and CA certificates are created and stored within the keyrings.
- DIGTCERT profile permissions are set up.

Tip: For certificate management you can use PKISERV. This might also be helpful to get certificate expiration warnings.

9.8 Set up Secure Sockets Layer (SSL) for WebSphere for z/OS

Secure Sockets Layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP transport, the ORB (client and server), and the secure LDAP client. Configuring SSL is different between client and server with WebSphere Application Server. If you want the added security of protected communications and user authentication in a network, you can use Secure Sockets Layer (SSL) security. The SSL support in WebSphere for z/OS has several objectives:

- To provide ways accepted by the industry to protect the security of messages as they flow across the network

  This is often called *transport layer security*. Transport layer security is a function that provides privacy and data integrity between two communicating applications. The protection occurs in a layer of software on top of the base transport protocol (for example, on top of TCP/IP). SSL provides security over the communications link through encryption technology, ensuring the integrity of messages in a network. Because communications are encrypted between two parties, a third party cannot tamper with messages. SSL also provides confidentiality (ensuring the message content cannot be read), replay detection, and out-of-sequence detection.

- To provide a secure communications medium through which various authentication protocols can operate

  A single SSL session can carry multiple authentication protocols, that is, methods to prove the identities of the parties communicating. SSL support always provides a mechanism by which the server proves its identity. The SSL support on WebSphere for z/OS allows these ways for the client to prove its identity:

  - Basic authentication (also known as SSL Type 1 authentication), in which a client proves its identity to the server by passing a user identity and password known by the target server. With SSL basic authentication:
    - A z/OS or OS/390 client can communicate securely with a WebSphere for z/OS server by using a user ID and password as defined by the CSIv2 Username and Password Mechanism (GSSUP).
    - A distributed platform client can communicate securely with a WebSphere for z/OS server by using a SAF user ID and password.
    - Because a password is always required on a request, only simple client-to-server connections can be made. That is, the server cannot send a client's user ID to another server for a response to a request.

- Client certificate support, in which both the server and client supply digital certificates to prove their identities to each other.

  Web applications can have thousands of clients, which makes managing client authentication an administrative burden. Through RACF certificate name filtering, SSL support on WebSphere for z/OS allows you to map client certificates, without storing them, to MVS user IDs. Through certificate name filtering, you can authorize sets of users to access servers without the administrative overhead of creating SAF user IDs and managing client certificates for every user.

- CSIv2 identity assertion support, which includes z/OS and OS/390 principals, X501 distinguished names, Kerberos principals, and X509 identity certificates.

- Identity assertion, or trusted association, in which an intermediate server can send the identities of its clients to a target server in a secure yet efficient manner. This support uses client certificates to establish the intermediate server as the owner of an SSL session. Through SAF, the system can check that the intermediate server can be trusted (to confer this level of trust, CBIND authorization is granted by administrators to SAF IDs that run secure system code exclusively). After trust in this intermediate server is established, client identities (SAF user IDs) need not be separately verified by the target server; those client identities are simply asserted without requiring authentication.

- To interoperate in a secure way with other products such as:
  - CICS Transaction server for z/OS
  - WebSphere on distributed platforms
  - CORBA-compliant Object Request Brokers

SSL is disabled by default and SSL support is optional. Running WebSphere for z/OS without using SSL affects only the SSL functions that protect communication and authenticate clients and servers. If you choose to use SSL, there are two types of SSL repertoires from which you must choose:

- System SSL (SSSL) is the SSL repertoire type used for Web container and ORB transport.
- Java Secure Socket Extension (JSSE) is the SSL repertoire type used for the JMX SOAP connector.

The following list describes how an SSL connection works:

Stage Description.

Negotiation	After the client locates the server, the client and server negotiate the type of security for communications. If SSL is to be used, the client is told to connect to a special SSL port.
Handshake	The client connects to the SSL port, and the SSL handshake occurs. If successful, encrypted communication starts. The client authenticates the server by inspecting the server's digital certificate. If client certificates are used during the handshake, the server authenticates the client by inspecting the client's digital certificate. Figure 9-1 on page 244 shows the flow and the components required to encipher and decipher data in an asymmetric handshake.
Ongoing communication	During the SSL handshake, the client and server negotiate a cipher spec to be used to encrypt communications.
First client request	The determination of client identity depends upon the client authentication mechanism chosen, which is one of the following: • CSIv2 user ID and password (GSSUP) • CSIv2 asserted identity • zSAS Kerberos • z/SAS basic authentication asserted identities • z/SAS asserted identities

Rules

Only server controllers and z/OS clients require access to Cryptographic Services System SSL. Your controllers and z/OS clients require access to the *hlq*.SGSKLOAD data set. Place SGSKLOAD into LPA. For more information, see *z/OS System Secure Sockets Layer Programming*, SC24-5901.

- Either a Java or C++ client on z/OS can interoperate with a WebSphere for z/OS or workstation server and use SSL. CSIv2 security only supports Java clients on z/OS.
- Part of the handshake is to negotiate the cryptographic specs used by SSL for message protection. There are two factors that determine the cipher specs and key sizes used:
  - The security level of the Cryptographic Services installed on the system, which determines the cipher specs and key sizes available to WebSphere for z/OS.

– The configuration of the server through the administrative console allows you to specify SSL cipher suites.

- For z/OS System SSL sockets you must use RACF or equivalent for storing digital certificates and keys. Placing digital certificates and keys into a key database in the HFS is not an option.

9.8.1 Certificates in WebSphere and RACF

Figure 9-2 illustrates the portion of the WebSphere and RACF certificate data model that is used for establishing SSL connections. A good understanding of this model is essential for configuring and debugging SSL connection failures related to certificate setup for WebSphere and the other systems with which it interoperates.

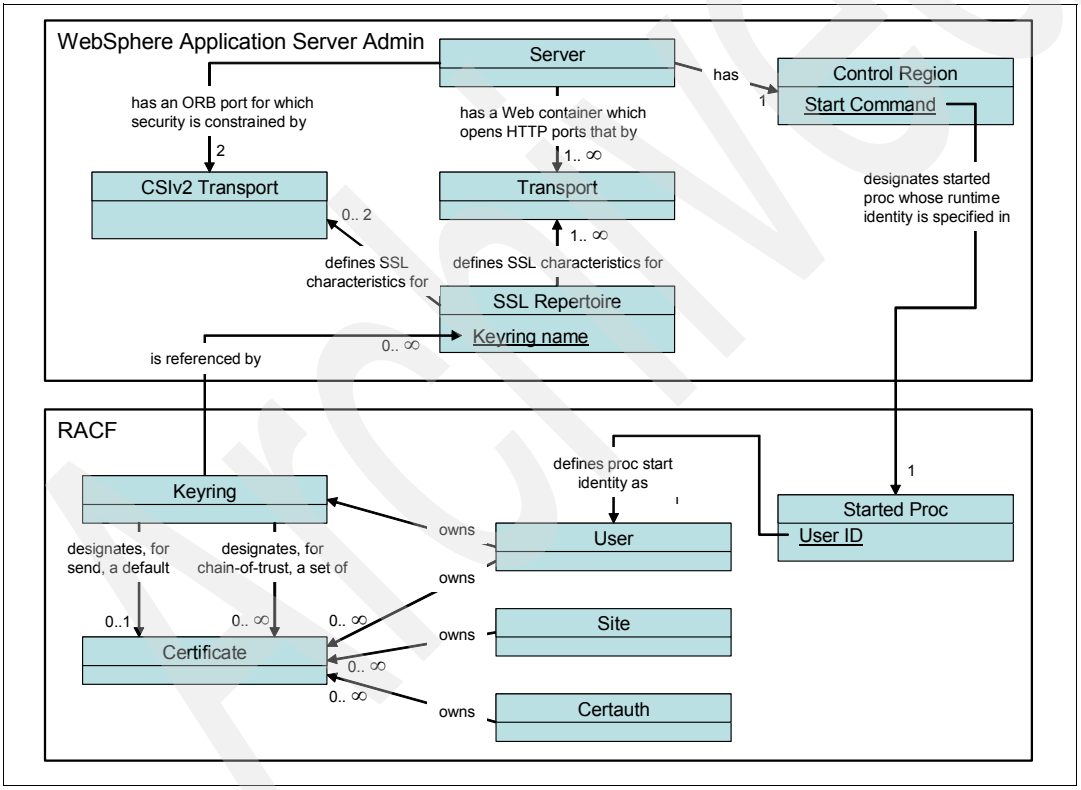


Figure 9-2 Certificate data model for WebSphere and RACF

The model does not include WebSphere and RACF concepts such as groups, certificate maps, and role mappings that support authorization after the SSL connection has been completed.

Here are some significant observations from the model:

- Listener ports secured by SSL are associated with the control region and with the user ID under which the control region runs.
- The keyring name and user ID fully define the relationship between WebSphere and RACF when locating the certificates that will be used to negotiate SSL connections.
- Each HTTPS and IIOP listener always serves a specific certificate associated with its repertoire's RACF keyring to clients; this is known in RACF as the "default" certificate for the keyring.

  The default certificate is also served as a client certificate for any remote SOAP or EJB services requested by applications running in the WebSphere containers.
- Each HTTPS and IIOP listener uses a list of "trusted" certificates found in its RACF keyring when evaluating the "chain of trust" when:
  - A client certificate is received by WebSphere acting as a server.
  - A server certificate is received by WebSphere acting as a client.
- It is possible to define multiple keyrings for a given server so that different default certificates can be used, for example, when WebSphere acts as a server, and when it acts as a client. For system SSL repertoires, the limitation is to use the same SAF keyring for all processes.

Figure 9-2 on page 268 also suggests a straightforward process when troubleshooting an SSL connection that results from an invalid certificate:

1. Identify the server based on the port.
2. Identify the user ID under which the server is executing.
3. Identify the SSL repertoire associated with the port and its keyring.

   For example, the keyring for an HTTPS port is identified from the administrative console by navigating to **servers** → **server** → **Web container** → **HTTP transports** → **Host**. Identify the SSL repertoire by name, and then click the **SSL configuration repertoires** link at the bottom of the page. From here, click the SSL repertoire named in the host definition, and the RACF keyring will be displayed in the field named "Key File Name."
4. Issue the following RACF command to list certificates on the keyring:

   `RACDCERT ID(`*`userid`*`) LISTRING(`*`ring-name`*`)`

   Where:

   `userid` is the user ID under which the server is executing.

   `ring-name` is the keyring name from the SSL repertoire.

5. Identify the label of the default certificate from the LISTRING command.
6. Issue the following RACF command to list the default certificate:

   `RACDCERT ID(`*`userid`*`) LIST(LABEL('`*`label-name`*`'))`

   Where:

   `userid` is the user ID under which the server is executing.

   `label-name` is the label of the default certificate.

   Data listed will include issuer's distinguished name, which is used to evaluate possible chain-of-trust issues, and validity dates, which could indicate that the certificate has expired.

9.8.2 SSL client certificate security for your WebSphere Application Server and clients

To define SSL client certificate security, you must first request signed certificates for your server and clients, and certificate authority (CA) certificates from the certificate authority that signed those certificates. The process of requesting certificates is beyond the scope of this book. For more information about requesting a certificate, see *z/OS System Secure Sockets Layer Programming*, SC24-5901.

After you have received signed certificates and CA certificates from the certificate authority, you must use RACF (or another SAF-based registry) to authorize the use of digital certificates, store certificates and keyrings in RACF, and define SSL security properties for your server through the administrative console.

Each client identified by a digital certificate must eventually be converted into a SAF user ID by the target WebSphere for z/OS server. If the client and server share the same RACF database, you do not have to do any additional configuration for this mapping. If the client and server do not share the same RACF database, you can configure the mapping by:

- Adding client certificates to the RACF database of the target server. This might be impractical in most cases.
- Mapping groups of clients into RACF identities using RACF certificate name filtering.
- Using a combination of the two.

The certificate arrangements involved in SSL client certificate authentication are:

- *For the client to authenticate the server*, the server (actually, the controller user ID) must possess a signed certificate created by a certificate authority (CA). The server passes the signed certificate to prove its identity to the client. The client must possess the CA certificate from the same certificate authority that issued the server's certificate. The client uses the CA certificate to verify that the server's certificate is authentic. Once verified, the client can be sure that messages are truly coming from that server, not someone else.
- *For the server to authenticate the client*, the client must possess a signed certificate created by a certificate authority (CA). The server must possess the CA2 certificate from the same certificate authority that issued the client's certificate. The server uses the CA certificate to verify that the client's certificate is authentic. Once verified, the server can be sure that messages are truly coming from that client, not someone else.

9.8.3 Define SSL security for servers and clients

This section includes the procedures you must follow to implement all SSL-based authentication mechanisms.

Using RACF to authorize the server to use digital certificates

SSL uses digital certificates and public/private keys. If your Application Server uses SSL, you must use RACF to store digital certificates and public/private keys for the user IDs under which the server controllers run.

If you plan to implement SSL client certificate support, you must also have certificate authority (CA) certificates from each certificate authority that verifies your client certificates. See *z/OS System Secure Sockets Layer Programming*, SC24-5901.

Perform the following steps authorizing the use of digital certificates:

1. For each server that uses SSL, create a keyring for that server's controller user ID.

Tip: From z/OS V1.4 and later, it is possible to share both keyrings and certificate private keys. The keyrings can be accessed by other users for certificate checking purposes. Even the use of the private keys might be allowed to other user IDs.

- To use someone else's keyring, specify the RACF user ID as the keyring prefix followed by a slash, for example: KEYFILE WEB001/my_keyring_name. The accessing user ID needs UPDATE access to IRR.DIGTCERT.LIST and IRR.DIGTCERT.LISTRING.
- To share the same certificate (and private key) you need to:
  - Generate (or add) the certificate under CERTAUTH or SITE.
  - Connect the certificate to the keyring with USAGE(PERSONAL).
  - Grant CONTROL access rights to IRR.DIGTCERT.GENCERT tor the accessing server's user ID.

2. Receive a certificate for all server's controllers from a CA (for example, VeriSign). To add the certificate to RACF that is saved in an MVS data set called WAS.BS01.CERT, issue the command:

```
RACDCERT ID(WDC1STU) ADD('WAS.BS01.CERT') WITHLABEL('CLUD1') +
PASSWORD('*****')
```

Important: The password must be specified only if the format of the certificate is PKCS#12.

3. Connect the signed certificate to the controller user ID's keyring and make the certificate the default certificate.
4. If you plan to have the server authenticate clients (SSL client certificate support), do the following tasks:
   - Receive each certificate authority (CA) certificate that verifies your client certificates. Give each CA certificate the CERTAUTH attribute.

     Example: Receive the CA certificate that will verify a client with RACF user ID CLIENT1. That certificate is in an MVS data set called WAS.CLIENT1.CA. Issue:

```
RACDCERT CERTAUTH ADD('WAS.CLIENT1.CA') WITHLABEL('CLIENT1 CA')
```

– Connect each client's certificate authority (CA) certificate to the controller user ID's keyring.

  Example: Connect the CLIENT1 CA certificate to the ring WASD1KEYRING owned by WDC1STU:

```
RACDCERT ID(WDC1STU) CONNECT(CERTAUTH LABEL('CLIENT1 CA') +
RING(WASD1KEYRING))
```

Setting up SSL security for clients

All clients must have access to the server's certificate authority (CA) certificate so they can authenticate the server during the SSL handshake. If you plan to implement SSL client certificate support, clients additionally must have their own certificates as the default certificate on their keyrings. If your clients are connecting to WebSphere for z/OS from WebSphere on workstations, you must import SSL certificates into the workstation system. For more information and instructions, see the I*BM WebSphere Information Center* at:

`http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp`

For SSL basic authentication and Kerberos, you must request a CA certificate from the same certificate authority that issued signed certificates for your controllers. If you plan to implement SSL client certificate support, you must additionally request a signed certificate for the client from a certificate authority.

Perform the following steps to authorize use of digital certificates by z/OS clients:

1. Create a keyring for a z/OS client with RACF user ID CLIENT1:

   ```
   RACDCERT ADDRING(CLI1RING) ID(CLIENT1)
   ```

2. Receive the server's certificate authority (CA) certificate and give it the CERTAUTH attribute.

   Example: You requested a CA certificate from VeriSign and they sent their certificate to you, which you stored in an MVS data set called WAS.CERT.CA with the command:

   ```
   RACDCERT ADD('WAS.CERT.CA') WITHLABEL('VERI CA') CERTAUTH
   ```

3. Connect the server's CA certificate to the client keyring:

   Example: Connect the VERI CA certificate to the CLI1RING keyring owned by CLIENT1:

   ```
   RACDCERT ID(CLIENT1) CONNECT(CERTAUTH LABEL('VERI CA') RING(CLI1RING))
   ```

4. In the client's environment file, code the security.sslkeyring= WebSphere variable to correspond to the client's keyring. For more information, see the WebSphere variables in the administrative console or the *Information Center*.

5. If you are implementing SSL client certificate support:
   - Receive the certificate for your client from the certificate authority.

     Example: You requested a certificate and the certificate authority returned a signed certificate which you stored in an MVS data set WAS.CLIENT1.CERT. Issue:

     ```
     RACDCERT ID(CLIENT1) ADD('WAS.CLIENT1.CERT') WITHLABEL('CLIENT1 CERT')
     PASSWORD('******')
     ```

   - Connect the client's signed certificate to the client user ID's keyring and make this certificate the DEFAULT certificate.

     Example: Connect the certificate labelled CLIENT1 CERT to the keyring CLI1RING owned by CLIENT1, by issuing:

     ```
     RACDCERT ID(CLIENT1) CONNECT(ID(CLIENT1) LABEL('CLIENT1 CERT') +
     RING(CLI1RING) DEFAULT)
     ```

Mapping client digital certificates to MVS user IDs

Each client that presents a digital certificate to authenticate its identity, but does not have an individual certificate registered with RACF on the target server's system or cell, must have a mapping to a valid MVS user ID. You can create this mapping by using RACF certificate name filters. You can create RACF certificate name filters based on either the client's or certificate issuer's distinguished name, as contained in the X.509 digital certificates.

Perform the following steps to set up certificate name filtering:

1. Define an MVS user ID for each user ID you associate with a certificate name filter.

   Consider assigning the PROTECTED and RESTRICTED attributes to each one. The PROTECTED attribute protects the user ID from being used to log on directly to the system and from being revoked through incorrect password attempts. The RESTRICTED attribute ensures that the user ID will not be used to access protected resources it is not explicitly authorized to access.

   ```
   AU WEBUSER DFLTGRP(MISC) OWNER(MISC) NOPASSWORD RESTRICTED
   ```

2. Create a certificate name filter.

   The following filter associates the user ID WEBUSER to any user presenting a certificate issued by VeriSign Class 1, who does not have an individual certificate registered with RACF on your system:

   ```
   RACDCERT ID(WEBUSER) MAP WITHLABEL('INTERNET OTHERS') +
   IDNFILTER('OU=VeriSign Class 1 Individual Subscriber.O=VeriSign,
   Inc.L=Internet')
   ```

   This filter is based on the issuer's name. You can create other filters based on the subject's name, or on combinations of the issuer's and subject's names.

For more information about certificate name filtering, see *z/OS: Security Server RACF Security Administrator's Guide*, SA22-7683.

9.8.4 Use certificates to set up HTTPS internal transport connections

An HTTPS internal transport can use server and client certificates to set up secure server-client connections for HTTPS application requests. The HTTPS internal transport enables you to set up client authentication using:

- Server certificates you have created and are administering, and for which you are your own certificate authority (CA)
- Client certificates signed by an internal CA

  Using an internal CA to sign your client certificates is independent of whether you used an internal or external CA to sign your server certificate.
- Cluster certificates signed by an external CA
- Client certificates that are signed by an external CA

  Using an external CA to sign your client certificates is independent of whether you used an internal or external CA to sign your server certificate. Before you can use a server certificate to set up secure HTTPS internal transport connections, you must:
  - Create or obtain a server certificate, if you do not already have one.
  - Create or obtain a CA certificate, if you do not already have one.
  - Create a controller keyring that is connected to your server certificate, and has this certificate as the default for this keyring.
  - Configure the HTTP transport from the administrative console.

If you also want to use a client certificate to set up secure HTTPS internal transport connections, you must perform the following additional tasks:

- Use the administrative console to specify that client certificates are allowed.
- Create or obtain a client certificate, if you do not already have one.

9.8.5 Set up secure HTTPS internal transport connections using a server certificate signed by an internal CA

Using SSL, WebSphere for z/OS enables you to set up your own certificate authority and administer your own certificates:

- Acting as your own certificate authority (CA) is recommended only for test environments and private intranets. With this method, you set up your own CA and sign certificates. You can optionally use client authentication to verify the identity of those accessing your controller.
- You must use System SSL to establish secure connections.
- You will issue the RACF command, RACDCERT, to create certificates and keyrings for your J2EE server.
- Use the Administrators dialog in the administrative console to give an MVS ID administrative authority over a Java server. See *WebSphere Application Server for z/OS Information Center* for more information.

Perform these steps to set up secure connections using self-signed CA certificates:

1. Create a self-signed CA certificate.
2. Create an SSL RACF controller keyring and connect your CA certificate to that keyring.
3. Create and sign your server certificate.
4. Connect your signed server certificate to the controller keyring.
5. Using the TSO/E OPUT command, copy the MVS data set containing your server certificate to your document root directory in the HFS.
6. Register the controller keyring with the Java server:
   a. Open the administrative console.
   b. Click **Servers** → **Application Servers** on the left navigation tree.
   c. Click the name of the server.
   d. On the Additional Properties menu of the Server panel, click **Web Container**.
   e. On the Additional Properties menu of the Web Container panel, click **HTTP Transport**.
   f. Click the **Host** you want to configure.
   g. Enter the Port number you want to bind.
   h. Select the check box to **Enable SSL**.
   i. Select the SSL alias in which you specified the controller keyring.

j. Click **OK**.

7. To verify that you can establish a secure connection with the controller, make sure the Java server is running, and then point your browser at the following URL:

   ```
   https://domain:port_number/directory/webapp_name
   ```

   Where:

domain	Is the domain where the Web application being requested resides.
port_number	Is the port number for SSL.
directory	Is the directory that contains the application.
webapp_name	Is the name of the certificate protected Web application being requested.

   Example:

   ```
   https://wtsc59oe.itso.ibm.com:9441/webap1/my.jsp
   ```

   The first time you enter this URL, you should receive a warning that the CA certificate is not trusted. You will then be prompted to accept the certificate for the current request and all future requests. If you accept the certificate, it will be added the browser's list of trusted CA certificates. The next time you enter this URL, you should not receive the warning.

8. Optionally, set up client authentication.

9.8.6 Set up secure HTTPS internal transport connections using client certificates signed by an internal CA

Using SSL, WebSphere for z/OS allows you to set up client authentication using client certificates signed by an internal CA. Using an internal CA to sign your client certificates is independent of whether you used an internal or external CA to sign your server certificate.

You must ensure that the internal CA that signs your client certificates is marked with a status of TRUST and that it is connected to your controller keyring. During the SSL handshake, the controller tells the client which CAs it trusts based on the trusted CAs in the controller keyring. The browser then searches its client certificates for ones issued by these CAs and allows the user to choose which client certificate to send to the controller.

Perform these steps to set up client authentication using client certificates signed by an internal CA:

1. Using the WebSphere for z/OS administrative console, verify that SSL client certificates are allowed:

   a. Select **Conversations** → the name of the conversation → **Cell** → cell name → **Java Servers** → the name of the appropriate Java server → **Modify**.

   b. In the properties form, select the SSL Client Certificates check box, if it is not already selected, to indicate that SSL client certificates are allowed.

2. Ensure that the CA certificate is in the client's browser, if this is a requirement for the client's browser. (Some browsers do not require the CA certificate to reside in the browser.) The following example assumes it is not already there.

   Example: Assuming that:

   – Your CA certificate must be added to the client's browser.

   – You are using the same CA certificate you created in section 9.8.5, "Set up secure HTTPS internal transport connections using a server certificate signed by an internal CA" on page 276 to sign client certificates, as well as your server certificate. (This certificate resides in the data set WAS.CACERT.DERBIN.)

   Download the CA certificate:

   a. Make sure the Java server is running with SSL initialized.

   b. Open a browser.

   c. Enter the following URL:

   ```
   https://wtsc59oe.itso.ibm.com:9441/was.cacert.derbin
   ```

   Follow the browser prompts to install the CA certificate. Newer versions of the Netscape Navigator and Microsoft Internet Explorer browsers automatically start a wizard to help you install the certificate. If you use Microsoft Internet Explorer, you might need to open the file rather than saving it to disk to start the wizard. See the online help in your browser or browser documentation for additional information. Generally for Netscape, if you receive a window asking if you would like to accept the CA to certify network sites, electronic mail (e-mail) users, and software developers, do so.

3. Create the client certificate and associate it with a RACF user ID. Assuming your CA certificate was added to the list of trusted CAs in your browser, you can now generate a client certificate under a RACF user ID. This enables the client certificate to be used to authenticate the user ID.

   Example: In this example, we create a client certificate with label CERT FOR CLIENT1 signed by the internal CA using label WAS Z/OS CA. The client

certificate will be created under user ID CLIENT1. To create the client certificate signed by the internal CA, issue the RACF command:

```
RADCERT ID(CLIENT1) GENCERT SUBJECTSDN(CN('CLIENT1')O('IBM') + OU('ITSO')
L('POUGHKEEPSIE') SP('NEW YORK')C('US')) WITHLABEL('CERT + FOR
CLIENT1')SIGNWITH(CERTAUTH LABEL('WAS Z/OS CA'))
```

The client certificate will be created with status TRUST. Trust indicates that the client certificate can be used to authenticate the user ID CLIENT1.

4. Add the signed client certificate to the client's browser. To perform this step, you must:
   - Export the client certificate to an MVS data set.

     Example: To export the client certificate to a data set so that the client certificate can be added to the client's browser, issue the following command:

     ```
     RACDCERT ID(CLIENT1) EXPORT(LABEL('CERT FOR CLIENT1')) +
     DSN('CLIENT1.P12') FORMAT(PKCS12DER) PASSWORD('*****')
     ```

     Where:

CLIENT1	Is the user ID associated with the client certificate being exported.
CERT FOR CLIENT1	Is the label of the client certificate.
'**CLIENT1.P12**'	Is the data set that will contain the client certificate.
PKCS12DER	Indicates that the client certificate and private key are DER encoded when saved to the data set.
*****	Is the password associated with the encrypted client certificate. You will be required to provide this password when you import the client certificate into the browser. The password is case sensitive.

   - FTP the client certificate to the client's workstation.

     Example: This example shows how to use the FTP command to transfer the PKCS12 data set containing the signed client certificate to the client's workstation. The following steps are performed on the workstation:

     i. Enter the FTP command and the host name or IP address of the controller.

     ii. When prompted, enter your user ID and password.

     iii. Enter `bin` to transfer the data set in binary format.

     iv. Transfer the data set to the workstation by entering:

     ```
     get 'CLIENT1.P12' client1.p12
     ```

     v. Enter `quit` or `bye` to exit.

– Load the client certificate into the client's browser.

  Example: This example shows how to load the PKCS12 file into the Netscape Communicator browser:

  i. Start the browser.

  ii. To access the security information, click **Communicator** → **Tools** → **Security** → **Info**.

  iii. Under Certificates, click **Yours**.

  iv. Click **Import a Certificate**. You might need to scroll down to see this option.

  v. Highlight the PKCS12 file.

  vi. Click **Open**, and enter the case-sensitive password protecting the file.

  vii. Click **OK**. The following message will be displayed:

  ```
  Your certificates have been successfully imported.
  ```

  viii. Click **OK**. You should be able to see the following certificate label in the window: `These are your certificates`. You might need to scroll down to find the label.

  > **Note:** On Netscape Navigator browser versions prior to Netscape 4.6.1, there might be a problem displaying the label. For example, the label name might appear as ????@????.

5. Verify that the client can use the client certificate to access a protected page. To verify that the client can establish a secure connection with a protected page, make sure the Java server is running, and point your browser to the following URL:

   ```
   https://wtsc59oe.itso.ibm.com:9441/webap1/my.jsp
   ```

When prompted by the browser, select the label for the client certificate. If the setup is correct, you will be able to view the protected page without prompts for a user ID and password.

9.8.7 Set up secure HTTPS internal transport connections using server certificates signed by an external CA

Using SSL, WebSphere for z/OS allows you to use an external Commercial CA to sign your server certificate.

The following example uses the base server controller started task user ID WDC1STU. Perform these steps to set up secure connections using server certificates signed by an external CA:

1. Create the controller keyring.
2. Create a server certificate request. In this step you will:
   a. Create a self-signed certificate in order to establish your common name (host name) and public-private key pair.
   b. Generate a server certificate request from the self-signed certificate and save it to a data set:

      ```
      RACDCERT ID(WDC1STU) GENREQ(LABEL('CLUD1')) DSN('WAS.BS01.CERTREQ')
      ```

      The self-signed certificate that you create will contain the controller's common name and public-private key pair. This information is required in order to generate the certificate request and obtain a server certificate signed by your external CA.
3. Transfer the server certificate request to your workstation:

   Example: To use the File Transfer Protocol (FTP) command to transfer the WAS.BS01.CERTREQ data set containing the server certificate request to your workstation, perform the following steps from your workstation:
   a. From a DOS prompt line, enter an FTP command specifying either the host name or IP address of the controller.
   b. When prompted, enter your user ID and password.
   c. Change to the directory where you put the data set containing the server certificate request, WAS.BS01.CERTREQ. For example, if the data set resides in the directory USER1, enter: `cd 'USER1'`.
   d. Transfer the file in ASCII format to the workstation by entering:

      ```
      get was.bs01.certreq
      ```

   e. Enter `quit` or `bye` to exit the FTP command process.
4. Send the request to a CA to be signed, using the CA's instructions for sending certificate requests and receiving signed server certificates.
5. Ensure that your CA certificate is in the RACF list of CA certificates and marked with a status of TRUST. These conditions must be met before you can receive the CA-signed certificate into your controller keyring. In this step you must:
   – Check whether your external CA is in the list of CAs in RACF and whether it is marked with a status of TRUST.

– Connect the CA certificate to your WASD1KEYRING keyring. To check the list of CAs, issue the following RACF command:

```
RACDCERT CERTAUTH LIST
```

After receiving the CA certificate, add it to RACF with TRUST status:

```
RACDCERT CERTAUTH ADD('WAS.CERT.CA') TRUST WITHLABEL('CA CERT FOR WDC1STU')
```

If your CA is in the list of CAs in RACF, but has a current status of NOTRUST, change the status to TRUST. For example, issue the following command:

```
RACDCERT CERTAUTH ALTER(LABEL('CA CERT FOR WDC1STU')) TRUST
```

Now that your CA certificate is in the RACF list of CA certificates and has a status of TRUST, connect the CA certificate to your WASD1KEYRING keyring.

6. Add your server certificate to the controller keyring. In this step, you will:
   – Alter the server certificate if necessary to make it look like the example.
   – Put the server certificate in the MVS data set WAS.BS01.CERT.
   – Add the server certificate to RACF and associate it with the user ID WDC1STU.

   Before you can add the signed server certificate to your controller keyring, you must put this certificate in an MVS data set. The certificate data set you create in this example is WAS.BS01.CERT. Alter the certificate data set if necessary. Include the BEGIN CERTIFICATE and END CERTIFICATE lines and all data in between as shown in the following example. If your certificate contains additional information before BEGIN CERTIFICATE or after END CERTIFICATE, remove all of the extraneous information in the file.

Example 9-1 Sample exported certificate

```
-----BEGIN CERTIFICATE-----
MIIBODCCATkCBDV8PgswDQYJKoZIhvcNAQECBQAwcjELMAkGA1UEBhMCVVMxDTALBgNVBAgTBE4uQy4
xDDAKBgNVBAcTA1JUUDEMMAoGA1UEChMDSUJNMRcwFQYDVQQLEw5XZWJzZXJ2ZXIgVGVzdDEfMBOGA1
UEAxMWbXZzMTY3LnJhbGVpZ2guaWJtLmNvbTAaFws5ODA2MDgxOTM5WhcLOTkwNjA4MTkzOVowNDELM
AkGA1UEBhMCVVMxDDAKBgNVBAoTAOlCTTEXMBUGA1UEAxMOcGtjczEwLmlibS5jb2OwXDANBgkqhkiG
9wOBAQEFAANLADBIAkEA1IYGldVmnKAI8hJQGTO74oXTDOTb+jFN8wkPqc+DVhYix1fjh/sbiuDZF66
BMh5hnHfJr75633CgjW1OEpIDOwIDAQABMAOGCSqGSIb3DQEBAgUAA4GBAF1KVppAM7Gh2F9BBIY/jP
MFlRp8+HAAVkK29Q4DxeF2FrTzQutKmO8duCWvxnJo4pgl5Uj29DSAsrX8mULfczyuZwVVXiCGnhNO3
pYj8bbQjoOedqQ7hYsRl3P4C72IyRwtWUukfVgwdoOmWXyEclx7eT5jsW4weVEqWvuht8j
-----END CERTIFICATE-----
```

Example 9-1 assumes that you received the server certificate on your workstation, and need to FTP this certificate, in ASCII format, to a z/OS or OS/390 data set. The following steps are performed on the workstation:

a. Enter the FTP command and the host name or IP address of the controller.

b. When prompted, enter your user ID and password.

c. Transfer the file to the z/OS or OS/390 data set that will be created on execution of the PUT command by entering:

```
put was.bs01.cert 'WAS.BS01.CERT'
```

d. Type `quit` or `bye` to exit.

Issue the following RACF command to add the server certificate signed by your external CA to RACF and associate it with the WDC1STU ID. In doing so, you will replace the self-signed certificate created in Step 2:

```
RACDCERT ID(WDC1STU) ADD('WAS.BS01.CERT') WITHLABEL('CERT FOR WDC1STU')
```

7. Connect your signed server certificate that is now in RACF to your WASD1KEYRING keyring and make this certificate the default certificate in the keyring. For example, issue the following RACF command:

```
RACDCERT ID(WDC1STU) CONNECT(ID(WDC1STU) LABEL('CERT FOR WDC1STU') +
RING(WASD1KEYRING) DEFAULT)
```

8. Register your controller keyring with the Application server. We can see through the administrative console what this configuration looks like in Figure 9-3 on page 284.

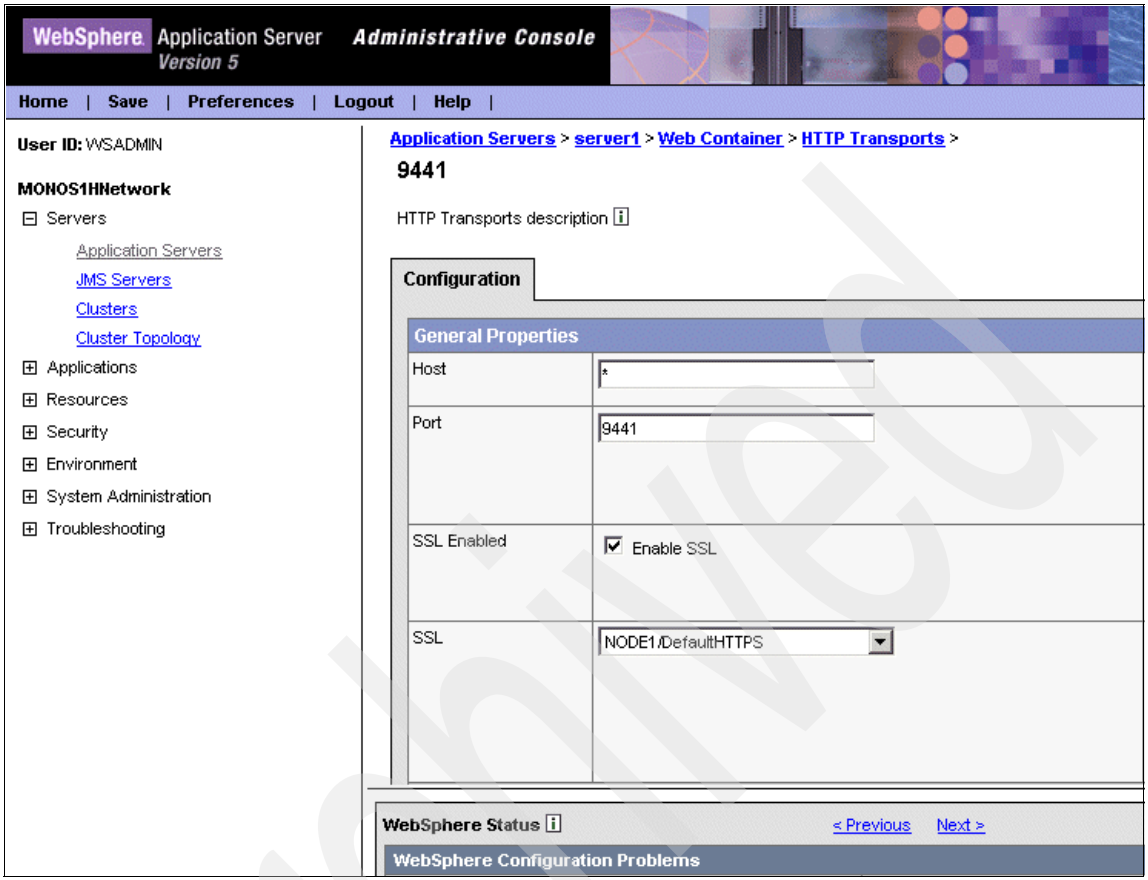


Figure 9-3 Setting the keyring

Perform the following steps:

a. Open the administrative console.

b. Click **Servers** → **Application Servers** on the left navigation tree.

c. Click the name of the server.

d. On the Additional Properties menu of the Server panel, click **Web Container**.

e. On the Additional Properties menu of the Web Container panel, click **HTTP Transport**.

f. Click the **Host** you want to configure.

g. Enter the **Port** number you want to bind.

h. Select the check box to **Enable SSL**.

i. Select the SSL alias in which you specified the controller keyring.

j. Click **OK**.

9. Verify that you can establish a secure connection with the controller. To do that, make sure the Java server is running, and point your browser at the same URL as in 9.8.6, “Set up secure HTTPS internal transport connections using client certificates signed by an internal CA” on page 277.

10. Optionally, set up client authentication.

9.8.8 Set up secure HTTPS internal transport connections using client certificates signed by an external CA

WebSphere for z/OS allows you to set up client authentication using client certificates that are signed by an external CA. Using an external CA to sign your client certificates is independent of whether you used an internal or external CA to sign your server certificate. However, before using this example, there are a few things you must do first:

- Ensure that the external CA that signs your client certificates is marked with a status of TRUST and that it is connected to your controller keyring. During the SSL handshake, the controller tells the client which CAs it trusts based on the trusted CAs in the controller keyring. The browser then searches its client certificates for ones issued by these CAs and allows the user to choose which client certificate to send to the controller.
- If you created and signed your server certificate using the steps in the section 9.8.7, “Set up secure HTTPS internal transport connections using server certificates signed by an external CA” on page 280, you can use the external CA defined in that example for the external CA in this example.
- If you created and signed your server certificate using the steps in the section 9.8.5, “Set up secure HTTPS internal transport connections using a server certificate signed by an internal CA” on page 276, ensure that your CA certificate is in RACF and marked with a status of TRUST, and that the external CA is connected to the controller keyring. Perform these steps to set up client authentication using client certificates signed by an external CA:

  a. Verify that SSL client certificates are allowed:

     i. Open the administrative console.

     ii. Click **Security** → **Authentication Protocols** → **CSIv2 Inbound Authentication** in the left navigation tree.

     iii. In the General Properties section of the Configuration Tab for the CSIv2 Inbound Authentication panel, check to see if either Supported or Required is selected for Client Certificate Authentication.

b. Ensure that the CA certificate is in the client's browser. Browsers vary as to whether they require the CA certificate to be in the browser. This example assumes you will ensure that your CA certificate is added to your browser if it is not already there.

   If you are using the same external CA certificate to sign client certificates that you used to sign your server certificate, clients might have already loaded the CA certificate into their browsers in step 5 of section 9.8.7, "Set up secure HTTPS internal transport connections using server certificates signed by an external CA" on page 280. If they have not, they should contact the external CA to obtain the CA certificate.

c. Obtain the client certificate. Follow the external CA's instructions for obtaining the signed client certificate and loading it into your browser.

d. Map a client certificate to a RACF user ID. RACF maps client certificates that are in various formats, including PKCS12, binary, and base-64 encoded ASCII. Some browsers might be able to output the client certificate in these formats or other formats. If, for instance, either the binary or base-64 encoded ASCII format is put out, the resulting file would contain the client certificate without the private key. If the PKCS12 format is put out, the resulting file would contain the private key (which RACF doesn't use) and the client certificate. If a browser does not produce the client certificate in the format that you want, contact the signing authority to obtain the client certificate in the desired format.

   The format of the client certificate in RACF can be different from the format of the client certificate in the browser. Because some browsers require client certificates in PKCS12 format, we will map a PKCS12 formatted certificate to a RACF user ID. You can export the client certificate from the browser so that it can be input to RACF. You can use the following RACF options to map a client certificate to RACF when the client certificate is created with some method other than RACF commands or PKI Services:

   - Certificate Name Filtering function

     This function is available for OS/390 Release 10 and later.

   - Automatic registration of digital certificates on the Web

     The Auto registration Web application enables a client to automatically register a certificate with the controller.

   - Using ISPF panels or the RACDCERT command

     These two options take the same inputs, but you use panels with ISPF and a command line with RACDCERT. This example shows how to use the RACDCERT command.

In this step, you will:

i. FTP the PKCS12 formatted client certificate from the workstation to the HFS on your z/OS system:

   Enter the FTP command and the host name or IP address of the controller, for example, **ftp wtsc59oe.itso.ibm.com**.

   When prompted, enter your user ID and password.

   Change to the directory where you will place the client certificate. For example: cd /ibm/security/user1.

   Enter `bin` to transfer the file in binary format.

   Transfer the file to the HFS by entering:

   ```
   put client1.p12
   ```

   Type `quit` or `bye` to exit.

ii. Copy the PKCS12 formatted client certificate from the HFS to an MVS data set. You must store client certificates on MVS in variable block (VB) format. The client certificates in this example are in an HFS directory. Use the TSO/E OGET command to move the certificate file from the HFS directory into an MVS sequential data set. If you move the client certificate into a new data set, the OGET command creates a VB sequential data set by default.

   You must use the OGET command to move the client certificate into the MVS sequential data set; exporting it from the browser to FTP it directly into the MVS data set does not work because the certificate file is not in the correct format:

   ```
   oget '/ibm/security/user1/client1.p12' 'client1.p12' binary
   ```

iii. Issue RACF commands to associate the client certificate with a RACF user ID exactly as in “Setting up SSL security for clients” on page 273.

   Verify that the client certificate is associated with a user ID that has been defined to RACF, by issuing the following RACDCERT command and specifying the user ID in the ID field:

   ```
   RACDCERT ID(CLIENT1) LIST
   ```

iv. To verify that a client can establish a secure connection with the controller, make sure the Java server is running, and point your browser at the same URL as in “Setting up SSL security for clients” on page 273.

When prompted by the browser, select the label for the client certificate. If the setup is correct, you will be able to view the protected page without prompts for a user ID and password.

Part 4

WebSphere Application Server for z/OS security infrastructure

In this part, we describe in detail the security features of a WebSphere Application Server for z/OS Version 5 system.

We start with an overview of the architecture of WebSphere Application Server for z/OS Version 5 and then describe what you should do to prepare the operating system security for WebSphere.

Next, we discuss the choice of user registries because in WebSphere Application Server V5 there is now support for LPAP and custom user registries as well as for SAF-based user registries such as RACF.

We then describe the security controls that protect the WebSphere administrative console. Because many security choices are made using the WebSphere administrative console, you need to take particular care about protecting it.

Then, we describe how to enable authentication and authorization for both the Web container and EJB container, including how you would configure some authentication in a proxy server such as Tivoli Access Manager.

10

WebSphere Application Server runtime security

IBM WebSphere Application Server for z/OS *can* use a System Authorization Facility (SAF)-based security subsystem as its registry for authentication and authorization purposes as described in Chapter 12, “Local operating system registries” on page 343.

IBM WebSphere Application Server for z/OS *must* use an SAF-based security subsystem for the protection of its runtime, therefore the WebSphere implementation on z/OS is deeply integrated into the SAF-based security subsystem. This chapter is designed to introduce you to WebSphere Application Server for z/OS security and enable you to make early planning decisions about system security.

In 10.4, “SAF naming standards and conventions” on page 297, we discuss the factors you should consider when planning your security implementation. This planning needs to be done before attempting to configure WebSphere Application Server for z/OS and OS/390. Then in 10.10, “The WebSphere ISPF installation security dialog” on page 308, we describe how you input your security choices into the WebSphere Application Server ISPF installation dialog to create a security domain for WebSphere Application Server for z/OS and OS/390. After your WebSphere Application Server has been configured and started, in 15.7, “Enabling global security” on page 489, we explain how to enable security *inside* WebSphere Application Server.

WebSphere supports access to resources by clients and servers in a network, so part of your security strategy should be to determine how to control access to these resources and prevent inadvertent or malicious destruction of the system or data. These are the aspects in the distributed network that you must consider:

- You must authorize servers to the base operating system services in z/OS. These services include SAF security, database management, and transaction management.
  - For the servers, you must distinguish between controllers and servants. Controllers run authorized system code, so they are trusted. Servants run application code and are given access to resources, so you should carefully consider the authorizations you give servants.
  - You must also distinguish between the level of authority runtime servers and your own application servers have. For example, the node agent server needs the authority to start other servers, while your own application servers do not need this authority.
- If you need to protect resources, identifying who accesses those resources is critical. Thus, any security system requires client (user) identification, also known as authentication. In a distributed network supported by WebSphere for z/OS, resources can be accessed as follows:
  - Within the same system from a WebSphere server
  - Within the same sysplex from a WebSphere server
  - Remote z/OS or OS/390 systems from a WebSphere server
  - Heterogeneous systems, such as WebSphere on distributed platforms, CICS, or other Java-compliant systems
  - WebSphere clients remote or local
- Additionally, clients can request a service that requires a server to forward the request to another server. In such cases, the system must handle delegation, the availability of the client identity for use by intermediate servers and target servers.
- Finally, in a distributed network, how do you ensure that messages being passed are confidential and have not been tampered? How do you ensure that clients are who they claim to be? How do you map network identities to z/OS or OS/390 identities? These issues are addressed by the following support in WebSphere for z/OS:
  - The use of SSL and digital certificates
  - Kerberos
  - Common Secure Interoperability Version 2 (CSIv2) transport security

10.1 WebSphere address space concepts

As described in 1.1, “WebSphere Application Server for z/OS Version 5 infrastructure overview and terminology” on page 4, the WebSphere Application Server for z/OS and OS/390 consists of a variable number of z/OS started tasks. It consists not only of a control region (CR) or controller and server regions (SR) or servants, but also of started tasks that comprise the WebSphere Application Server for z/OS and OS/390 infrastructure.

All of these address spaces, and the IBM HTTP Server, are protected by the standard System Authorization Facility (SAF) and Authorized Program Facility (APF) capabilities of the operating system.

Figure 10-1 shows the structure of an application server. The same structure applies to the naming and system management servers. A controller represents the external gateway into an application server. It is the endpoint of all TCP/IP connections with the WebSphere Application Server. The controller contains no application program code, but only operating system and middleware code provided by IBM. This code is considered authorized and has been installed because it is considered trusted. IBM operating system and middleware code is committed to protecting the system’s integrity.

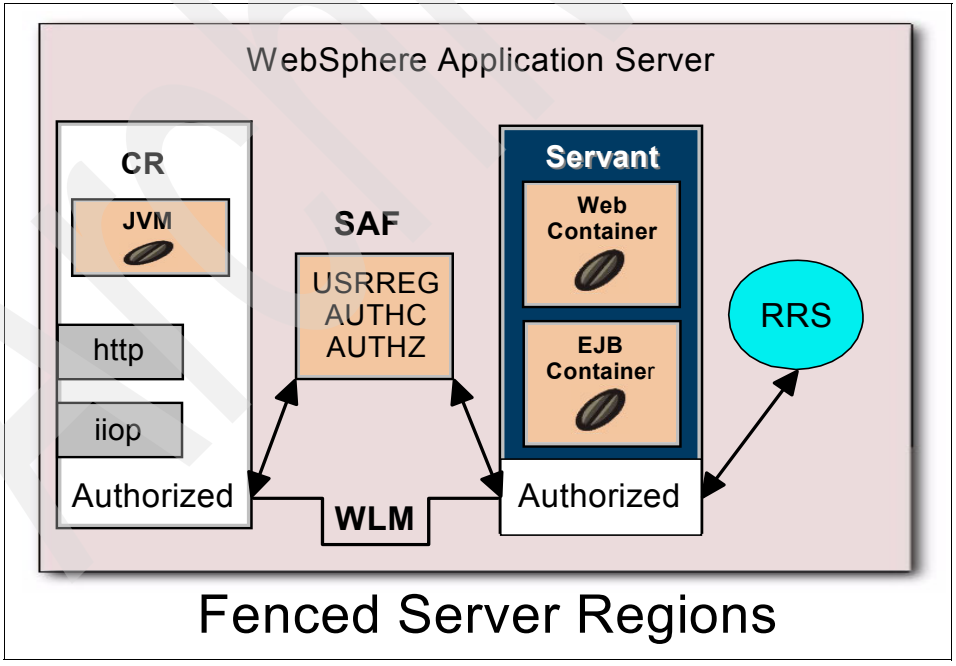


Figure 10-1 Fenced server regions

The controller is responsible for examining incoming requests. It will not use any of the data received from the network in a way that could compromise the system's integrity without validating it. The controller's own data is in a system storage key (2), and is therefore prevented by the hardware from being overwritten by user data received over the network (for example, by a buffer overflow attack).

After examining an incoming request, the controller places the request on a queue managed by the Workload Manager (WLM). The request can then be processed by a servant. There is no way for a network connection to pass requests to a servant without having them examined by the trusted middleware code in the controller.

The servant contains user-written and other application code. This code executes in a non-authorized state and user storage key (8), making it impossible for this code to access privileged instructions or sensitive operating system data.

Besides application code, there is some WebSphere Application Server code that runs in the servant in an authorized state. This is limited to making System Authorization Facility (SAF) calls, removing work from the WLM queue, and registering with Resource Recovery Services (RRS). The authorized code ensures that it is being called in a legitimate way, such that applications cannot exploit these calls for nefarious purposes.

Based on installation configuration options, the controller and servant can be assigned to run under user IDs with limited capabilities. There is never a need for a controller or servant to run with root UID(0) or superuser authority.

10.2 WebSphere Application Server SAF integration

As you can see from the foregoing discussion, WebSphere Application Server for z/OS and OS/390 makes full use of the unique security and integrity capabilities of z/OS such as APF, SAF, and zSeries storage keys. When properly configured, WebSphere Application Server for z/OS and OS/390 runs in a manner that provides excellent security and integrity. Figure 10-2 on page 295 shows the big picture of how the WebSphere Application Server V5.0 environment can be securely integrated into the most powerful security infrastructure: z/OS Security Server.

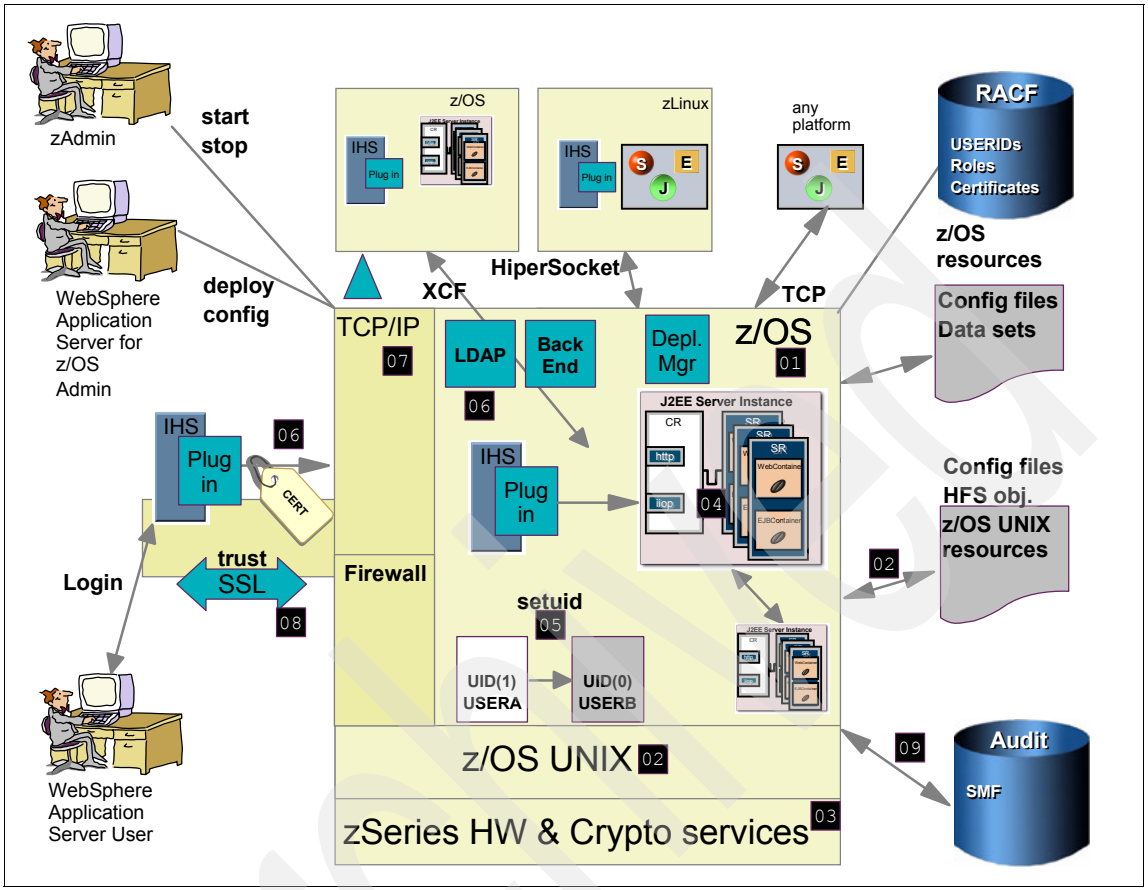


Figure 10-2 WebSphere integration into a z/OS security infrastructure

Shown with many of the components is a number that is used in Table 10-1 on page 296. This table identifies the component and where you should look for further information.WebSphere Application Server V5.0.

> **Note:** This chapter assumes that your z/OS environment has a proper "baseline" security enabled. That is, your critical data sets (such as APF, link list, LPA, SMF, and PARMLIB) are properly protected.

The table has four columns: An "Item" that references Figure 10-2, Protected Resource (what needs to be controlled), Protection Mechanism (a brief description of how the resource is controlled), and Further Information Available In (which points you to a source of more information).

Table 10-1 Protected resources and protection mechanisms

Item	Protected resource	Protection mechanism	Further information available in
1	z/OS Infrastructure	Basic Setup, USER and Group concepts, Naming conventions	10.3, “Basic RACF setup for z/OS” on page 296.
2	z/OS UNIX	Basic Setup and structures	10.5.1, “z/OS UNIX level security” on page 301.
3	ICSF	ICSF started task, ICSP data sets, use of ICSF services	9.6.1, “RACF protection for ICSF” on page 263 and 9.6.2, “RACF setup to secure OCSF and OCEP” on page 264.
4	WebSphere Runtime	Basic Setup	10.5, “Setting up RACF controls for products related to WebSphere” on page 301.
5	IBM HTTP Server	HTTP server runtime	10.8, “RACF protection for IBM HTTP Server for z/OS” on page 305.
6	LDAP Server	Transport and access	10.6, “RACF protection for LDAP servers on z/OS” on page 304.
7	TCP/IP	Port, Intrusion Detection, Firewall, syslogd	Not covered in this book.
8	SSL	Transport security	9.1, “Cryptographic support” on page 240.
9	Auditing	SMF data examination	Chapter 19, “WebSphere Application Server logging and auditing” on page 663.

10.3 Basic RACF setup for z/OS

Every z/OS and OS/390 installation has unique security requirements, naming standards, and various degrees of properly architected RACF databases. WebSphere provides some default definitions for servers, resources and other classes. If you can use a “sandbox” system without incorporating these into your production security administration, life will be easier. (Experiment with your first few WebSphere systems before incorporating the WebSphere security definitions into your overall corporate security scheme.)

10.4 SAF naming standards and conventions

> **Tip:** Good naming standards for all resources in the z/OS environment are a necessary condition, but not a sufficient condition to have neat and tidy, easy to manage RACF databases. The sufficient condition is the permanent, everyday enforcement of these standards without compromise.

We assume that you have naming standards for several types of groups and user IDs. Also we assume that you have a policy for ownership of resources

10.4.1 RACF group structure

An important condition for a good RACF database is its group structure. It is outside the scope of this book to deal with the best way to design such a structure. However, it is sufficient to recommend a very simple structure, as shown in Figure 10-2 on page 295.

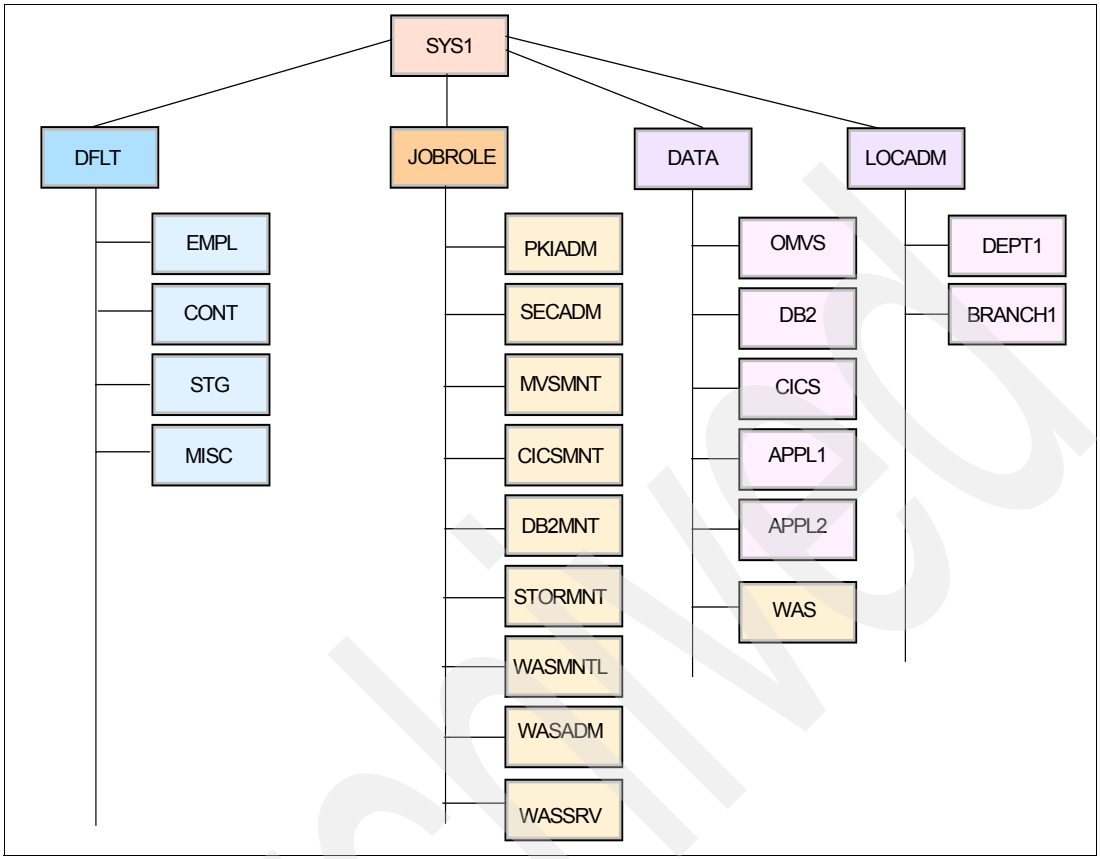


Figure 10-3 Very simple RACF group structure example

In the example in Figure 10-3, the subgroups of group SYS1 are:

DFLT The subgroups are all default groups.

JOBROLE The subgroups are all jobrole groups.

DATA The subgroups are all groups matching high-level qualifiers for data sets.

LOCADM The subgroups are all groups for decentralized RACF administration, containing users with authority of GROUP-SPECIAL.

Note: You should not connect any user IDs to these four groups.

Default group for started task user IDs

As part of the jobrole group structure implementation, we created a RACF default group for all started task user IDs on our systems. You might have such a group with names STC, STCG, and STCGRP or something else. Our preferred name is STG and the command to create it is:

```
AG STG SUPGROUP(DFLT) OWNER(DFLT)
```

Note: The group DFLT is a subgroup of group SYS1 and has as subgroups all default groups; for example, default group EMPL for all employees, default group CONT for all contractors and temps, default group STG for all started task user IDs, and default group MISC for miscellaneous machine user IDs. Default groups should never be permitted to any resource.

Jobrole groups

In this redbook, we introduce the concept of a *jobrole* RACF group as a necessary condition for a properly architected RACF database. The user IDs connected to such a group need the same set of accesses to various resources in order to do their everyday job.

For example, the group of MVS systems programmers, let's call it MVSMNT, needs access ALTER to all operating system data sets and all data sets related to software products. However, they do not need even READ access to data sets storing data from financial, human resources, or other business applications. When an MVS systems programmer joins or leaves the team, the RACF administrator needs only to connect or remove him to or from the group MVSMNT. In order for such a concept to be implemented, a lot of preliminary work is needed such as interviews with representatives of all areas of the enterprise, logical assessment by a RACF architect, and security policy reviews followed by the difficult task of actually reengineering the whole RACF group structure.

If you have applications running in your z/OS UNIX environment, such as WebSphere, PKI server, and any Web servers, you have to extend the jobrole concept into the z/OS UNIX environment in order to work out which jobrole groups will be the *owning groups* of HFS files and their level of access.

We assume that a job role group structure is implemented at least in the Information Technology department of your organization and we will refer to the z/OS UNIX system groups as follows:

- UNIX system programmers group as USSMNT
- Security administrators group as SECADM
- Decentralized security administrators groups, all subgroups of LOCADM, as LOCADM

10.4.2 Creating machine user IDs

It is necessary to create user IDs for various subsystems, programs, and procedures used in a z/OS environment. These user IDs do not belong to human users and we will refer to them as machine user IDs.

Started task user IDs

Started task user IDs (STUs) are necessary to run procedures (programs for various subsystems and products). In the z/OS UNIX environment, the STUs are called daemons. We assume that your site has a naming standard for started task user IDs (STUs). Our preferred naming standard is *productnameSTU*.

In the framework of the jobrole concept for group structure, we recommend that STUs are never connected to job role groups, but permitted individually to RACF profiles, instead. However, as an exception user IDs running CICS regions and WebSphere servers can be connected to jobrole groups.

In your z/OS UNIX environment, you might need to make some STUs or their jobrole groups owners of HFS files and directories, or use ACLs to permit them to files and directories with a suitable level of access.

Miscellaneous user IDs

Here we refer to various miscellaneous user IDs such as batch submission IDs, CICS default users, console IDs, and surrogate user IDs. They should have their own default group, for example MISC, and should be permitted individually to RACF profiles.

10.4.3 System data set profiles

Most installations have for many years used well established naming standards for data sets. Usually all IBM system software resides in data sets prefixed with SYS1, while in-house modifications to software, as well as system data, reside in data sets prefixed with SYS2, or other user-defined names. Our preferred prefix is SYSU.

10.4.4 Ownership

Although the RACF OWNER operand might not have any significance in RACF except when a decentralized administration is in place, we recommend that all resources have a meaningful OWNER (should always be a RACF group), for example:

- User IDs are owned by their default groups.
- System data set profiles are owned by MVSMNT.
- CICS software data sets are owned by CICS systems programmers.

- Application data sets are owned by respective business groups.
- Security profiles (BPX, IRR in the RACF FACILITY class) are owned by SECADM.

10.5 Setting up RACF controls for products related to WebSphere

Although there are no prerequisite products for WebSphere Version 5, you might need RACF environments for z/OS UNIX-level security, Public Key Infrastructure Services (PKI Services), Integrated Cryptographic Services Facility (ICSF), front-end z/OS HTTP server, Lightweight Directory Access Protocol (LDAP) server, and DB2.

10.5.1 z/OS UNIX level security

We recommend that you have z/OS level of UNIX security on your system. More information about the two possible levels of security (UNIX level of security and z/OS level of security) is available in Chapter 27 of *z/OS UNIX System Services Planning*, GA22-7800.

The RACF controls necessary to establish the z/OS UNIX level of security are:

- Program control
- Daemon and server control

Program control

For program control you have to create profile ** in class PROGRAM and then ADDMEM the following libraries:

```
RDEF PROGRAM ** OWNER(MVSMNT) UACC(READ)
RALT PROGRAM ** ADDMEM('CEE.SCEERUN'//NOPADCHK +
'CBC.SCLBDLL'//NOPADCHK +
'GLD.SGLDLNK'//NOPADCHK +
'GSK.SGSKLOAD'//NOPADCHK +
'SYS1.CSSLIB'//NOPADCHK +
'TCPIP.SEZALOAD'//NOPADCHK +
'SYS1.LINKLIB'//NOPADCHK +
'CSF.SCSFMOD0'//NOPADCHK +
'CSF.SCSFMOD1'//NOPADCHK +
'BBO.SBBOLOAD'//NOPADCHK)
```

Library BBO.SBBOLOAD contains the WebSphere programs and you must ADDMEM it to ** PROGRAM if you plan not to load it into LPA.

Library GSK.SGSKLOAD contains the system SSL programs. You must ADDMEM it to ** PROGRAM and also add it to LNKLIST.

Tip: You might want to convert from an existing profile * to profile ** to prevent the output of RLIST PROGRAM * ALL displaying information about all profiles created in class PROGRAM, to displaying only the content of profile **, instead. But be careful: you have to ADDMEM to profile ** all already ADDMEMed libraries to profile * (if additional to the ones above) before issuing RDEL PROGRAM * and SETR WHEN(PROGRAM) REFRESH.

Also, you might want to ADDMEM the following libraries to profile ** in class PROGRAM:

- If using the IBM Resource Measurement Facility (RMF™), *hlq*.SERBLINK
- If using DB2, *hlq*.SDSNLOAD and *hlq*.SDSNEXIT
- If using WebSphere MQ, *hlq*.SESQLINK
- If using Run-Time Library Services (RTLS), *hlq*.SCEERTLS
- If you are obtaining an IEATDUMP by setting the SysDumpName directive and setting the Recovery directive to Msg/Dump, Normal, or Full, *hlq*.MIGLIB

Note: You have to replace the italics above with HLQs that are used at your site, most likely SYS1.

After you have defined a ** generic profile in the PROGRAM class, you should restrict access to the ICHDSM00 (RACF DSMON program) and the IRRDPI00 (RACF Dynamic Parse program). Users who have EXECUTE authority or better authority to these programs in library SYS1.LINKLIB are able to execute these programs. To remove this access, define these profiles explicitly and grant access only to those users who you want to use the utility.

Example 10-1 ICHDSM00 and IRRDPI00 RACF definitions

```
RDEFINE PROGRAM ICHDSM00 UACC(NONE) ADDMEM('SYS1.LINKLIB'//NOPADCHK)
RDEFINE PROGRAM IRRDPI00 UACC(NONE) ADDMEN('SYS1.LINKLIB'//NOPADCHK)

PERMIT ICHDSM00 CLASS(PROGRAM) ID(users who are allowed to us DSMON)
PERMIT ICHDSM00 CLASS(PROGRAM) ID(users who are allowed to use IRRDPI00)
```

If you failed to make a load library program controlled, you will get the following message in the SDSF LOG (the example given is for a DB2 load library).

Example 10-2 Message when a load library is not program controlled

```
ICH420I PROGRAM DSNAOCLI FROM LIBRARY DB2H7.SDSNLOAD CAUSED THE
ENVIRONMENT TO BECOME UNCONTROLLED.
BPXP014I ENVIRONMENT MUST BE CONTROLLED FOR DAEMON (BPX.DAEMON)
PROCESSING.
BPXM023I (ITSOLDP) 338
GLD5002A The __passwd() function failed; not loaded from a program
controlled library.
```

To rectify, just ADDMEM the library to profile ** in class PROGRAM.

Setting up program control for HFS programs only

If you want only HFS files to be checked for program control, and do not want programs loaded from MVS libraries to be checked, you can set up profile BPX.DAEMON.HFSCTL in class FACILITY. However, doing this weakens some of the security provided by the BPX.DAEMON profile, because user IDs with daemon authority are allowed to load uncontrolled programs from MVS libraries into their address space.

Profile BPX.DAEMON must be defined before issuing the commands:

```
RDEF FACILITY BPX.DAEMON.HFSCTL OWNER(SECADM) UACC(NONE)
PERMIT BPX.DAEMON.HFSCTL CLASS(FACILITY) ID(daemon's user IDs) ACCESS(READ)
```

Daemon and server control

The command to create a profile for restricting your daemons from being able to change their identity is:

```
RDEF FACILITY BPX.DAEMON UACC(NONE) OWNER(SECADM)
```

Place on the access list only daemons who are allowed to change their identity:

```
PE BPX.DAEMON CL(FACILITY) ID(daemons' user IDs) ACC(READ)
```

The command to create a profile to set the scope of z/OS resources that the server can access when acting as a surrogate for its clients is:

```
RDEF FACILITY BPX.SERVER UACC(NONE) OWNER(SECADM)
```

Then, place on the access list only server IDs that can act as surrogates:

```
PE BPX.SERVER CL(FACILITY) ID(servers' and daemons' user IDs) ACC(READ)
PE BPX.SERVER CL(FACILITY) ID(servers' and daemons' user IDs) ACC(UPDATE)
```

READ Means that both the server ID and the RACF ID of the client must be authorized to resources and the client must supply a password.

UPDATE Means that the server acts as surrogate of the client, that is, uses the identity and access granted to the client without the client's password being specified.

10.6 RACF protection for LDAP servers on z/OS

The RACF setup for the LDAP server consists of the following commands:

- Started task user ID

```
ADDUSER LDAPSTU DFLTGRP(STG) NAME(LDAP Started Task User') +
OMVS(AUTOUID) HOME('/') PROG('/bin/sh'))
```

- Profile in class STARTED

```
RDEF STARTED LDAP*.** STDATA(USER(LDAPSTU) GROUP(STG))
SETR RACLIST(STARTED) REFR
```

- Access to DB2 from TSO and BATCH

```
PE DB2subsystem.BATCH ID(LDAPSTU) ACC(R)
```

- Daemon and server control

```
PE BPX.DAEMON CL(FACILITY) ID(LDAPSTU) ACC(R)
PE BPX.SERVER CL(FACILITY) ID(LDAPSTU) ACC(U)
SETR RACLIST(FACILITY) REFR
```

- Access to OCSF

```
PE CDS.** CL(FACILITY) ID(LDAPSTU) ACC(R)
```

- Access to profiles in class CSFSERV

```
PE ** CL(CSFSERV) ID(LDAPSTU) ACC(R)
```

This access is needed only if ICSF is active.

10.7 RACF protection for DB2

DB2 resources can be accessed by the WebSphere infrastructure and by WebSphere applications. To add operating system security to these resources, you can use the RACF DSNR resource class to protect DB2 resources. There are three functional areas in RACF to consider regarding protection for DB2:

- The RACF DSNR class controls access to the DB2 subsystems. If the DSNR class is active, WebSphere for z/OS controllers and servants need access to the *db2_ssn*.RRSAF profiles, where *db2_ssn* is your DB2 subsystem name. If a controller or servant does not have access, that region will not initialize.

- DB2 identification and signon exits (DSN3@ATH and DSN3@SGN) assign authorization IDs. If you want to use secondary authorization IDs (RACF group names), you must replace the default exits with these two sample routines.
- WebSphere for z/OS does not support the protection of DB2 objects through the DSNX@XAC exit. To protect DB2 objects, you must use GRANT statements.

10.8 RACF protection for IBM HTTP Server for z/OS

The RACF environment for IBM HTTP Server for z/OS consists of:

- Started task user ID

  The command to create a started user ID for your Web server is:

```
ADDUSER WEBSTU DFLTGRP(STG) OMVS(UID(0) HOME('/usr/lpp/internet')
PROGRAM('/bin/sh'))
```

  Note: If you want to define your Web server user ID with a nonzero UID and according to your naming standard for STUs, do not forget to give this user ID appropriate permissions to directories and files:

```
SETFACL -m u:WEBSTU:r-x /usr/lpp/internet/sbin/httpd_V5R3M0
SETFACL -m u:WEBSTU:r-x /web/pki1/httpd.conf
SETFACL -m u:WEBSTU:rwx /web/pki1/logs
SETFACL -m d:u:WEBSTU:rwx /web/pki1/logs
SETFACL -m f:u:WEBSTU:rwx /web/pki1/logs
SETFACL -m u:WEBSTU:rw- /web/pki1/httpd-pid
```

  The Web server's nonzero user ID must be given read/execute access to the security DLLs, read/write access to the key database file, and read access to the stash file.

- Profile in class STARTED

  The command to create a profile in class STARTED for your Web server procedure is:

```
RDEF STARTED WEB*.** STDATA(USER(WEBSTU) GROUP(STG))
```

- Daemon and server control

  The commands to control your Web server started task user ID are:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(WEBSTU) ACCESS(READ)
PERMIT BPX.SERVER CLASS(FACILITY) ID(WEBSTU) ACCESS(READ)
```

- Access to profiles in class CSFSERV

  This access is needed only if ICSF is active. For more information, refer to 9.6.1, “RACF protection for ICSF” on page 263.

```
PE CSF* CL(CSFSERV) ID(WEBSTU) ACC(R)
```

10.9 Summary of RACF general resource classes used by WebSphere

This section contains a concise list of the RACF general resource classes used by WebSphere Application Server for z/OS Version 5.

Table 10-2 RACF general resource classes used by WebSphere

RACF class	Description
APPL	Controls access to applications. Kerberos is set up as an application, required so the users can enter the **kpasswd** command.
CBIND	Controls the client’s ability to bind to the server. With WebSphere we need to control access to the server.
DATASET	Access to data sets.
DIGTCERT	Contains digital certificates and information related to them.
DSNR	Controls access to DB2 subsystems.
EJBROLE	Member class for Enterprise Java Beans authorization roles. The APPLDATA field in an EJBROLE profile defines the target Java identity when running in RUNAS ROLE mode.
FACILITY	Miscellaneous uses: Profiles are defined in this class so that resource managers can check users’ access to the profiles when the users take some action. Here is where we place profiles for Digital Certificate, DCE and Kerberos, plus UNIX System Services profiles, BPX.DAEMON.
GEJBROLE	Grouping class for Enterprise Java Beans authorization roles.
KERBLINK	Mapping class for user identities of local and foreign principals. Used in Kerberos we can map a unique RACF user ID to each foreign principal.
LOGSTRM	Controls which applications can access the system logger resources.

RACF class	Description
OPERCMDS	Controls who can issue operator commands for JES and MVS. Required to allow WebSphere started task, to stop and start other WebSphere servers. Allows to restrict operational control of a specific application server to certain user IDs.
PROGRAM	With the NOPADS option se, it defines the trusted programs (load modules). Any security-relevant program that changes the identity of its address space or thread from USS needs to be declared here.
PTKTDATA	PassTicket key class enables the security administrator to associate a RACF-secured signon secret key with a particular mainframe application that uses RACF for user authentication. Required for Kerberos setup, but no profile needs to be created in this class, just activated.
REALM	Used to define the local and foreign realms. Required for Kerberos setup, as you have to define your local realm to this class. You also use this class to define foreign realms, such as another z/OS, OS/390, and Microsoft Windows.
RRSFDATA	Used to control RACF remote sharing facility functions. Required as part of Kerberos setup.
SERVAUTH	Contains profiles that are used by servers to check a client's authorization to use the server or to use the resources managed by the server. You can use this class to protect TCP/IP ports. If you are using this class, you must give WebSphere and Kerberos access.
SERVER	Controls the server's ability to register with the daemon. In WebSphere we use this class to control whether a servant can call authorized programs in the controller
SOMDOBJS	Controls the client's ability to invoke the method in class. Only used for Web applications running in WebSphere Application Server 3.5 for z/OS and OS/390.
STARTED	Used in preference to the started procedures table to assign an identity during the processing of an MVS START command. For example, WebSphere, Kerberos are defined as started tasks in this profile.
SURROGAT	Whether surrogate submission or logon is allowed, and if allowed, which user IDs can act as surrogates. SURROGAT is used here in conjunction with BPX.SRV.* profiles in the SURROGAT class to allow security context switches for unauthenticated user IDs.

10.10 The WebSphere ISPF installation security dialog

WebSphere Application Server for z/OS and OS/390 Service Level W502000 introduced a change to the ISPF installation dialog, which means that now you first define your WebSphere Application Server for z/OS and OS/390 security requirements, and then later, you define the type of application server you want to create. The security definitions can be used when subsequent WebSphere Application Server servers are defined that need to share the same security domain. For example, you will probably want all application servers in a cluster, including the deployment manager node, to share the same RACF user IDs and groups for the WebSphere Application Server controller and servant region address spaces. They will also want to share the same SSL keyring, because you will want to manage the cluster as though it were a single system.

The security customization panel of the installation dialog opens, as shown in Example 10-3.

Example 10-3 ISPF installation dialog: Security panels

```
-----------------     WebSphere for z/OS Customization      ------------------
 Option  ===>

 Security Customization (1 of 1)

    Specify the following for your WebSphere customization.
    Press ENTER to continue.

 Installation dependent WebSphere Security Customization definitions

   SSL Customization

    WebSphere Certificate Authority Keylabel:  A1TestCertAuth
    RACF Keyring Name.....................:  A1WASKeyring

   Use RACF Authorization and Delegation.....:  Y
   Support Passtickets for z/SAS authentication:  Y
       Passticket Profile name...............:  CBS390
       Passticket KEYMASK value..............:  1212121212121212

   Enable WebSphere Application Server to authenticate
    RACF users via WebSphere login tokens.....:  Y

   Generate RACF commands for above..........:  Y

 Installation dependant z/OS Security Customization definitions

   Authorize Servers to APPL profile.............:  Y
   Sample OPERCMDS definitions for WebSphere Servers:  Y
```

Note: The ISPF security dialog might look slightly different on your system, depending on the service level you have applied to WebSphere. Service Level W502000, for example, rearranged the sequence of panels such that the security configuration, including security domain information, is presented before configuration of any nodes.

10.10.1 Security Customization panel settings

Pressing PF1 when displaying the Security Customization panel displays a help screen that provides some information about the options presented. Here are some additional observations:

WebSphere Certification Authority Keylabel

When global security is enabled, WebSphere requires the use of SSL to access the administrative console. Additionally, SSL might be required by individual applications. To prepare for this, the security customization panel generates RACF RACDCERT commands to generate the certificate and keyring used for SSL. In a production environment, the certificate should be signed by a Certification Authority. This setting allows you to specify the label of the Certification Authority that will be used to sign the server certificate. It is assumed that the Certification Authority certificate does not already exist, so it will be created for you. The commands will not be created unless you specify `Y` for the value of “Generate RACF commands for the above,” further down the panel.

RACF keyring name

Provide the name of the keyring used by the WebSphere controller region for SSL security. Commands to create the keyring will be generated, along with the server certificate. The server certificate will be signed by the Certificate Authority keylabel specified by WebSphere Certificate Authority Keylabel. In addition, the commercial CA certificates that are installed by default in RACF will be connected to the server’s keyring. The commands will not be created unless you specify `Y` for the value of “Generate RACF commands for the above,” further down the panel.

Use RACF authorization and delegation (pre-5.1)

Specifying `Y` here will generate the RACF EJBROLE class profiles used for the four administrative console roles (administrator, monitor, configurator, and operator) and the JNDI administrative roles CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The commands will not be created unless you specify `Y` for the value of “Generate RACF commands for the above,” further down the panel.

Note: Specifying `Y` here generates the commands to create the RACF EJBROLE profiles, but it does not cause WebSphere to use them. That is controlled by the properties com.ibm.security.saf.authorization and com.ibm.security.saf.delegation, specified using the administrative console on the custom properties page of the local OS user registry under the security settings.

Use SAF EJBROLE profiles to enforce J2EE roles

Specifying `Y` here causes RACF statements to be generated, which create EJB roles, and causes both com.ibm.security.SAF.authorization and com.ibm.security.SAF.delegation to be set to true. `N` results in no EJB roles being defined and causes the two variables to be set to false. The two variables control the following functions:

- com.ibm.security.SAF.authorization:
  - True causes the SAF user registry EJBROLE profiles to be used for authorization, and com.ibm.security.SAF.delegation to be used in determining run-as role mapping.
  - False causes the deployment descriptor security mappings to be used and the caller ID to be used instead of run-as (com.ibm.security.SAF.delegation is ignored).
- com.ibm.security.SAF.delegation (if authorization=true):
  - True causes the APPLDATA field on the EJBROLE profile to be used to determine the z/OS identity for a given run-as role.
  - False causes caller ID to be used.

In summary, specifying `Y` causes RACF EJBROLE profiles to be defined by the installation jobs and the local OS registry to be configured for authorization and J2EE run-as role mapping. `N` causes the deployment descriptor binding files to be used.

Important: If you enter N in this field, but choose later to enable RACF authorization by setting com.ibm.security.SAF.authorization to true, you will not be able to access the administrative console without defining the required EJBROLE profiles inside RACF; the required RACF commands are not generated by the customization dialogs when N is entered. Here is the template for the RACF commands you will need:

```
SETROPTS CLASSACT(EJBROLE)

RDEFINE EJBROLE [domain_prefix.]administrator UACC(NONE)
RDEFINE EJBROLE [domain_prefix.]monitor       UACC(NONE)
RDEFINE EJBROLE [domain_prefix.]configurator UACC(NONE)
RDEFINE EJBROLE [domain_prefix.]operator     UACC(NONE)

PERMIT [domain_prefix.]administrator CLASS(EJBROLE)
ID(default_WS_CFG_group) ACCESS(READ)
PERMIT [domain_prefix.]monitor CLASS(EJBROLE) ID(default_WS_CFG_group)
ACCESS(READ)
PERMIT [domain_prefix.]configurator CLASS(EJBROLE) ID(
default_WS_CFG_group) ACCESS(READ)
PERMIT [domain_prefix.]operator CLASS(EJBROLE) ID(default_WS_CFG_group)
ACCESS(READ)

RDEFINE EJBROLE [domain_prefix.]CosNamingRead UACC(READ)
RDEFINE EJBROLE [domain_prefix.]CosNamingWrite UACC(READ)
RDEFINE EJBROLE [domain_prefix.]CosNamingCreate UACC(READ)
RDEFINE EJBROLE [domain_prefix.]CosNamingDelete UACC(READ)

PERMIT [domain_prefix.]CosNamingRead CLASS(EJBROLE)
ID(default_WS_unauth_userid) ACCESS(READ)

SETROPTS RACLIST(EJBROLE) REFRESH
```

Where:

domain_prefix is the name of your security domain if you have defined it in the administrative console (**Security** → **Global Security** → **Custom properties** → **security.zOS.domainName**).

default_WS_unauth_userid is the local default identity (**Security** → **Global Security** → **z/OS Security Options** → **Local Identity**).

default_WS_CFG_group is the WebSphere configuration group name (check the group name owning your configuration files in the HFS).

Support PassTickets for zSAS authentication

Specifying `Y` here, along with the PassTicket profile name and the PassTicket KEYMASK value, will generate a RACF PTKTDATA class profile that is used by RACF to validate PassTickets used to log on to WebSphere. The commands will not be created unless you specify `Y` for the value of “Generate RACF commands for the above”, further down the panel.

PassTicket profile name

This is where the name of the PassTicket profile is specified. The name is not a variable, but must be CBS390.

PassTicket KEYMASK value

This is where the PassTicket KEYMASK value is specified.

Enable WebSphere Application Server to authenticate RACF users through WebSphere login tokens

Specifying `Y` or `N` here does not appear to make any difference in the RACF commands generated.

Generate RACF commands

Specifying `N` here will prevent the generation of RACF commands to create certificates or keyrings, EJBROLE class profiles or PassTickets, regardless of the values specified further up the panel. Specifying `Y` will cause the generation of the RACF commands based upon the values specified.

Authorize servers to APPL profile

Specifying `Y` here will cause the generation of RACF commands that define the APPL class profile CBS390, with UACC of NONE. If defined, the profile controls which users have access to WebSphere. This is similar to the use of APPL class profiles by CICS, TSO, and so on. As the APPL class profile CBS390 is arguably redundant to the use of role-based authorization using EJBROLE class profiles, you should consider whether setting the UACC of the CBS390 profile to READ is more appropriate to your environment. The WebSphere Application Server V5.02 level and later enables you to change the default of CBS390 to the one that is defined in your security domain.

Sample OPERCMDS definitions for WebSphere servers

Specifying `Y` here generates sample commands to define profiles in the OPERCMDS class. The profiles protect the START, STOP, MODIFY, CANCEL, and FORCE commands, and permit the user IDs of the WebSphere daemon, controller, and servant regions to issue the commands, if you are not currently protecting MVS commands with the OPERCMDS class.

Note: You should be aware that defining a few commands will prevent the majority of your users from issuing MVS commands. Proper planning is required before activating the OPERCMDS class. The commands generated here are only a sample. We recommend that you enable OPERCMDS independently from the WebSphere installation process and not to generate the commands here.

10.10.2 ISPF customization dialogs for RACF command generation

The installation of WebSphere comes with ISPF customization dialogs in which you supply variables representing names of RRS and IXGLOGR data sets, CTRACE data sets, HFS data set, short and long names for HFS directories, job names, names of PROCs for SYS1.PROCLIB, TCP/IP ports, RACF user IDs and groups, and so on. The dialogs produce REXX EXECs in an output data set called xxx.DATA and TSO jobs in an output data set called xxx.CNTL. The variables are saved to and loaded from a data set called CNFSAVE. For naming conventions refer to 6.3, “Naming conventions” on page 112.

Note: You should always save your variables into a data set of the type CNFSAVE. You have to always LOAD your variables (by pressing L in the ISPF dialogs menu) before generating a new set of CNTL and DATA data sets.

As part of the ISPF customization dialog, a REXX EXEC called BBOWBRAC residing in the DATA data set is fed with RACF variables such as user IDs, groups, UIDs and GIDs, which then are wrapped into RACF commands generated into another dynamically built REXX EXEC called BBOWBRAK. The latter is executed with a job BBOCBRAK from the CNTL data set.

Running BBOWBRAK will allow RACF people or people assigned the task to install WebSphere not familiar even with the RACDCERT command, to create a RACF environment and then deal with further customizations.

Although BBOWBRAC is a valuable tool, we undertook a different approach in this book: First explain how to create a RACF environment for WebSphere and then run a straightforward job through which the new product seamlessly fits into the organization’s RACF database.

This approach is based on the assumption that most customers will have their own sets of naming standards, procedures and RACF design style already in place, which might not accommodate the BBOWBRAC-generated RACF commands.

10.10.3 RACF definitions, naming conventions, and considerations for WebSphere environments on z/OS

In this section we discuss the creation of RACF environments for base servers and Network Deployment cells with deployment manager (DM). There are three sets of RACF entities for WebSphere:

- Common set: the Same for base server and DM
- Base server set: Different for each base server
- Deployment manager set: Different for each Network Deployment cell

Persons with any of the following authorizations can run the jobs generated by the ISPF dialogs:

- User IDs with UID=0
- User IDs permitted with access CONTROL to profile SUPERUSER.FILESYS.** in class UNIXPRIV

We recommend the second option.

User IDs with SYSTEM SPECIAL should run the RACF commands in EXEC BBOWBRAK.

Important: The concept of WebSphere SAF security domains was introduced to the code with WebSphere Application Version 5.02. If you are on a 5.02 level or later, and if you decide to create security domains, some of your RACF conventions will look different from the ones described here. We recommend that you review the documentation that comes with the V5.02 product upgrade.

10.10.4 Define RACF groups for WebSphere

In this section, we define all RACF groups for WebSphere.

WebSphere system programmers group

You might also want to appoint one or more UNIX system programmers to install WebSphere with SMP/E and to run the ISPF customization dialogs, except the RACF job. Our suggested name for such a jobrole group is WASMNT. The command to create specific jobrole groups for WebSphere system programmers is:

```
AG  WASMNT SUPGROUP(JOBROLE) OWNER(JOBROLE) OMVS(AUTOGID)
CO (userid1 userid2) GROUP(WASMNT)
```

WebSphere configuration group

A group called configuration group in *WebSphere Application Server for z/OS Version 5 Installation and Customization*, GA22-7910, consists of all started task user IDs running the WebSphere procedures and must be the owning group permitted with rwx to all WebSphere runtime directories and files. The following command defines this group (we appended STG to the product name WAS to emphasize that this jobrole group consists of started task user IDs only):

```
AG WASSTG SUPGROUP(JOBROLE) OWNER(JOBROLE) OMVS(AUTOGID)
```

WebSphere administrators group

WebSphere administrators play a very powerful role in your organization. The decisions they make when establishing various security mechanisms for authentication and authorization determine who will access your WebSphere applications and what privileges they will have when doing so. We recommend appointing as WebSphere administrators one or more of your RACF administrators. If you have PKI Services installed on z/OS, you might appoint someone from the PKI administrators group as well. Alternatively, all of your RACF and PKI administrators from jobrole groups SECADM and PKIADM might be doing WebSphere administration.

Define your jobrole group for WebSphere administrators with the command:

```
AG  WASADM SUPGROUP(JOBROLE) OWNER(JOBROLE) OMVS(AUTOGID)
CO (userid1 userid2) GROUP(WASADM)
```

Note: You might already have selected a range of GIDs for your jobrole groups. If you have such a range and want to use AUTOGID, modify profile BPX.NEXT.USER in class FACILITY with your ranges for started task UIDs and jobrole GIDs as follows:

```
RDEF FACILITY BPX.NEXT.USER APPLDATA('your UID range st task/your GID
range jobrole')
```

Irrespective of whether you use AUTOGID and AUTOUID or just assign the next available GID in your range, you have to make sure that GIDs are unique.

Both WASSTG and WASADM need to have rwx access to HFS directories and files, belonging to WebSphere. To avoid creation of access and default ACLs for group WASADM, we will connect human administrators and started task user IDs for WebSphere to group WASADM and abandon group WASSTG in our further presentation.

If your policy is not to have a mixture of human user IDs and machine user IDs connected to a jobrole group, you need to make WASSTG the owning group of

the WebSphere runtime HFS by providing its name as WebSphere configuration group in the ISPF dialog and create ACLs for your mounting point.

Example 10-4 Access, directory, and file default ACLs for group WASADM

```
setfacl -m group:WASADM:rwx /WebSphere/BS01/
setfacl -m d:g:WASADM:rwx /WebSphere/BS01/
setfacl -m f:g:WASADM:rwx /WebSphere/BS01/
```

Servants group

The command to create the jobrole group for servants started task user IDs is:

```
AG  WASSRV SUPGROUP(JOBROLE) OWNER(JOBROLE) OMVS(AUTOGID)
```

This group will be authorized to resources outside WebSphere, for example DB2.

10.10.5 Define RACF user IDs for the WebSphere infrastructure

We will present commands defining user IDs for base servers and managed servers (members of a Network Deployment cell). Our examples are based on base server number 1 and an Network Deployment cell called PROD for managed servers.

Daemon-started task user IDs

The command to create the user ID known as daemon that runs the Locate service procedure for base servers is:

```
AU WDD1STU NAME('WS DAEMON BS01') DFLTGRP(STG) OWNER(STG) NOPASSWORD +
OMVS(AUTOUID HOME('/tmp/') PROG('/bin/sh'))
```

The command to create the user ID that runs the Locate service procedure for cell PROD is:

```
AU WPDDMSTU NAME('WS DAEMON PROD') DFLTGRP(STG) OWNER(STG) NOPASSWORD +
OMVS(AUTOUID HOME('/tmp/') PROG('/bin/sh'))
```

Both user IDs should be connected to the configuration group WASADM:

```
CONNECT (WDD1STU WPDDMSTU) GROUP(WASADM)
```

Controller-started task user IDs

The command to create the user ID that runs the Locate service procedure ( ) for base servers is:

```
AU WDC1STU NAME('WS CONTROLLER BS01') DFLTGRP(STG) OWNER(STG) + NOPASSWORD
OMVS(AUTOUID HOME('/tmp/') PROG('/bin/sh'))
```

The command to create the user ID that runs the Application Server controller for cell PROD is:

```
AU WPCDMSTU NAME('WS CONTROLLER PROD') DFLTGRP(STG) OWNER(STG) + NOPASSWORD
OMVS(AUTOUID HOME('/tmp/') PROG('/bin/sh'))
```

Both user IDs should be connected to the configuration group WASADM:

```
CONNECT (WDC1STU WPCDMSTU) GROUP(WASADM)
```

Servant-started task user IDs

The command to create the user ID who runs the servant procedure for base servers is:

```
AU WDS1STU NAME('WS SERVANT BS01') DFLTGRP(STG) OWNER(STG) NOPASSWORD +
OMVS(AUTOUID HOME('/tmp/') PROG('/bin/sh'))
```

The command to create the user ID that runs the servant procedure for cell PROD is:

```
AU WPSDMSTU NAME('WS SERVANT PROD') DFLTGRP(STG) OWNER(STG) + NOPASSWORD
OMVS(AUTOUID HOME('/tmp/') PROG('/bin/sh'))
```

Both user IDs should be connected to the servants group WASSRV:

```
CONNECT (WDS1STU WPSDMSTU) GROUP(WASSRV)
```

CTRACE-started task user ID

The command to create the user ID that runs the CTRACE procedure ( ) for any server is:

```
AU CTRCSTU NAME('WAS TRACE WRITER') DFLTGRP(STG) OWNER(STG) NOPASSWORD
```

WebSphere administrator user ID

This user ID (default name WSADMIN) has no password interval and should be defined with the NOEXPIRED password operand. The commands are:

```
AU WASADMIN DFLTGRP(MISC) NAME('WAS ADMINISTRATOR')OMVS(AUTOUID +
HOME('/tmp') PROG('/bin/sh'))
PW USER(WSADMIN) NOINTERVAL
ALU WSADMIN PASSWORD(*******) NOEXPIRED
```

This user ID should be connected to the configuration group WASADM:

```
CONNECT WSADMIN GROUP(WASADM)
```

We highly recommend that you change its default password WSADMIN to a password of your choice.

Unauthenticated user ID

The unauthenticated user ID should be created with the RESTRICTED and NOPASSWORD attributes, because it will act on behalf of unauthenticated clients. The command to create the user ID is:

```
AU WASDFTU DFLTGRP(MISC) OMVS(AUTOUID HOME('/tmp') PROG('/bin/sh')) +
NAME('WAS DEFAULT USER') RESTRICTED NOPASSWORD
```

Because this user has the RESTRICTED attribute, it must be explicitly permitted with READ to several resources having UACC(READ) or ID(*) with access READ. The most obvious are: profile ** in class PROGRAM, a few profiles starting with IRR.DIGTCERT in class FACILITY, and profile ** in class APPL (by the way, if it has UACC(READ), this is equivalent to not having this profile at all).

10.10.6 Protect WebSphere-related data sets with RACF

We recommend that data sets needed for WebSphere be named yourprefix.WAS.

Alternatively, if you do not use prefixing, use an HLQ of WAS. We used WAS as HLQ.

WebSphere data sets for ISPF dialogs

To define RACF profiles for your WebSphere MVS data sets, issue the following commands:

```
AG WAS SUPGROUP(DATA) OWNER(DATA)
DEF ALIAS (NAME('WAS') REL('user catalog name'))
AD 'WAS.**'  OWNER (WASMNT)
```

This profile protects the CNFSAVE, CNTL, and DATA data sets.

The candidates for access to these data sets are jobrole groups involved with the installation of WebSphere using ISPF dialogs. Depending on your available skill sets, you can permit your WebSphere system programmers with ALTER, your WebSphere administrators with UPDATE or ALTER, and your RACF administrators with READ:

```
PE 'WAS.**' ID(WASMNT) ACC(ALTER)
PE 'WAS.**' ID(WASADM) ACC(UPDATE)
PE 'WAS.**' ID(SECADM) ACC(READ)
```

WebSphere HFS data sets

To define a RACF profile for the WebSphere HFS data sets, issue the following commands (we assume you have group OMVS and profile OMVS.**):

```
AD 'OMVS.WAS.**'  OWNER (WASMNT)
```

```
PE 'OMVS.WAS.**'  ID(WASMNT) ACC(ALTER)
```

WebSphere CTRACE data set

To define a RACF profile for the WebSphere CTRACE data sets, issue the following command:

```
AD 'WAS.**.CTRACE'  OWNER (WASMNT)
PE 'WAS.**.CTRACE'  ID(WASMNT) ACC(ALTER)
PE 'WAS.**.CTRACE'  ID(CTRCSTU) ACC(UPDATE)
```

WebSphere log stream data sets

To define a RACF profile for the WebSphere log stream data sets, issue the following command:

```
AD 'WAS.**.LOG' OWNER (WASMNT) UACC(READ)
PE 'WAS.**.LOG' ID(WASMNT) ACC(ALTER)
PE 'WAS.**.LOG' ID(IXGLSTU) ACC(UPDATE)
PE 'WAS.**.LOG' ID(WDC1STU WPCDMSTU WDS1STU WPSDMSTU) ACC(UPDATE)
```

Where IXGLSTU is the started task user ID for your MVS IXGLOGR procedure, and we assume that it is not running as TRUSTED.

WebSphere data set for RACF commands

You might want to create a data set in which to keep successive versions of member BBOWBRAK with RACF commands generated from the ISPF dialogs. Your RACF administrators can edit the commands according to your policies and then run them, in close cooperation with the WebSphere installers.

To define a RACF profile for the WebSphere IXGLOGR data sets, issue the following command:

```
AD 'WAS.RACF.** OWNER (SECADM) UACC(NONE)
PE 'WAS.RACF.** ID(SECADM WASADM) ACC(ALTER)
PE 'WAS.RACF.** ID(WASMNT) ACC(READ)
```

10.10.7 Profiles for WebSphere in class STARTED

In this section we present the creation of profiles in class STARTED that match procedure member names in SYS1.PROCLIB for WebSphere controllers, daemons, and servants. A profile is also created for the CTRACE procedure.

Profiles for daemon, controller, and servants procedures

To define profiles in class STARTED for procedures used by WebSphere with the TRACE option, issue the following commands:

- For the daemon procedure (base server)

```
RDEF STARTED WAS5DD1.** OWNER(WASMNT) STDATA(USER(WDD1STU) GROUP(STG)+
TRACE(YES))
```

- For the controller procedure (base server)

```
RDEF STARTED WAS5DC1.** OWNER(WASMNT) STDATA(USER(WDC1STU) GROUP(STG) +
TRACE(YES))
```

- For the servant procedure (base server)

```
RDEF STARTED WASD1S.** OWNER(WASMNT) STDATA(USER(WDS1STU) GROUP(STG) +
TRACE(YES))
```

The corresponding profiles for a managed server are:

```
RDEF STARTED WS5PDMD.** STDATA(USER(WPDDMSTU) GROUP(STG) TRACE(YES)) RDEF
STARTED WS5PDMC.** STDATA(USER(WPCDMSTU) GROUP(STG) TRACE(YES)) RDEF
STARTED WSPDMS.** STDATA(USER(WPSDMSTU) GROUP(STG) TRACE(YES))
```

Profile for CTRACE procedures

The names of the CTRACE procedure members for base server and managed server could be CTRBS01 and CTRPROD, respectively. You can create only one generic profile to cover all your CTRACE procedures, issuing the command:

```
RDEF STARTED CTR*.** STDATA(USER(CTRCSTU) GROUP(STG) TRACE(YES))
```

10.10.8 Profiles for WebSphere in class LOGSTRM

IBM recommends that the log stream resource names match the installation's data set naming conventions, with user ID or system logger application name as the first qualifier. Usually the log stream resource name is made of the log stream data set name without its high-level qualifier. For example, the log stream data name for a base server is WAS. BS01.V500.SC59.LOG, and to define the corresponding log stream name in class LOGSTRM, issue the command:

```
RDEF LOGSTRM BS01.** OWNER(WASMNT) UACC(READ)
```

You have to authorize the group that creates the log streams (groupWASMNT) with ALTER, and the user IDs that write into the log streams (all started task user IDs) with UPDATE:

```
PE BS01.** CL(LOGSTRM) ID(WASMNT) ACC(ALTER)
PE BS01.** CL(LOGSTRM) ID(WDD1STU WDC1STU WDS1STU) ACC(UPDATE)
```

Similarly for a managed server, issue the commands:

```
RDEF LOGSTRM PROD.** OWNER(WASMNT) UACC(READ)
PE PROD.** CL(LOGSTRM) ID(WASMNT) ACC(ALTER)
PE PROD.** CL(LOGSTRM) ID(WPDMSTU WPCDMSTU WPSDMSTU) ACC(UPDATE)
```

The LOGR policy information describes the characteristics of each log stream and coupling facility structure that will be used when there are active connections to the log stream.

Just in case you have not done it already to authorize people who can create, change or delete LOGR policy, define profile MVSADMIN.** in class FACILITY:

```
RDEF FACILITY MVSADMIN.** OWNER(MVSMNT) UACC(NONE)
PE MVSADMIN.** CL(FACILITY) ID(WASMNT MVSMNT) ACC(ALTER)
```

Authorize people who require reports on the LOGR policy with READ:

```
PE MVSADMIN.** CL(FACILITY) ID(WASADM) ACC(READ)
```

For more information about log streams and coupling facility structures for sysplex, refer to *z/OS MVS Setting Up a Sysplex*, SA22-7625.

10.10.9 Profiles for WebSphere in class CBIND

You can use the RACF class CBIND (optionally) to restrict the client's ability to access clusters, or you can deactivate the class if you do not require this kind of access control. There are two types of profiles WebSphere for z/OS uses in the CBIND class:

- One that controls whether a local or remote client can access servers. The name of the profile has this form:

  CB.BIND.cluster_name

  Where cluster_name is the name of the cluster.

- One that controls whether a client can use components in a cluster. The name of the profile has this form:

  CB.cluster_name

You must authorize all systems management user IDs (for example, the human user IDs of group WASADM and user ID WSADMIN) to have read access to the CB.cluster_name and CB.BIND.server_name RACF profiles.

To create backstop profiles for all not yet created resources, issue the commands:

```
RDEF CBIND CB.BIND.** OWNER(WASMNT) UACC(NONE)
RDEF CBIND CB.** OWNER(WASMNT) UACC(NONE)
```

To define profiles in class CBIND for access to clusters and usage of cluster components in base server and managed server configurations, issue the following commands:

```
RDEF CBIND CB.CLUD1 OWNER(WASMNT) UACC(READ)
```

Where CLUD1 is the name of the transition cluster for base server.

```
RDEF CBIND CB.BIND.CLPDM OWNER(WASMNT) UACC(READ)
```

Where CLPDM is the name of the transition cluster for managed server.

To define profiles in class CBIND for access to servers and usage of server components in base server and managed server configurations, issue the following commands:

```
RDEF CBIND CB.BIND.BBO* UACC(READ)
RDEF CBIND CB.BBO* UACC(READ)
```

Now, you have to authorize group WASADM with access CONTROL to all CB.BIND profiles:

```
PERMIT CB.BIND.BBO* CLASS(CBIND) ID(WASADM) ACCESS(CONTROL)
PERMIT CB.BIND.CLUD1 CLASS(CBIND) ID(WASADM) ACCESS(CONTROL)
PERMIT CB.BIND.CLPDM CLASS(CBIND) ID(WASADM) ACCESS(CONTROL)
```

10.10.10 Profiles for WebSphere in class SERVER

Servants must have access to profiles in the RACF SERVER class. This controls whether a servant can call authorized routines in the controller. Controllers do not require such access control. Only authorized programs, loaded from Authorized Program Facility (APF) libraries, run in controllers.

To create backstop profiles for all not yet created resources, issue the commands:

```
RDEF SERVER CB.** OWNER(WASMNT) UACC(NONE)
RDEF SERVER CB.*.BBO* OWNER(WASMNT) UACC(NONE)
```

To define profiles in class SERVER in base server and managed server configurations, issue the following commands:

```
RDEF SERVER CB.*.CLUD1 OWNER(WASMNT) UACC(NONE)
RDEF SERVER CB.*.CLUD1.** OWNER(WASMNT) UACC(NONE)
PERMIT CB.*.CLUD1 CLASS(SERVER) ID(WDS1STU) ACC(READ)
PERMIT CB.*.CLUD1.** CLASS(SERVER) ID(WDS1STU) ACC(READ)
```

Where CLUD1 is the name of the transition cluster for base server.

```
RDEF SERVER CB.*.CLPDM UACC(NONE)
RDEF SERVER CB.*.CLPDM.** UACC(NONE)
PERMIT CB.*.CLPDM CLASS(SERVER) ID(WPSDMSTU) ACC(READ)
PERMIT CB.*.CLPDM.** CLASS(SERVER) ID(WPSDMSTU) ACC(READ)
```

Where CLPDM is the name of the transition cluster for managed server.

10.10.11 Profiles in class REALM

Profiles in RACF class REALM should be defined if you plan to use Kerberos.

This is the command to create a profile in class REALM produced by the ISPF dialogs:

```
RDEF REALM SAFDFLT APPLDATA('WAS5')
```

10.11 HFS security

User and group authority at the HFS level should be considered when evaluating or planning application deployment. Improper access can prevent proper deployment of an application, prevent an application from starting or operating correctly, or allow unexpected access to an application.

Deploying applications to nondefault directories

One of the options in the administrative console when installing a new application is to provide a nondefault directory to install the application. This option is illustrated in Figure 10-4 on page 324.

If a directory to install the application is not provided during the installation process, WebSphere Application Server deploys the application using the default installation directory. The application .ear file is expanded in the HFS under the WebSphere Application Server installation path. The HFS directory structure for this path is by default owned by the WebSphere Application Server system ID and group ID.

When an installation directory is provided during the deployment process, WebSphere Application Server installs the application using the provided installation path. This path must have read and write permissions for the WebSphere Application Server group ID. Otherwise, the application might fail to install or application files might not be found at runtime due to the Application Server servant region lacking access to read the HFS file structure.

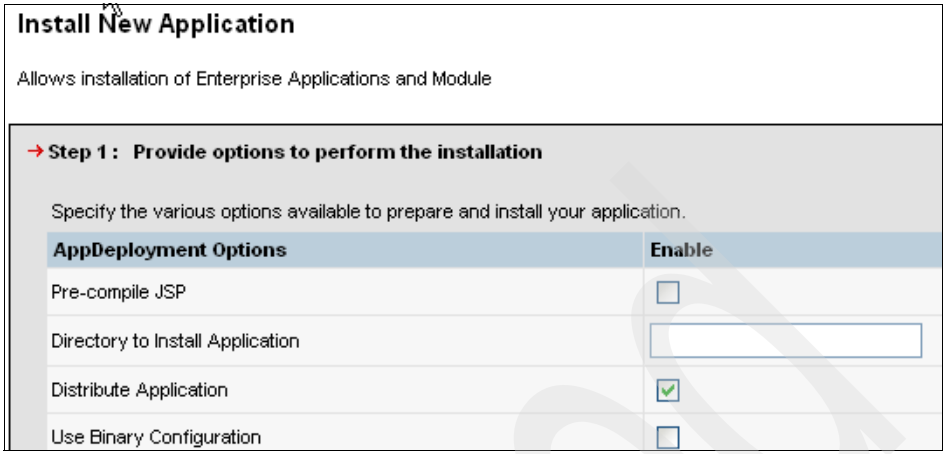


Figure 10-4 Installing a new application

10.12 HFS ACLs

It is sometimes beneficial to limit file access beyond the user and group access that USS permission bits allow. Java 2 security can be used to limit file I/O for specific applications. Often, this method does not provide the needed flexibility, or the performance impact of Java 2 security is not desired. In order to allow for greater granularity and increased functionality in HFS access control, the use of HFS ACLs is needed. This section describes one problem requiring the use of ACLs, gives a brief explanation of how ACLs work, and finishes with an implementation of a solution to the identified problem.

10.12.1 The problem: Use of ACLs

In a base Application Server, control and servant processes run as separate jobs, each with a distinct job name. The default security configuration establishes a separate user ID for each job. These two user IDs each must have write access to the same home directory, because each bears certain responsibilities for updating configuration files. This same principle applies identically to the deployment manager server.

The access to the home directory required by these user IDs is accomplished by putting the user IDs assigned to the control and servant processes in the same SAF group. There is also an administrative user ID (defaults to “wsadmin”) that is also a member of the same SAF group. The home directory is created with the administrative ID as the owner and the common group as the assigned group.

The home directory tree permissions are RWXRWXR-X for all directories except the config directory, which is RWXRWX---.

On a federated node, the node agent and jmsserver control processes (also distinct job names) are assigned distinct user IDs. These user IDs are typically in the same common group as the user IDs assigned the application server's control and servant processes on that same node. The federation process makes no change to the ownership, nor to the file permissions, of the node's home directory.

All application servers on a Network Deployment node (that is, a federated node) have read/write access to the node's home directory. This means that applications hosted in these application servers have the ability write to the node's configuration. Security imperatives require that such write access should not be possible, especially write access from an application running in one application server affecting the configuration of another application server on the same node.

ACLs can be used to permit (or deny) specific users or groups access to specific files or directories, or both, in the HFS. An understanding of ACL behavior is necessary for proper implementation. The following sections introduce HFS ACLs. If you already have an understanding of the intricacies of HFS ACLs, 10.12.6, “The solution: An ACL example” on page 327 contains the ACL approach to a solution to the above problem.

10.12.2 Details of HFS ACL functionality

z/OS has implemented the ability to grant or deny file access to individual users or groups as a functional extension of the access control in the UNIX platform. Before this support, a file has a single user and group owner with specific access control. That user and group could be granted read, write, and execute authority as needed. All other users would be controlled by a single access point defined in permissions given to “other”. The permission bit model does not allow for granting and denying access to specific users and groups, such as is possible using RACF profiles. HFS ACLs implement this functionality.

An ACL can contain up to 1024 individual entries. Each entry specifies a qualifier to indicate whether the entry pertains to a user or group, the actual user or group name (or UID/GID) itself, and the permissions being granted by this entry. The allowable permissions are the standard POSIX permissions: read, write, and execute.

To manage an ACL for a file, you must either be the file's owner or a superuser (UID=0 or READ access to SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class). This is consistent with authorization of the file mode using the

chmod command. Changes to an ACL are audited according to the SETROPTS LOGOPTIONS setting for the FSSEC class.

There are three types of ACLs. Access ACLs, file model ACLs, and directory model ACLs can all be created. Access ACLs provide protection for a file system object. File model ACLs and directory model ACLs are inherited by files and subdirectories that are created in the location where these ACLs exist. These types are described in further detail in 10.12.3, “ACL inheritance” on page 326. An ACL is deleted automatically whenever its respective file is deleted.

ACL documentation will also refer to both base and extended ACLs. The following definitions describe those types:

- Base ACL entries are the same as permission bits (owner, group, other). You can change the permissions using the **chmod** or **setfacl** command. They are not physically part of the ACL although you can use the **setfacl** command to change them and the **getfacl** command to display them.
- Extended ACL entries are ACL entries for individual users or groups. Like the permission bits, they are stored with the file, not in RACF profiles. Each ACL type (access, file model, directory model) can contain up to 1024 extended ACL entries. Each extended ACL entry specifies a qualifier to indicate whether the entry pertains to a user or a group, the actual UID or GID itself, and the permissions being granted or denied by this entry. The allowable permissions are read, write, and execute.

10.12.3 ACL inheritance

To help reduce administrative overhead associated with creating and maintaining ACLs, the notion of inheritance has been implemented. A file model ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file model ACL. A directory model ACL is copied to a newly created subdirectory as both its access ACL and directory model ACL. Inherited ACLs can be subsequently modified or deleted.

HFS space utilization should be considered before implementing model ACLs in the file system, especially file model ACLs. Use of both file and directory model ACLs in every directory in the file system will result in separate physical ACLs being created for every new file and directory. This is probably more than necessary. Although the existence of an access ACL for every directory will not likely cause space concerns, the same cannot be said of files, especially if the inherited ACLs are large. And in reality, files should not generally require an access ACL. More likely, the POSIX “other” permissions should be set such that all users have the access required. But only users with search/execute access to the parent directory will be able to get at the files contained within. Therefore, an access ACL can be used on the parent directory to grant search access only to

those users and groups who should have file access. The parent directory's access ACL can have been created automatically as the result of a directory model ACL on its parent.

10.12.4 Enabling and disabling ACL checking

Although ACLs can be created, modified, deleted, and inherited at any time, they will only be used in access decisions when the (existing) FSSEC class is active.

```
SETROPTS CLASSACT(FSSEC)
```

It should be noted that as ACLs are created and used over time, it becomes an increasingly bad idea to disable ACL checking, because it becomes very difficult to predict the state of your file system security on a system-wide basis. A comparison can be drawn to turning off generic data set protection after having defined hundreds of generic data set profiles for different users and applications over time. The ability to disable ACLs might only be useful as an emergency recovery aid, and only relatively soon after deploying ACLs. Further, you should consider that if you are using ACLs to grant, rather than to deny, access to particular users and groups, disabling ACLs will more likely result in a loss of file access authority rather than a gain. The following command can be used to disable ACL checking:

```
SETROPTS NOCLASSACT(FSSEC)
```

10.12.5 ACL creation: setfacl and getfacl

The **`setfacl`** USS command is used to set, modify, or remove an HFS ACL. The **`getfacl`** command is used to display ACLs. Details about ACL management are available in Chapter 5 of *z/OS UNIX System Services Planning*, GA22-7800. The command syntax for the **`setfacl`** and **`getfacl`** commands is available in Chapter 2 of *z/OS UNIX System Services Command Reference*, SA22-7802. These chapters will be a valuable reference when following the example found in the following section.

10.12.6 The solution: An ACL example

On a federated node, only the node agent requires unrestricted read/write access to the home directory tree. The applications servers on that node require read access to the home directory tree except for a small number of runtime directories, such as wstemp, where they require read/write access.

If wsadmin will be used in disconnected mode (that is, -conntype none), the user ID or IDs from which wsadmin will be used in this mode must have unrestricted read/write access to the home directory tree.

If the WebSphere Application Server for z/OS post installer is run in batch mode, the user ID assigned to that job must also have unrestricted read/write access to the home directory tree.

The directories in the WebSphere home directory tree will be categorized by usage:

- Public: Home, bin, lib, properties
- Runtime: wstemp, temp, logs
- Administrative: All others

If the automatic post installer is in use, properties/version and properties/service is added to the Runtime category.

The user IDs used by WebSphere will be divided into two distinct groups: an administrative group and an application server group. The node agent and wsadmin user IDs will be placed in the administrative group; the application server control and servant user IDs will be placed in the application server group. If there is a JMS server, it will also be placed in the application server group.

For simplicity, public access to those files necessary to run a WebSphere command line client will have unrestricted read/execute access to those directories that contain the files necessary to enable client execution. For users who prefer to restrict even this access, client user IDs could be separated into yet a third group: WebSphere clients. Table 10-3 summarizes these groupings.

Table 10-3 ACL access groupings

User	Security group	Directory access
Node agent	Administrative	RWX to all
wsadmin command line	Administrative	RWX to all
Batch post installer	Administrative	RWX to all
Application Server	Application Server	R-X to Administrative, Public RWX to runtime
JMS Server	Application Server	R-X to Administrative, Public RWX to runtime
WebSphere client	Client	R-X to public

To switch over to ACLs, the base permissions for the WebSphere home directory will be turned off, and specific extended ACLs will be assigned to grant appropriate permission to each of the various WebSphere security groups.

Configuration steps

The following steps must be performed by a Telnet user with authority to issue the **setfacl** command against the WebSphere home directory. For the sake of illustration, the examples that follow assume the existence of a WebSphere Application Server node home directory named /WebSphere/V5R0M0/AppServer. All commands are to be entered from the /WebSphere/V5R0M0 directory.

Also for this example, Table 10-4 describes the users and groups.

Table 10-4 Example: Assumed user IDs and groups

User ID	Group	Usage
WSASCNTL	WGAPPSRV	Application server control process
WSASSERV	WGAPPSRV	Application server servant process
WSJSCNTL	WGAPPSRV	JMS server control process
WSNACNTL	WGADMIN	Node agent control process
WSADMIN	WGADMIN	wsadmin and batch post-installer
WSCLIENT	WGCLIENT	WebSphere application client

Note: All group names used in **setfacl** commands must be defined and must have valid OMVS group IDs (GIDs) at the time the **setfacl** commands are issued. Otherwise, **setfacl** parsing errors will result.

Complete the following steps:

1. Turn off all base, extended, and default permissions:

```
setfacl -hm g::---,o::--- AppServer
setfacl -hm g::---,o::--- $(find AppServer -name "*")
setfacl -h -D a  AppServer
setfacl -h -D a  $(find AppServer -name "*")
setfacl -h -D d  AppServer
setfacl -h -D d  $(find AppServer -type d -name "*")
setfacl -h -D f  AppServer
setfacl -h -D f  $(find AppServer -type d -name "*")
setfacl -h -D e  AppServer
setfacl -h -D e  $(find AppServer -name "*")
```

2. Enable access for the client group.

   Decide if clients will be in a specific group or will be granted public access.

   If they will be granted public access, permit:

```
setfacl -hm o::r-x AppServer
setfacl -hm o::r-x AppServer/bin
setfacl -hm o::r-x $(find AppServer/bin -name "*")
setfacl -hm o::r-x AppServer/lib
setfacl -hm o::r-x $(find AppServer/lib -name "*")
setfacl -hm o::r-x AppServer/properties
setfacl -hm o::r-x $(find AppServer/properties -name "*")
```

   If clients will be in a specific group, permit:

```
setfacl -hm g:WGCLIENT:r-x AppServer
setfacl -hm g:WGCLIENT:r-x AppServer/bin
setfacl -hm g:WGCLIENT:r-x $(find AppServer/bin -name "*")
setfacl -hm g:WGCLIENT:r-x AppServer/lib
setfacl -hm g:WGCLIENT:r-x $(find AppServer/lib -name "*")
setfacl -hm g:WGCLIENT:r-x AppServer/properties
setfacl -hm g:WGCLIENT:r-x $(find AppServer/properties -name "*")
```

   Repeat to set the default for new files:

```
setfacl -hm f:g:WGCLIENT:r-x AppServer
setfacl -hm f:g:WGCLIENT:r-x AppServer/bin
setfacl -hm f:g:WGCLIENT:r-x $(find AppServer/bin -type d -name "*")
setfacl -hm f:g:WGCLIENT:r-x AppServer/lib
setfacl -hm f:g:WGCLIENT:r-x $(find AppServer/lib -type d -name "*")
setfacl -hm f:g:WGCLIENT:r-x AppServer/properties
setfacl -hm f:g:WGCLIENT:r-x $(find AppServer/properties -type d -name "*")
```

3. Enable access for the application server group.

   If the post installer will be run only in batch mode, permit read and execute for everything:

```
setfacl -hm g:WGAPPSRV:r-x AppServer
setfacl -hm g:WGAPPSRV:r-x $(find AppServer -name "*")
```

   Add selected write permissions:

```
setfacl -hm g:WGAPPSRV:rwx AppServer/logs
setfacl -hm g:WGAPPSRV:rwx $(find AppServer/logs  -name "*")
setfacl -hm g:WGAPPSRV:rwx AppServer/temp
setfacl -hm g:WGAPPSRV:rwx $(find AppServer/temp -name "*")
setfacl -hm g:WGAPPSRV:rwx AppServer/wstemp
setfacl -hm g:WGAPPSRV:rwx $(find AppServer/wstemp -name "*")
setfacl -hm g:WGAPPSRV:rwx AppServer/tranlog
setfacl -hm g:WGAPPSRV:rwx $(find AppServer/tranlog  -name "*")
```

Repeat to set the default for new files:

```
setfacl -hm f:g:WGAPPSRV:r-x AppServer
setfacl -hm f:g:WGAPPSRV:r-x $(find AppServer -type d  -name "*")
setfacl -hm f:g:WGAPPSRV:rwx AppServer/logs
setfacl -hm f:g:WGAPPSRV:rwx $(find AppServer/logs  -type d  -name "*")
setfacl -hm f:g:WGAPPSRV:rwx AppServer/temp
setfacl -hm f:g:WGAPPSRV:rwx $(find AppServer/temp -type d  -name "*")
setfacl -hm f:g:WGAPPSRV:rwx AppServer/wstemp
setfacl -hm f:g:WGAPPSRV:rwx $(find AppServer/wstemp -type d  -name "*")
setfacl -hm f:g:WGAPPSRV:rwx AppServer/tranlog
setfacl -hm f:g:WGAPPSRV:rwx $(find AppServer/tranlog  -type d -name "*")
```

If the post installer is run as part of control process startup, permit also:

```
setfacl -hm g:WGAPPSRV:rwx AppServer/properties/version
setfacl -hm g:WGAPPSRV:rwx $(find AppServer/properties/version -name "*")
setfacl -hm g:WGAPPSRV:rwx AppServer/properties/service
setfacl -hm g:WGAPPSRV:rwx $(find AppServer/properties/service -name "*")
```

Repeat to set the default for new files:

```
setfacl -hm f:g:WGAPPSRV:rwx AppServer/properties/version
setfacl -hm f:g:WGAPPSRV:rwx $(find AppServer/properties/version -type d
-name "*")
setfacl -hm f:g:WGAPPSRV:rwx AppServer/properties/service
setfacl -hm f:g:WGAPPSRV:rwx $(find AppServer/properties/service -type d
-name "*")
```

4. Enable access for administrative group:

```
setfacl -hm g:WGADMIN:rwx AppServer
setfacl -hm g:WGADMIN:rwx $(find AppServer -name "*")
```

Repeat to set the default for new files:

```
setfacl -hm f:g:WGADMIN:rwx AppServer
setfacl -hm f:g:WGADMIN:rwx $(find AppServer -type d  -name "*")
```

10.13 zFS security

For the purposes of the file system security described in this book, a zSeries file system (zFS) has the same functionality as an HFS. Details of the differences between HFS and zFS are available in Chapter 7 of *z/OS UNIX System Services Planning*, GA22-7800.

11

Registries

Information about users and groups must reside in a user registry. In WebSphere Application Server, a user registry keeps:

- User information pertaining to the user identity, password, and groups. It helps to authenticate a user and retrieves information about users and groups
- Authorization information towards objects that are hosted in the WebSphere Application Server.

Implementation is provided to support different registry types, local (System Authorization Facility (SAF)-based) and the following remote or pluggable registries:

- Lightweight Directory Access Protocol (LDAP)
- Tivoli Access Manager for e-business
- Custom user registry

This chapter presents general user registry concepts.

- Authentication
- Authorization

Local registry concepts are presented in Chapter 12, “Local operating system registries” on page 343. Remote registry concepts are covered in detail in Chapter 13, “Remote registries” on page 361.

11.1 User registry overview

The term “registry” is used to describe a data store which functions primarily as a read-only collection of objects arranged in a simple hierarchy. User registries primarily contain a list of users, and associated groups of users, though they can also contain other relatively static information such as organizational structure.

11.1.1 User registry authentication data

WebSphere always uses a registry for authentication data.

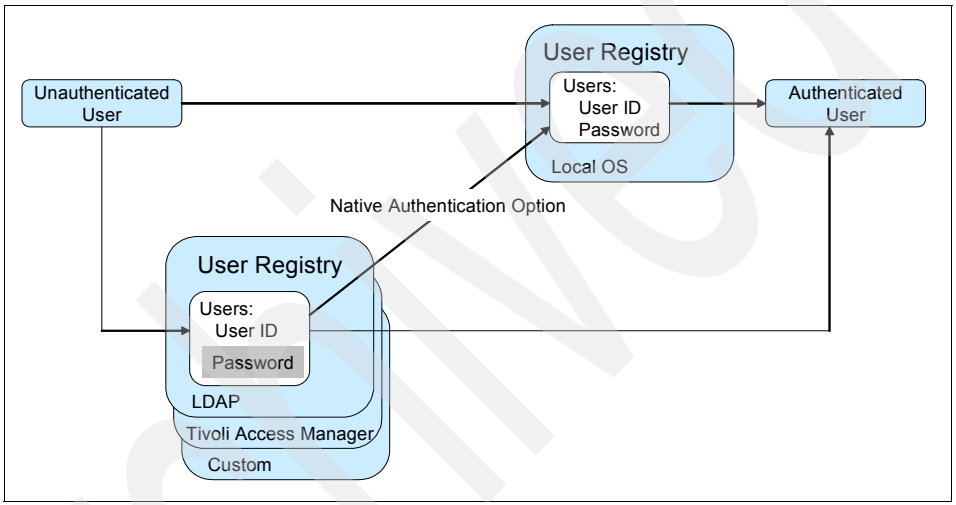


Figure 11-1 User authentication

The figure shows that all user registries contain user IDs and passwords. However, some registries such as LDAP do not encrypt passwords. For this reason, most security implementations will use a feature described as native authentication, which means delegating the password check to RACF, where passwords are encrypted. The details of this process are described in 13.5, “Sample scenario: Using LDAP native authentication” on page 374

A registry also contains user names to support auditing requirements. The registry can also contain additional user information that is not required for authentication, such as membership in an organizational hierarchy.

11.1.2 User registry authorization data

User registries can contain information that can be used to map users and groups to J2EE roles to support the WebSphere authorization process. This is described as registry-based authorization, as shown in Figure 11-2.

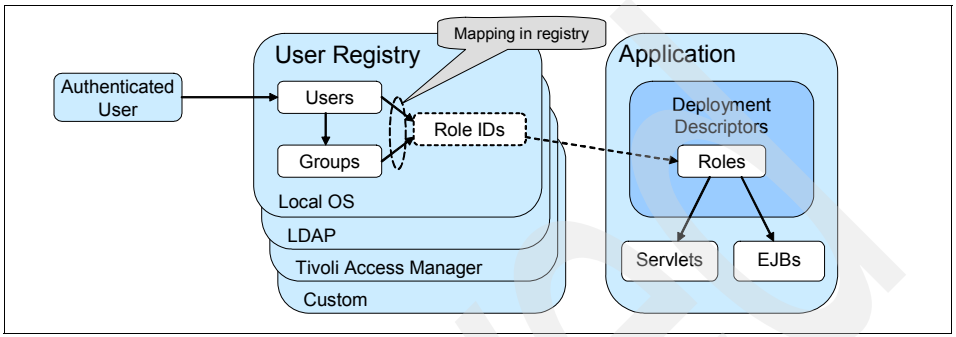


Figure 11-2 User registry-based authorization

User registry configuration in WebSphere can also support using deployment descriptors as the source of authorization information, as shown in Figure 11-3.

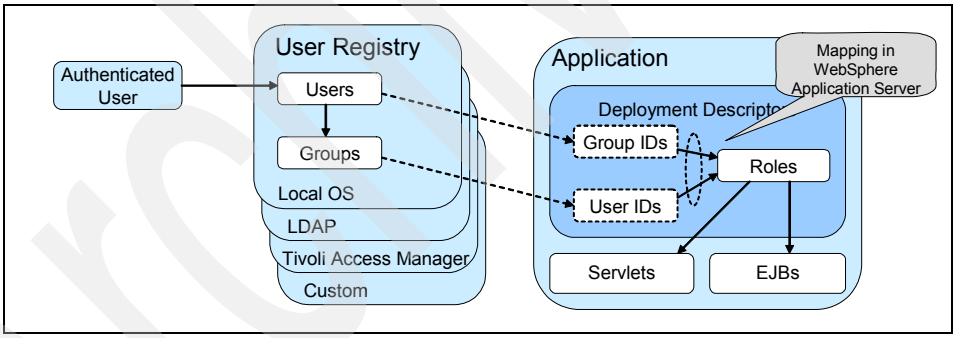


Figure 11-3 WebSphere-based authorization

Figure 11-2 and Figure 11-3 provide a visual view of the data hierarchy by which authorization to servlets and EJBs are associated to user IDs. They do not represent the sequence in which WebSphere authentication objects navigate the data; for details about that process, refer to Figure 11-6 on page 339.

11.1.3 Local versus remote registries

A user registry is characterized as “local” or “remote”. This distinction is used to contrast registries which must reside on the same host system or sysplex as the WebSphere application, and registries which can be hosted anywhere.

RACF (and other third-party SAF-compliant products) provide established authentication and authorization services for z/OS. RACF is closely integrated with z/OS subsystems, and was designed to support only local access. The close integration of this facility means that RACF authentication and authorization supports higher performance than remote registries.

Other user registries provide local client access to a remote registry server, so that application servers running on different hosts can access the same registry. This includes LDAP and Tivoli Access Manager, as well as custom registries. A remote registry is required for single sign-on support, but the trade-off is decrease performance relative to RACF. A detailed explanation of single sign-on, and the role that remote registries play in the implementation, is in 16.2.8, “HTTP-based single sign-on” on page 536.

There is a significant wrinkle in this distinction, however, in that a single sign-on solution is likely to use a remote registry enabled for native authentication against a local RACF registry running on one of the systems included in the single sign-on domain, as discussed in 11.1.1, “User registry authentication data” on page 334.

11.1.4 Choosing a registry

Figure 11-4 on page 337 illustrates the fact that a WebSphere cell is configured with a single user registry; this is one of the global security settings in the WebSphere administrative console.

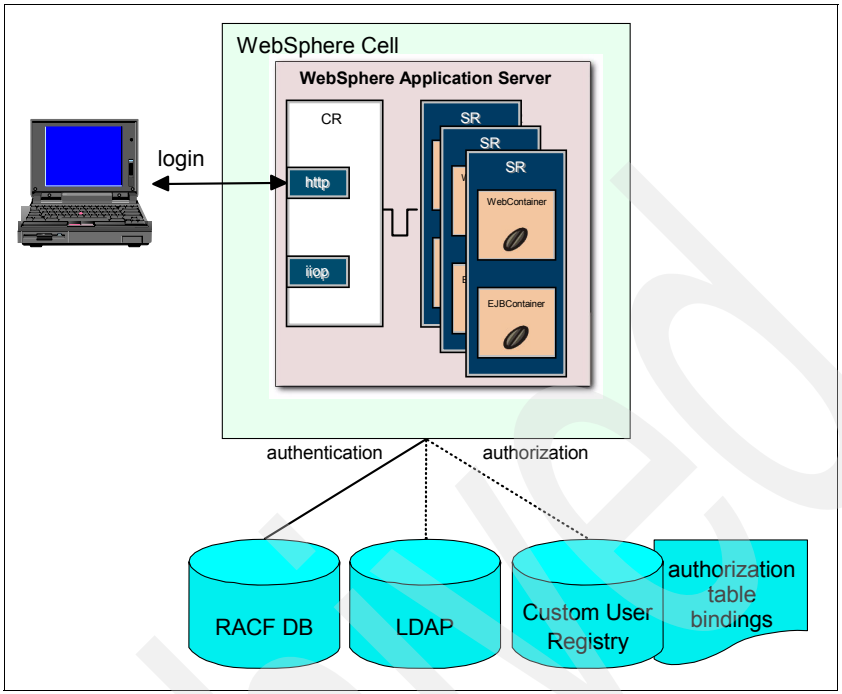


Figure 11-4 Choosing a registry

Choosing the user registry must take into account several technical requirements, including auditing and control, performance, server topology, the need for single sign-on, and ease of maintaining user data. Chapter 12, “Local operating system registries” on page 343 describes the features and benefits of using a local OS registry, and Chapter 13, “Remote registries” on page 361 describes the features and benefits of using a remote registry.

A comparison of all of the available registries, local and remote, is presented in summary form in 13.10, “Registry choices summary table” on page 413.

11.2 Authentication overview

Authentication is the process of establishing a client’s identity and determining whether the client is valid within a particular context. A client can be an end user, a machine, or an application. When configuring a cell, you must select a single authentication mechanism (that collaborates with a user registry). The choices

for authentication mechanisms for WebSphere Application Server for z/OS and OS/390 include:

- Simple WebSphere Authorization Mechanism (SWAM), which is available only on the base Application Server (not available on the Network Deployment) configuration
- Lightweight Third Party Authentication (LTPA)
- Integrated Cryptographic Service Facility (ICSF)

Authentication mechanisms are executed by the authentication module. The correlation between the authentication module and the user registry is depicted in Figure 11-5.

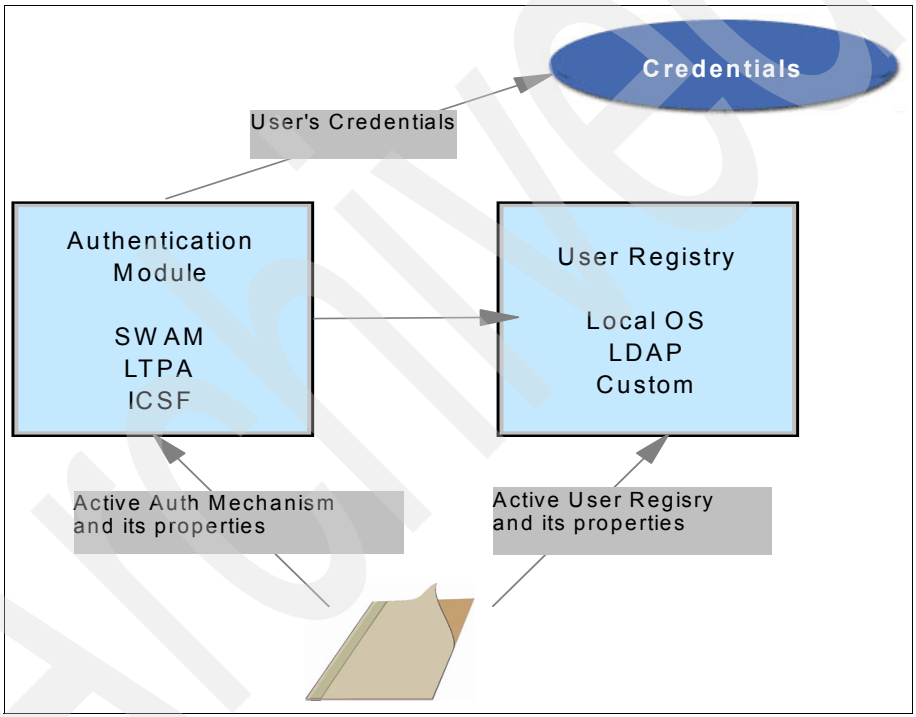


Figure 11-5 Authentication mechanisms and registry

The authentication module uses the user registry to create user credentials that are usable for the authorization process.

The correlation between authenticators and authentication modules is depicted in Figure 11-6 on page 339.

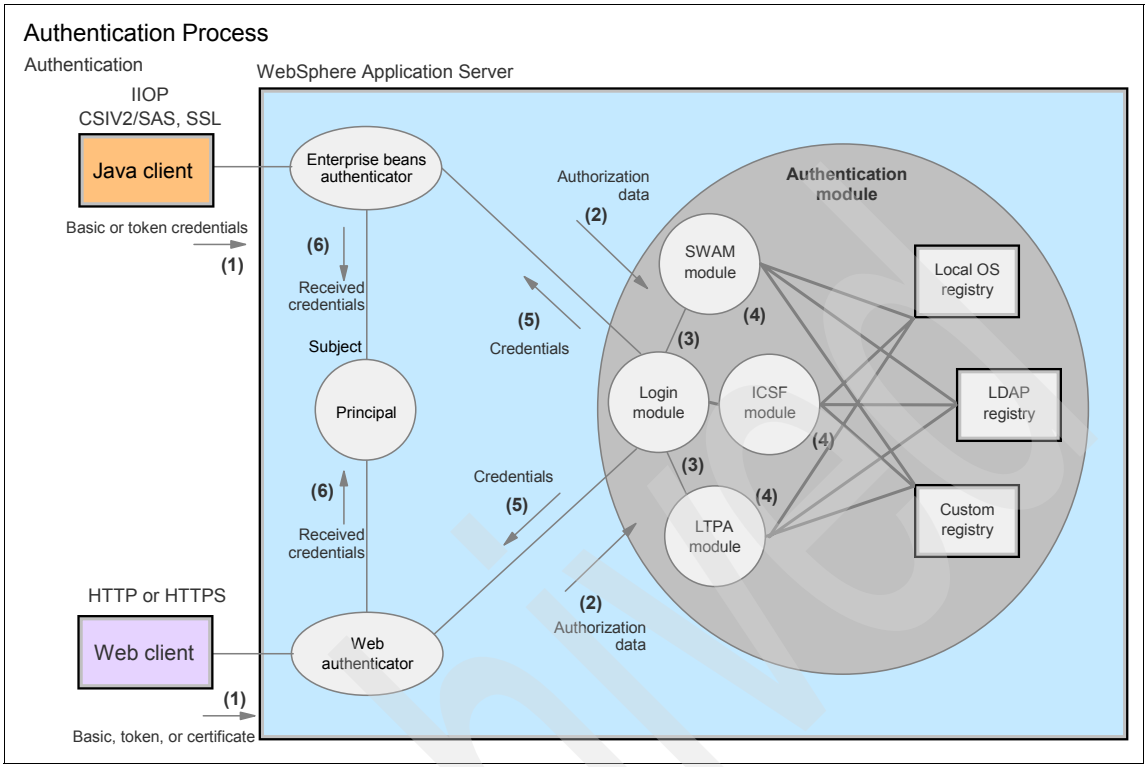


Figure 11-6 The authentication process

The authentication process is as follows:

1. Authentication credentials are passed to the WebAuthentication Module or EJB Authentication module in the CSIV2 and zSAS layer of the WebSphere Application Server.
2. The authentication module then passes the user data to the login module, which determines whether or not to process an LTPA, ICSF, or SWAM login.
3. The LTPA, ICSF, or SWAM login modules receive the user data for authentication.
4. The authentication of the user is completed against one of the supported registries (local OS, LDAP, or custom user registry).
5. If authentication is successful, the login module proceeds to generate a JAAS subject and credentials list and returns this information to the Web or EJB Authentication module.
6. The Web Authentication Module or EJB Authentication Module stores the information for future access requests.

11.3 WebSphere authentication mechanisms

Unless you can be sure that all messages exchanged flow exclusively within a trusted network, authenticity of clients and clusters, message confidentiality, and message integrity become important issues. A client might want to be sure that it is receiving a service from a legitimate cluster, and a cluster might want to be sure who the client is. Each party also wants to be sure that messages exchanged are protected from tampering or snooping by a malicious third party, so security in the transportation medium (message protection) is a concern. WebSphere Application Server for z/OS and OS/390 provides several authentication mechanisms, some of which involve message protection. Based on the nature of your network, you must decide which authentication mechanism you need.

WebSphere Application Server for z/OS Version 5 supports the following authentication mechanisms, regardless of the configured registry:

- The Simple WebSphere Authentication Mechanism (SWAM) is intended for simple, non-distributed, single application server runtime environments. The single application server restriction is due to the fact that SWAM does not support forwardable credentials. If a servlet or enterprise bean in application server process 1 invokes a remote method on an enterprise bean living in application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, can cause authorization failures.

  Because SWAM is intended for a single application server process, single sign-on (SSO) is not supported. The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

- Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. It supports forwardable credentials and single sign-on (SSO). LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data and later decrypt and verify the signature. The LTPA protocol enables WebSphere Application Server to provide security in a distributed environment using cryptography. If you need to interoperate with distributed WebSphere Application Servers not running under z/OS, you need to use LTPA.

- Integrated Cryptographic Service Facility (ICSF) uses hardware encryption available on zSeries servers running the z/OS operating system. ICSF also generates security tokens for authenticated users, which can be propagated over to other zSeries servers running the z/OS operating system. The main advantage of ICSF is that the keys are stored in secure hardware and only the key label is provided.

11.4 Authorization overview

An identity is derived by the authentication and identity mapping process most likely for authorization purposes. You can authorize clients (users) or servers to access resources managed by WebSphere Application Server or by the operating system.

The characteristics of each client require special consideration:

- Is the client on the local system or is it on a remote system? You must consider the security of the network for remote clients.
- Will you allow unidentified (unauthenticated) clients to access the system? Some resources on your system might be intended for public access, while others need to be protected. In order to access protected resources, clients must establish their identities and be authorized to use those resources.

Authorization is the process of checking whether a user or principal is allowed to access a protected resource in the way intended by the program (in most cases). For authorization to be meaningful, the user or principal needs to be authenticated or an indentity has to be trustfully assigned (for example, Started Task Table).

With WebSphere Application Server for z/OS and OS/390, authorization can happen at two different levels:

- Operating system resources can be protected at the operating system level. If a program accesses a protected resource, the resource manager uses a call to System Authorization Facility (SAF) to let the security manager, typically RACF, perform an authorization check.
- WebSphere Application Server for z/OS resources (such as bindings and Web sites) can be protected at the WebSphere infrastructure application level. If a J2EE application has a security constraint, the container might use a SAF call to let the security server perform an authorization check (if a local SAF registry is being used), call Tivoli Access Manager (see 14.1.4, “Scenario 2: Tivoli Access Manager authentication and authorization for WebSphere” on page 424), or check authorization against a binding file provided with the application at installation time.

Authorization details for WebSphere resources

The authorization aspect of WebSphere security is based on whether or not an authenticated user has rights to access a requested resource. In most instances, resources are protected through the implementation of user roles and groups that are generated when the application is being deployed. WebSphere follows the authorization process depicted in Figure 11-7 to decide whether to permit or deny user access to specific resources.

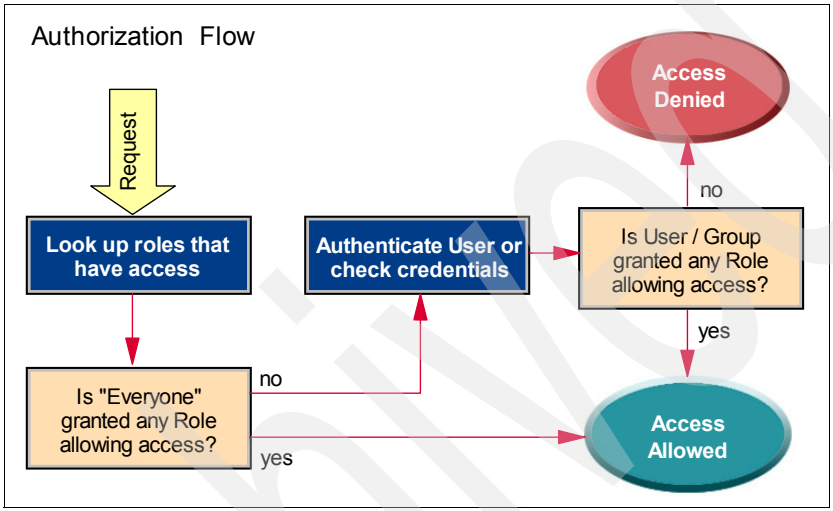


Figure 11-7 WebSphere authorization flow

WebSphere follows this authorization process, as shown in Figure 11-7:

1. A request for a Web resource is received by either the Web or EJB container in WebSphere Application Server.
2. If no user is currently logged in, the application server checks to see if the requested resource is a “public” resource. In this case, the role is mapped to “everyone,” and the action requested is permitted.
3. If the requested resource is restricted by a user or group mapping, the user is asked to authenticate with the application server.
4. After a successful login, the user’s credentials are checked to establish whether or not the user is mapped to the required role or is a member of an authorized group. If the user is a member of one or more authorized groups or roles, the requested action is granted. Otherwise, the action is denied and a return code is generated.

12

Local operating system registries

This chapter presents topics based on the use of a local, SAF-based, operating system registry:

- Authentication
  - User identification and authentication
  - Digital certificates and user certificate mappings
  - RACF PassTickets
- Authorization
  - Operating system resource authorization
  - WebSphere application resource authorization
  - SAF-based authorization concepts
  - Identity mapping with local registries
- Enablement of the local registry interface

Remote registry concepts are covered in detail in Chapter 13, “Remote registries” on page 361.

12.1 Authentication with a local registry

When should I use a local registry?

- When the authenticated users are present in the local security subsystem (intranet)
- When best performance is mandatory
- When comprehensive end-to-end security is needed (user ID present in the Web server, Web container, EJB container, and back-end system
- When good auditing is needed
- When the users and application security need to be managed by the RACF security administrators

Proper security for any system requires that users or programs be assigned an identity to identify themselves and prove they are who they claim to be (in other words, authenticate themselves).

The following list explains the kinds of user identification and authentication WebSphere Application Server for z/OS and OS/390 uses within and across systems:

- Identities derived by WebSphere itself using its authentication module, zSAS or CSIV2. These identities are assigned to principals in a Java object that is called the Subject.
- Login local clients usually use their user IDs and passwords to identify themselves when requesting a service. Other requestors might not need to provide a password, because authentication might be based on certificates or Started Task Table entries. Whenever an address space is created, its credential information is also created and then stored in an Access Control Environment Element (ACEE).
- WebSphere Application Server for z/OS and OS/390 uses a transportable form of the user's ACEE, called a RACF Object (RACO) or environment object, for local clients and clusters. The environment object is used throughout the WebSphere Application Server for z/OS and OS/390 implementation and ensures that any task is performed under the requestor's identity. See Figure 12-1 on page 347 for an illustration of the environment element and environment object. No further authentication is required, because the user's identity is already established by the operating system. Just as with other z/OS applications, WebSphere uses the operating system

to keep track of the user identities and makes calls to the security service during the execution of a piece of work.

An environment object is simply a transportable form of the ACEE, which an authorized application (APF, system key, or supervisor state) could acquire from a SAF-based registry, transport from one address space to another on the same system, or from one system to another in a sysplex, and return to the SAF-based registry to represent a user's security environment. However, WebSphere does not use the environment object when a request goes from one J2EE server to another within a sysplex. An environment object is the result of resolving the ACEE pointers and putting all the data into a single byte stream in serial fashion. After this is done, the ACEE can be transported from one server (address space) to another. The environment object is used by WebSphere when a controller needs to pass the identity of an authenticated user to the server regions. This avoids the need to repeat authentication in the server regions.

- Within the sysplex, all security protocols (except for environment objects) are supported between clients and clusters within the sysplex. Additionally, PassTickets are supported, where the client's user ID is used for identification and a PassTicket for authentication.

  A PassTicket is a one-time-use password that is dynamically generated. Because communications within a sysplex flow directly over a protected network, WebSphere for z/OS avoids the overhead of message encryption for these communications. In other words, when systems in a sysplex are directly connected, WebSphere for z/OS determines that the communication is guaranteed to be secure, and does not use encryption. When a client connects to a cluster, part of the connection includes a negotiation between the client and cluster about what security protocol is to be used. The concept of PassTickets is explained in detail in "RACF PassTickets" on page 349.

- In server-to-server connections, a server who calls another server on behalf of a client is, under specific circumstances, able to transmit the identity of the client to the server that is called. A highly-trusted connection between the two servers must exist for this transfer of identity without further authentication to be allowed. The requesting server simply sends the client's user ID in a process called *asserted identity*.

Identifying and verifying users

Identification can be in the form of a user ID and password, PassTicket, started task user ID, or digital certificate, which must be unique to each individual. When presenting identification to the operating system, the password or certificate is verified against a SAF-based registry. This registry allows you to control password options such as requiring the password to pass certain criteria (such as being a minimum length, containing a mixture of alphanumeric characters, having an expiration date).

User IDs and passwords

Authentication begins as soon as you enter a user ID and associated password into the computer operating system. The SAF-based registry then checks its database (sometimes referred to as the user registry) to determine if the user ID you entered is valid and if the password (which is one-way encrypted for security) matches what is stored in the database. After the user ID is found in the registry and it is verified that the password matches, it authenticates you based on the return code generated by the SAF-based registry. A return code of 0 indicates successful authentication. This is the same process used in batch jobs when they inherit the security context (ACEE), unless you force a new one by coding USERID= on the job card.

How does the environment object relate to a Java principal?

After successful authentication, WebSphere Application Server for z/OS builds subjects with principals and environment objects for the credentials for the authorization check. This means the RACO becomes a credential referenced by the Subject (on a Local OS). So, when WebSphere passes an environment object from one place to another for authentication, it can take the environment object from one subject and send it to a different server. The receiving server can build a new subject (including a principal and the environment object) with information from the environment object. This means that the environment object does not transport the principal, but does transport all the information needed to build a principal in the destination environment, plus more.

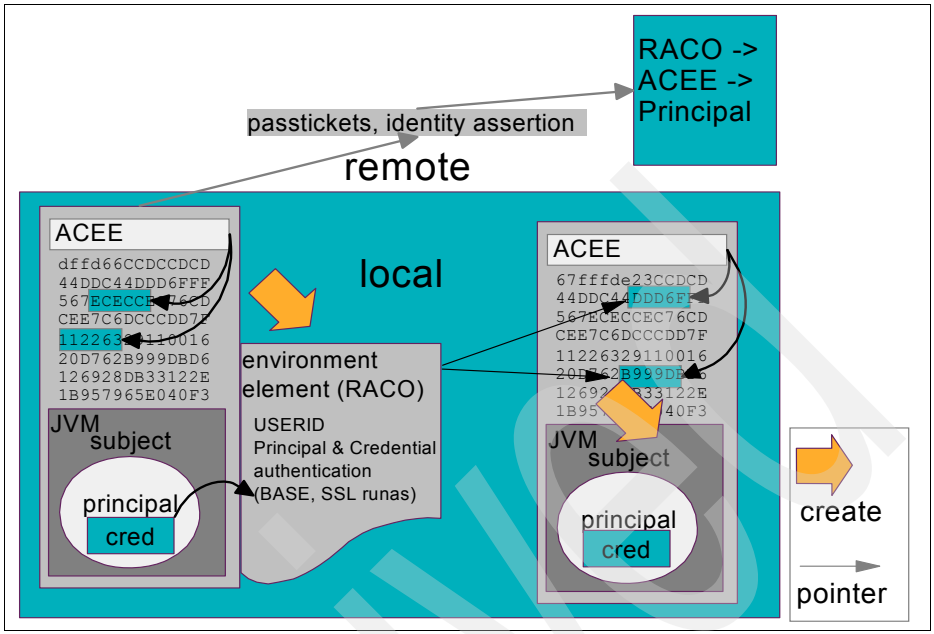


Figure 12-1 The environment element

SAF-based registry password rules allow you to adjust the quality of the password challenge. They are very flexible, allowing the system administrator to set minimum and maximum password lengths, the mixture of alphanumeric characters, and the frequency with which passwords must be changed.

All good security products require a good password encryption standard. SAF-based registry does this by encrypting the password as soon as it is entered, using a one-way Data Encryption Standard (DES) encryption algorithm. In this way, the password is never stored in clear text. An entered password is hashed and the hash value is compared against the one stored in the SAF-based registry database.

Tip: A SAF-based registry allows your passwords to expire. Not all authentication methods support expired passwords nicely; RFC or Java standard implementations fail to deliver expired password support. If you need this functionality, you can choose between the following methods:

- Tivoli Access Manager for e-business WebSEAL front end connected to LDAP running on z/OS back end by a SAF-based registry database.
- Enhanced form-based login.
- Basic authentication with a front end of an IBM HTTP Server accessing the same SAF-based registry as WebSphere Application Server. IBM HTTP Server should have the expired password support installed.

Digital certificates and user certificate mappings

Clients can authenticate to a server by presenting an X.509 digital certificate and proving ownership of the private key that is associated with the certificate. Client authentication with a certificate is part of the Secure Sockets Layer (SSL) V.3 or Transport Layer Security (TLS) 1.0 protocol. This means that an SSL or TLS session is required. The transfer of the client certificate and the certificate verify message (that proves ownership of the private key) are part of the SSL handshake. RACF itself provides three different ways to associate a RACF user ID with a client certificate:

- A certificate can be stored in the RACF database with a user ID associated with it. This can either be done using the RACF command RACDCERT or using a certificate self-registration application that allows Web browser clients to register their own certificates in RACF. Public Key Infrastructure (PKI) Services for z/OS can be used to manage this infrastructure. This method provides a one-to-one relationship between certificates and user IDs.
- Certificate name filter rules can be defined in RACF using the RACDCERT command. If the portion of the certificate owner's distinguished (DN) name or the certificate issuer's distinguished name in the filter rule match the certificate, the user ID assigned to the filter rule is associated with the certificate. This method is typically not used to provide a one-to-one mapping between certificates and user IDs because this would require a separate filter rule for each certificate. Certificate name filter rules are used in cases where a number of certificates map to a common user ID, typically a surrogate user ID.
- A certificate can contain an extension named host identity mappings. This extension contains a list of host names and associated user IDs. If the certification authority that signed the client certificate is trusted for this extension and an entry in the list corresponds with the host name of the system, RACF uses the user ID from the list entry.

Note: If you are using a security product other than RACF, check with your vendor for support for associating user IDs with certificates. For all intents and purposes, certificate-based authentication needs to associate a user ID with the certificate.

Users defined in the registry can have mappings to identities on other platforms. This allows flexible user identity mapping on platforms. Because most users in the registry only need Web access to platforms, this facilitates tighter control on the platforms.

You can authorize your servers to accept connections for clients whose certificates contain the host identity mappings extension by administering profiles in the SERVAUTH class. Each server you want to authorize must have a unique RACF user ID (if not already defined). Note that servers can run as jobs or started procedures.

RACF PassTickets

The RACF secure signon function provides an alternative to the RACF password called PassTicket. Instead of having the user's clear text password flow over the network, a RACF PassTicket can be generated by a requesting product or function and used as the user authenticator to a RACF-secured network application.

This password substitution flows within traditional password-based protocols and is a one-time-only password. In addition to the possibility of improved security for passwords in the network, PassTicket technology can be used to effectively move the authentication of a mainframe application user ID from RACF to another authorized function running on the host system, or to the work station local area network (LAN) environment. The PassTicket has its own RACF class called PTKTDATA. Because PassTickets can be generated dynamically by a trusted server, based on a secret encryption key that is shared with the target RACF server, they can also be used to forward an authentication from one trusted server to another. Using PassTickets eliminates the need to repeatedly send RACF passwords across the networks as clear text.

Note: A PassTicket is considered to be within the valid time range when the time of generation (with respect to the clock on the generating computer) is within plus or minus 10 minutes of the time of evaluation (with respect to the clock on the evaluating computer).

12.2 Authorization with a local registry

With WebSphere Application Server for z/OS and OS/390 running with a local registry configured, authorization is performed by the operating system's security manager (RACF or an equivalent product). Therefore, it is essential that authenticated users are assigned to a security manager (RACF) user ID. Access rights for this user will be checked against profiles in the RACF database.

12.2.1 Operating system resource authorization

Note: You can authorize against bindings even with RACF configured for authentication.

When a program accesses a resource that is protected at the operating system level, an authorization check is performed by the security manager (RACF). Typical operating system resources are, for example, MVS data sets, z/OS UNIX file system objects, connections, TCP/IP ports, or execution rights for programs.

In z/OS and z/OS UNIX a RACF Access Control Environment Element (ACEE) is generally associated with a thread. Therefore, the authorization check performed by RACF is using the security environment of the current thread. It is important to know which user ID the current thread is running under.

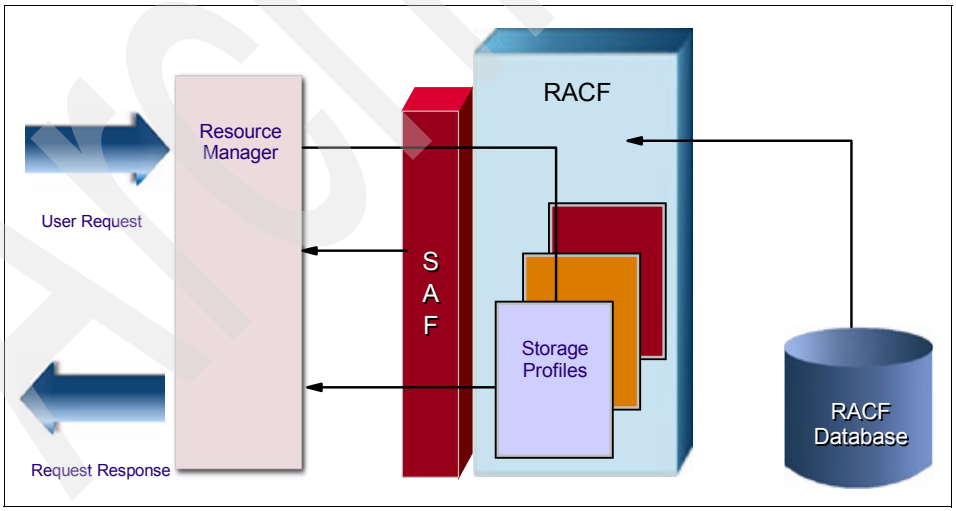


Figure 12-2 WebSphere Application Server resource authorization

12.2.2 WebSphere resource authorization

In WebSphere Application Server for z/OS and OS/390, the Web container and EJB container are both able to perform resource authorization by the WebSphere Application Server infrastructure. Although the authorization is performed by the security server (RACF), it needs to be understood that the authorization is performed by the WebSphere Application Server itself.

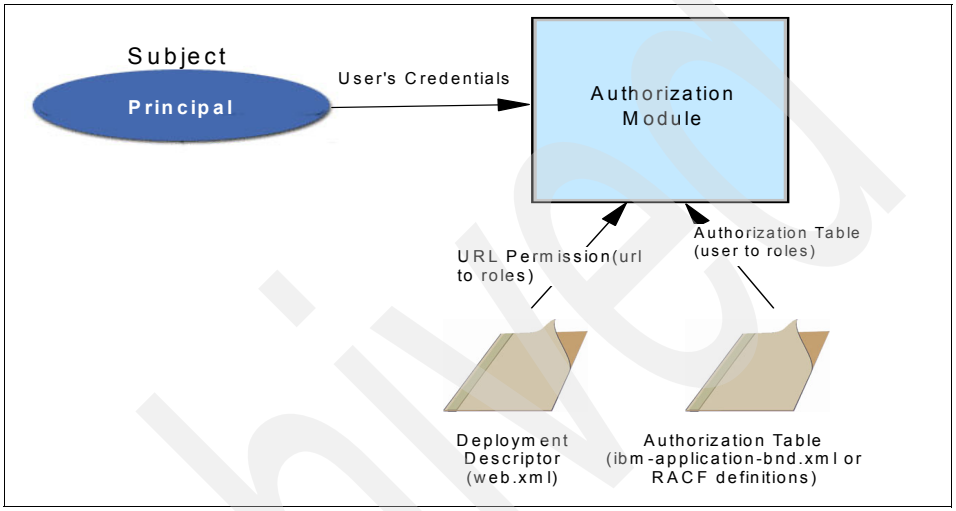


Figure 12-3 WebSphere Application Server Web container resource authorization

12.2.3 SAF-based WebSphere authorization concepts

System Authorization Facility (SAF)-based authorization (for WebSphere, using the RACF EJBROLE profile) can be used as the mechanism for adding local users if the user registry is defined as LocalOS.

RACF class EJBROLE

EJBROLES can be used to control a client's access to enterprise beans and resource collections. In the web.xml file, in the .war file, or in the Ejb-jar.xml file one has the possibility to define security roles. When roles are coded in one of the deployment descriptors, this has to match up with the RACF profile in the EJBROLE class, and users who need access to these resources need to have access to the corresponding EJBROLE profile.

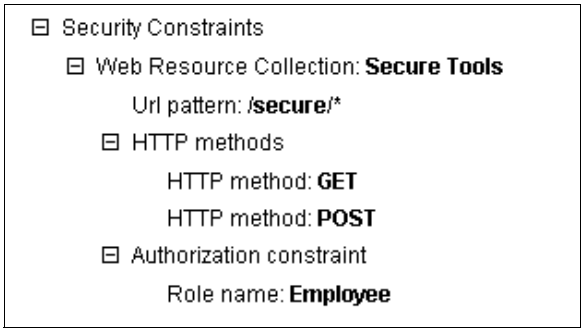


Figure 12-4 Administrative view of a web.xml deployment descriptor

If the user ID is valid and has at least READ access to the EJBROLE profile that is defined in the deployment descriptor, the user is allowed to access the resource.

This brings us to the implementation of roles in WebSphere Application Server for z/OS Version 5. Roles, as defined by J2EE, are not inherent in the SAF model. In order to implement J2EE roles, a new SAF class, EJBROLE, was created. (Do not be confused by the name EJBROLE. When this class was defined, it was thought that it would be useful only for EJBs. It is used, however, for J2EE roles in both EJBs and Web applications.)

When an application deployer uses a role in a component's deployment descriptor, the role name must be identical to the name of an EJBROLE profile. A security administrator defines EJBROLE profiles and permits SAF users or groups to the profiles. In order to be considered eligible for a role, a user must have READ access to the EJBROLE profile or be connected to an SAF group that has READ access.

Note: The information above for EJBROLE also applies to Grouping EJBROLE (GEJBROLE) below.

RACF grouping class GEJBROLE

The SAF interface also supports a grouping class for the EJBROLE class. This grouping class is called GEJBROLE. It is particularly useful when you have a need to give access to the same users or groups for several roles.

The GEJBROLE grouping class provides a capability not natively available in other J2EE servers. Using the J2EE security model, if we have several components or applications that use different role names for similar functions

(such as Hire, Promote, GrantPayraise for managerial functions), there are several ways to handle this:

- One would be to adjust the applications' deployment descriptors so that they conform to the roles already defined in our enterprise (for example, Managers). This is time consuming and error prone, especially because it might require a readjustment of the deployment descriptor each time the application is changed or reinstalled.
- Another approach would be to define the EJBROLE profiles for each of the roles required by the application. Then the users and groups to be given access to these roles would have to be permitted. This could become an administrative headache, because the same users and groups would be permitted to several different profiles with similar meanings.
- The grouping class provides a third approach that avoids the worst pitfalls of the other two. We would still define EJBROLE profiles for each of the roles required by the application, but instead of permitting all of the same users and groups to the new profiles, we create a profile (such as Supervisors) in the grouping class and add all of the new EJBROLE profiles to it. Every user and group that needs access to these roles can now be permitted in one place, the Supervisors profile. We can further avoid administrative work by simply adding our existing EJBROLE profile (Managers) to the grouping class profile (Supervisors).

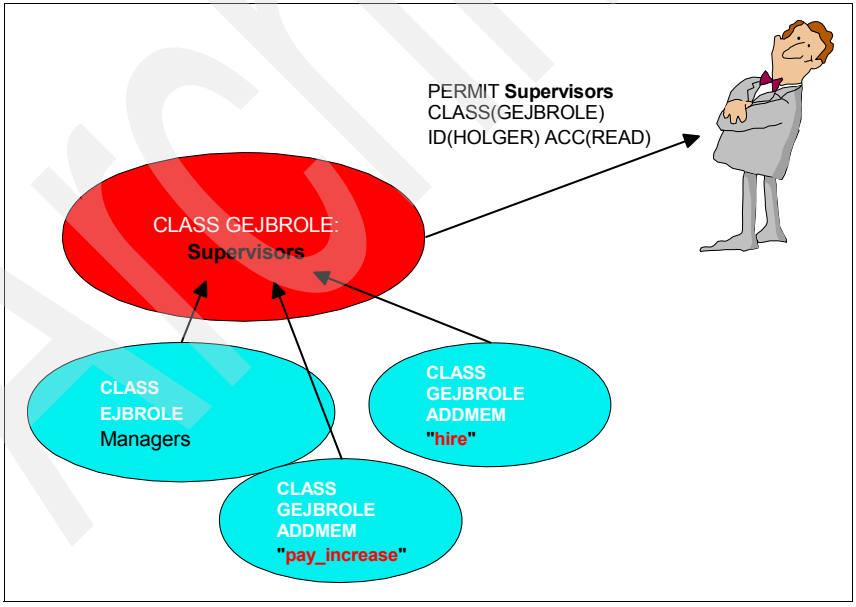


Figure 12-5 Grouping EJBROLEs with GEJBROLE

Figure 12-5 on page 353 shows the relation between GEJBROLEs, EJBROLEs, and EJBROLEs within the GEJBROLE (ADDMEM).

Tip: Implementing GEJBROLEs:

- Plan organizational role profiles in RACF class GEJBROLE first.
- Create the access list by permitting user groups to the GEJBROLE profiles, and then add roles to the GEJBROLE profiles.
- A GEJBROLE with only one EJBROLE is OK.
- Do not use a mixture of EJBROLE and GEJBROLE for permitting users to roles. If possible, permit users to GEJBROLE profiles only.
- Generally use GEJBROLE in preference to EJBROLE.

Security domains

The specification of a security domain prefix affects the EJBROLE profiles used by WebSphere Application Server for z/OS system resources when SAF authorization is chosen.

When security.ZOS.domainType=cellQualified is specified, the WebSphere runtime J2EE application EJBROLE profiles are defined by including a security domain prefix.

The security.zOS.domainType property specifies if there is a security domain used to qualify security definitions. In WebSphere Application Server for z/OS, values for the domain type can be specified as follows:

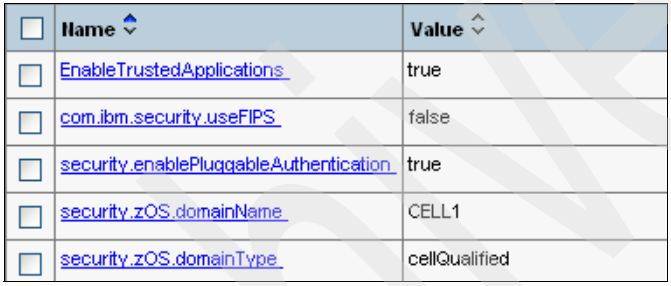
`none` This indicates that SAF security definitions are of the global sysplex scope.

`cellQualified` This indicates that WebSphere runtime uses the domain name specified in property security.zOS.domainName to qualify SAF security definitions.

If security.zOS.domainType is not defined, or a value is not set, `none` is assumed.

Note: Although the property to enable the prefixing of EJBROLE profiles is called security.ZOS.domainType=cellQualified, the name you set on the property security.ZOS.domainType does not have to be the same as the cell name. Imagine that you have an LPAR with one production cell and several test cells. You might want to define only two sets of EJBROLE profiles, one for the production cell and another shared by all the other cells. For the EJBROLE profiles that are to be used by only the production cell, it will make sense to use the production cell's name as the prefix. For the EJBROLE profiles used by the several test cells, you can choose any suitable security.ZOS.domainName, for example, security.ZOS.domainName=TEST.

An example of a security domain definition is illustrated in Figure 12-6. In this example, a cell qualified domain was implemented with domain name `CELL1`.



Name	Value
EnableTrustedApplications	true
com.ibm.security.useFIPS	false
security.enablePluggableAuthentication	true
security.zOS.domainName	CELL1
security.zOS.domainType	cellQualified

Figure 12-6 Defining security domains

Note: In the previous example, access to the administrative console is restricted to users permitted to EJBROLEs CELL1.administrator, CELL1.configurator, CELL1.operator, or CELL1.monitor.

Security domain property values can be set in the administrative console by selecting **Security** → **Global Security** → **Custom Properties**.

The value for security.zOS.domainName must be an uppercase string from 1 to 8 characters in length. If a domain name value is specified and cellQualified is selected, the domain name is also used to identify the application name used in the APPL and PassTicket profiles. If a domain name value is not specified, the default value `CBS390` is used.

A security domain can also be set up in the WebSphere Application Server for z/OS customization dialogs. The advantage of setting up the security domain in the customization dialogs is that RACF commands are generated to define necessary EJBROLEs, taking into consideration the security domain prefix.

RACF commands are also generated to permit users and roles to these EJBROLEs.

When defining security domains using the administrative console, EJBROLEs preceded by the security domain prefix need to be manually created in RACF. Users need to also be manually permitted to the new EJBROLES.

Why should I enable this function?

Assume that you have multiple WebSphere cells sharing the same RACF database. These cells might typically represent different environments such as production, preproduction, or different testing levels. By permitting a users to EJBROLE “administrator,” we are really allowing the user to be the administrator of not a single cell, but of all the cells in the environment. This access can be restricted by setting up a security domain, using the cell name as the security domain prefix, for example. By permitting users to EJBROLE “CELLNAME.administrator,” we can allow a user to be the administrator of a single cell.

12.2.4 How to interact with a local registry

To make the local registry work on your system, do the following tasks:

- Enable the local registry for your cell.
- Configure that registry to allow local authorization.
- Define EJBROLE and or GEJBROLE profiles.
- Enable EJBROLEs to use local identities for run-as caller settings.

WebSphere Application Server for z/OS Version 5 will now seamlessly interoperate with the local registry.

Enabling a local registry

To enable a local registry, you simply select localOS when activating global security (see 15.7, “Enabling global security” on page 489) for the WebSphere cell.

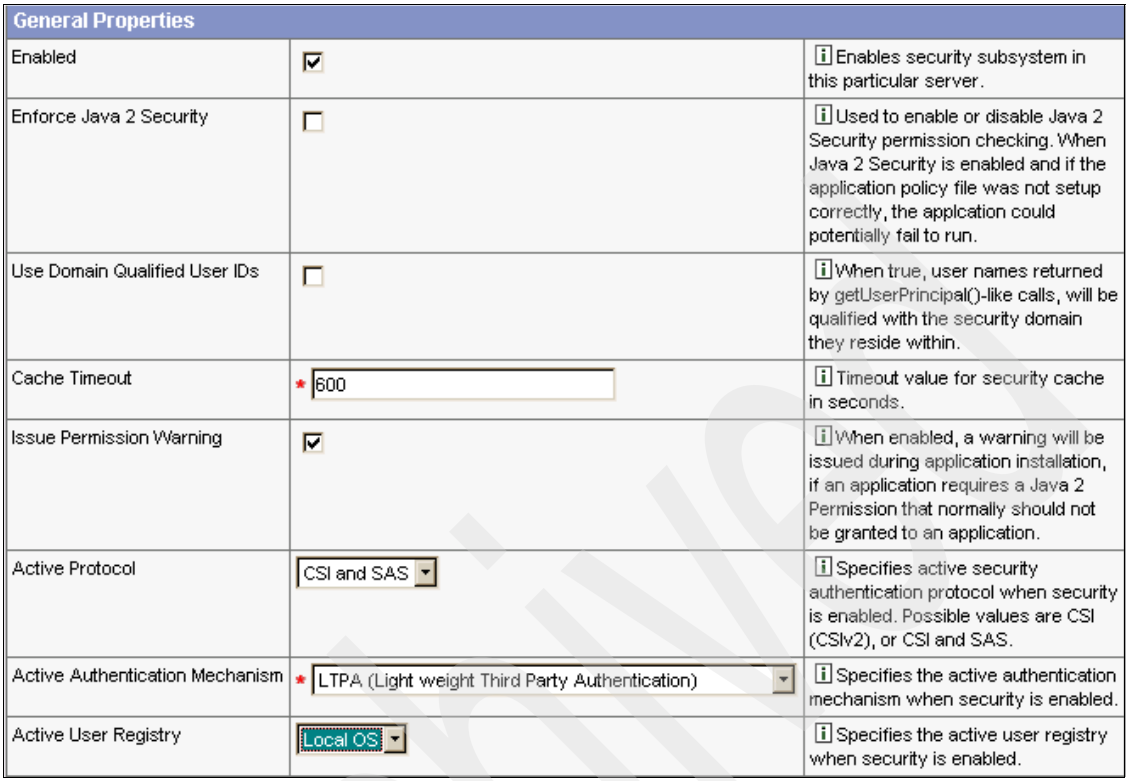


Figure 12-7 Enabling a local registry

Configure a local registry for local authorization

By enabling LocalOS as the active user registry, you can now configure WebSphere to use the SAF interface for authentication. For the purpose of authorization you have the ability to authorize against SAF or against authorization tables (ibm-application-bnd.xml). To use SAF for authorization you need to set com_ibm_security_SAF_authorization = true, as shown in Figure 12-8 on page 358.

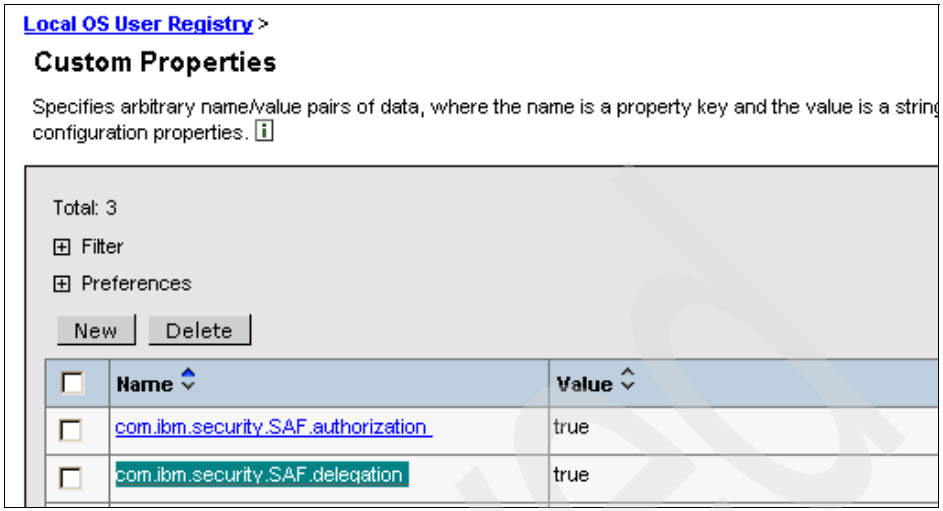


Figure 12-8 LocalOS User Registry properties for authorization

Activating EJBROLEs for J2EE security constraints

This section describes activating EJBROLEs for J2EE security constraints.

Set up EJBROLEs

This sample shows the RACF commands to set up an EJBROLE and how to grant access to it.

To define the EJBROLE profile, use this command:

```
RDEFINE EJBROLE Employee UACC(NONE)
```

To grant access to the defined role, use this command:

```
PERMIT Employee CLASS(EJBROLE) ID(USREMP) UACC(READ)
```

Set up GEJBROLEs

This sample shows the RACF commands to set up a GEJBROLE and how to grant access to it.

To define the GEJBROLE profile, use this command:

```
RDEFINE GEJBROLE CEOTASKS UACC(NONE)
```

To add logical roles to the group of roles, use this command:

```
RALTER GEJBROLE CEOTASKS ADDMEM(GrantPayRise, PromoteWorkers)
```

To grant access to all the defined roles, use this command:

```
PERMIT CEOTASKS CLASS(GEJBROLE) ID(THEBOSS) UACC(READ)
```

Enable EJBJOLEs to derive a local identity when running as caller

The value specified for APPLDATA is the RACF user ID that comes into effect if you use the RunAs role approach (see “Using run-as with an SAF registry” on page 55). If a method is configured to run as an EJBROLE, the user ID specified in the APPLDATA field is the user ID that the process runs as in WebSphere.

To define a run-as identity for a role, use this command:

```
RALTER GEJBROLE CEOTASKS APPLDATA(ROLEUSR)
```

This value will only be honored when the local OS registry custom variable com.ibm.security.SAF.delegation is set to true, as shown in Figure 12-8 on page 358.

13

Remote registries

This chapter provides an introduction to the remote registry solutions that are supported by WebSphere Application Server for z/OS Version 5. Use remote registries such as Lightweight Directory Access Protocol (LDAP) and custom user registries (CUR) to authenticate and authorize users to WebSphere applications. Topics covered in this chapter include:

- Authentication
- Authorization
- Identity mapping
- Tivoli Access Manager for e-business AMWAS module
- Trust association interceptor (TAI)
- Custom user registries (CUR)
- Sample scenarios:
  - Native authentication using LDAP on z/OS
  - Front-ending WebSphere with Tivoli Access Manager
  - Implementing custom user registries
  - Implementing trust association interceptor

13.1 Remote or pluggable registries for WebSphere

WebSphere Application Server for z/OS Version 5 has the capability to interact with remote registries including Lightweight Directory Access Protocol (LDAP) and custom user registries (CUR), rather than limiting the registry to the local OS registry as discussed in Chapter 12, “Local operating system registries” on page 343.

Remember: Even if you configure IBM WebSphere Application Server for z/OS to operate with a remote registry, it will still use the System Authorization Facility (SAF) to secure its own runtime, and WebSphere’s runtime accesses to operating system resources.

This functionality allows a more versatile WebSphere environment, making room for cross-platform integration by allowing the use of existing user registries and authorization tables with the security functions in WebSphere Application Server.

Why should you use a remote registry?

Remote registries offer the following benefits:

- Single sign-on solutions

  One user ID to maintain across multiple environments through LDAP or CUR

- Ability to incorporate RACF and DB2 into the security solution

  Ability to use the security of RACF for password encryption but add additional user attributes in DB2 by using LDAP native authentication (LNA)

- Cross-platform authentication mechanisms

  Ability to use registries on other platforms (Linux for IBM @server, Microsoft Windows NT®, and AIX®) as a means for securing IBM WebSphere Application Server for z/OS

- Identity mapping capabilities

  Using IBM Directory Integrator (IDI) to map one user to the equivalent user behind the scenes

13.1.1 Authentication with a remote registry

General authentication concepts for WebSphere Application Server are discussed in 11.2, “Authentication overview” on page 337.

When using a remote registry to protect WebSphere resources, authentication mechanisms are accessing remote data sources such as an LDAP server or a remote database to get access to user IDs and passwords, or alternate authentication mechanisms such as digital certificates and user mappings.The registries that are supported for WebSphere Application Server for z/OS Version 5.01 are local OS, LDAP, and custom user registries. Later releases of WebSphere Application Server for z/OS and OS/390 will also support the AMWAS module for Tivoli Access Manager for e-business.

13.1.2 Authorization with a remote registry

General authentication concepts for WebSphere Application Server are discussed in 11.4, “Authorization overview” on page 341. When using a remote registry to protect WebSphere resources, you will store and retrieve role-to-user mappings in the Authorization Table (ibm-application-bnd.xml). Group and user information that is stored in an LDAP server can be retrieved through WebSphere's mapping tools, as well as RACF group and user information when configuring against the Local OS registry.

However, when you are configuring a custom user registry, the ability to allow WebSphere to manage role-to-user mapping is no longer applicable. This happens because WebSphere has to allow for all possible data types to be presented as an authentication source, from a flat file to a foreign database. The functionality available will authenticate users and grant them access to those objects marked “any authenticated” within WebSphere Application Server for z/OS Version 5, or to those objects where access has been given explicitly in the user-role bindings (“Mapping Users to Roles” section in the administrative console). The bindings file, ibm-application-bnd.xmi, is not controlled by the CUR.

As an alternative to using a CUR, one can use an LDAP server to store user and group information or implement a front-ending solution with Tivoli Access Manager and implement a custom Cross Domain Authentication Service (CDAS) for authentication.

Using LDAP for user-to-group bindings

For people wanting to authenticate using a user ID/password combination that is stored in a database, or people wanting to authenticate using RACF user IDs and passwords, do the following tasks:

- Enable native authentication as described in the sample scenario below.
- Load all groups into the DB2 back end of the LDAP server.
- Configure WebSphere Version 5 for z/OS to authenticate against LDAP.

Using Tivoli Access Manager with a CDAS

- For those wanting to customize their logon procedures including authenticating against a file or foreign database
- For those wanting to store all group membership information into the HTTP header to be passed to an application for fine-grained authorization decisions being internal in an application

Cross Domain Authentication Service is an interface that supports custom authentication modules to be invoked by the plug-in for authentication. With this feature, you can write your own authentication modules. This feature is similar to Access Manager's WebSEAL-based implementation of CDAS.

13.2 The concept of identity mapping

Enterprise environments are complex in both design and functionality. As a given environment grows, the ability to maintain several different user registries becomes a requirement as well as an administrative headache. The ability to manage various identities through one central user registry or administrative tool is available by implementing one of the solutions described below.

13.2.1 Overview

User administration is comprised of managing several systems, servers, and applications, all with their own authentication and authorization requirements. As environments become increasingly complex, so does the ability to keep users, groups, and authorization tables in sync.

Application developers also see implementation issues with internal application security codes when multiple registries are being used for authentication and authorization. The problem of determining which user identity to run to achieve the desired authorizations for various servers and applications can cause much confusion and result in undesired security results within the application itself.

Solutions to managing multiple registries and user identities are found by implementing single sign-on solutions, installing and configuring an integration tool such as IBM Directory Integrator (IDI), or installing and configuring identity mapping software such as Enterprise Identity Mapping for z/OS.

13.2.2 Implementations and available products

In this section, we discuss the implementations and available products

Enterprise Identity Mapping for z/OS (EIM)

Enterprise Identity Mapping for z/OS is a software product that is based on LDAP directory technology, providing a directory to store EIM identities and allowing integration with existing user registries and authorization tables. This architecture (Figure 13-1) provides the capability to define a user and establish all related user entities and authorities for that user throughout an enterprise. EIM also provides a selection of APIs for application development, allowing you to dynamically interact with the registry through these APIs within a given application.

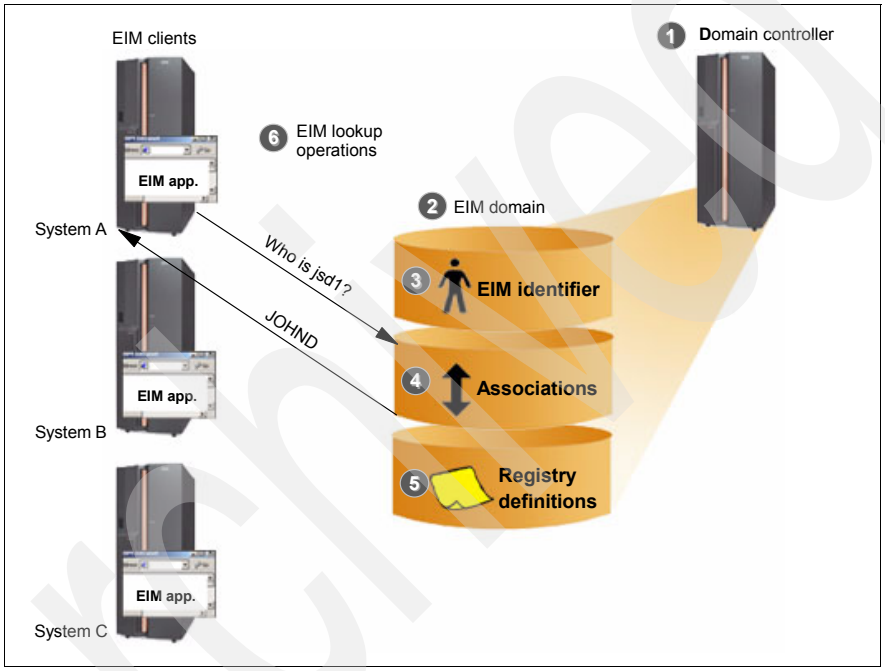


Figure 13-1 Enterprise identity mapping for z/OS

For more information about EIM for z/OS, see:

http://publibz.boulder.ibm.com/epubs/pdf/eima1100.pdf

IBM Directory Integrator (IDI)

IBM Directory Integrator is a Java-based product that allows events such as additions, deletions, or modifications of a given entry in a registry, database, or table, to kick start a series of administrative events throughout the enterprise environment. This tool is often used to keep all data between registries and proprietary authorization tables in sync. IDI also has methodologies to handle password synchronization if necessary.

13.3 Trust association interceptor

The trust association feature is, in essence, a point in the WebSphere authentication flow where an organization can insert their own code to achieve whatever authentication outcome they desire. Authentication takes place in a front-end authentication server or Remote Proxy Security Server (RPSS). WebSphere then accepts and acts on this authentication process, rather than driving its own authentication process. WebSphere uses a feature called *trust association interceptor* or *TAI* to make this possible. TAI is a Java class that implements the interface called TrustAssociationInterceptor.

How does TAI relate to your registry? TAI itself allows your infrastructure to offload only the authentication process to a front-end authentication server. TAI works with any registry you select, remote or local. It works best if the cell running TAI shares the registry with the authenticating server.

It is also important to note that the TAI is not an all or nothing approach. It can be coded so that only some requests are validated by it. That is, the TAI can decide that it will not perform a trust association operation on a request, in which case the authentication process returns to WebSphere for processing.

As the name implies, WebSphere is going to "trust" the result returned to it by the TAI. Therefore, WebSphere needs to trust the server that did the authentication. What the TAI must return to WebSphere is a valid user ID acceptable to the WebSphere Application Server user registry.

When should you enable the trust association interceptor?

- If you authenticate in the DMZ and can build trust between WebSphere and the authenticating server
- If your authentication server shares the same user registry as your WebSphere Application Server cell
- If you want to enable single sign-on solutions
- If you want to enable cross-platform solutions
- In Tivoli Access Manager for e-business scenarios with WebSEAL or caching proxy plug-ins

WebSEAL, as an authenticating front end, can be used as your reverse proxy:

- It hides your application server from the exposed network.
- It integrates with IBM HTTP Server plug-ins.
- It can be used to protect URIs.
- It offers coarse-grained and fine-grained authentication to target servers.

The WebSphere TAI on z/OS does not implement the trust relationship between the authenticating server and WebSphere Application Server itself. We recommend that you establish a level of trust by enforcing a trusted connection between these two parties. This connection could, for example, be a VPN or specially secured network segments.

TAI is coded as a Java class. In a WebSphere environment, there can be several TAI classes active. Each TAI class has these three main Java methods:

- isTargetInterceptor

  This method determines whether this TAI class is to process the request received. If it returns a value of false, WebSphere invokes the next trust association interceptor if any are specified, or processing returns to WebSphere, which then performs standard authentication processing. This method can use the HTTPServletRequest object, which contains the HTTP headers of the request. The method can use information obtained from this object to make its decision as to whether or not it will process the request.

- validateEstablishedTrust

  Having decided that this class is to process the request, this method determines if the request is valid. Rather than having WebSphere perform its own authentication, WebSphere relies on this method to verify that the request it is processing comes from a trusted source, where authentication was successful. This method can also use the HTTPServletRequest object. How it does this depends on how authentication has been done by the remote authenticating process. Typically, the remote authentication process will need to add some HTTP header tag to the request after it completes authentication. This method would then know to look for that HTTP header tag and validate it. If all connections use a VPN from a trusted source, it might not be necessary to check if requests come from a trusted source.

- getAuthenticatedUserName

  Having decided that the request is valid, the purpose of this method is to return a user ID that will then be used by WebSphere. The user ID needs to be a valid user ID in the WebSphere Application Server user registry. How this method determines what user ID to return will again be dependent on the information available to it in the request object and how the organization where the TAI is being implemented wants to handle this.

The validateEstablishedTrust method should verify that requests come from the RPSS and the getAuthenticatedUserName method should get the user identity and give it to WebSphere.

Typical TAI implementation

Typical TAI implementations include both an Remote Proxy Security Server (RPSS) and WebSphere Application Server. WebSEAL is an RPSS. WebSEAL is a popular authentication product because it acts as an SSL proxy, connects to various HTTP ports intelligently, and has TAI already built into its infrastructure. WebSphere Edge Server also has a Tivoli Access Manager plug-in to act as an RPSS. WebSphere Edge Server is the preferred solution for high-availability environments. This provides a trusted method of passing authenticated identities from one application server to another.

Typical flow of an HTTP request in a TAI-enabled environment

The following steps show the typical flow of an HTTP request:

1. A request is made for a protected WebSphere resource.
2. The RPSS sends back a request for authentication (user ID/password).
3. The user ID/password combo is authenticated against the RPSS-configured user registry.
4. WebSphere receives the request for the resource with user credentials included.
5. The TAI plug-in intercepts the request, extracts the user information from the RPSS added field, and passes it to WebSphere for authorization.
6. WebSphere maps the user to the required roles to determine whether or not to permit access to the requested resource.

Typical infrastructure

Figure 13-2 on page 369 illustrates a typical trust association interceptor scenario using Tivoli Access Manager WebSEAL.

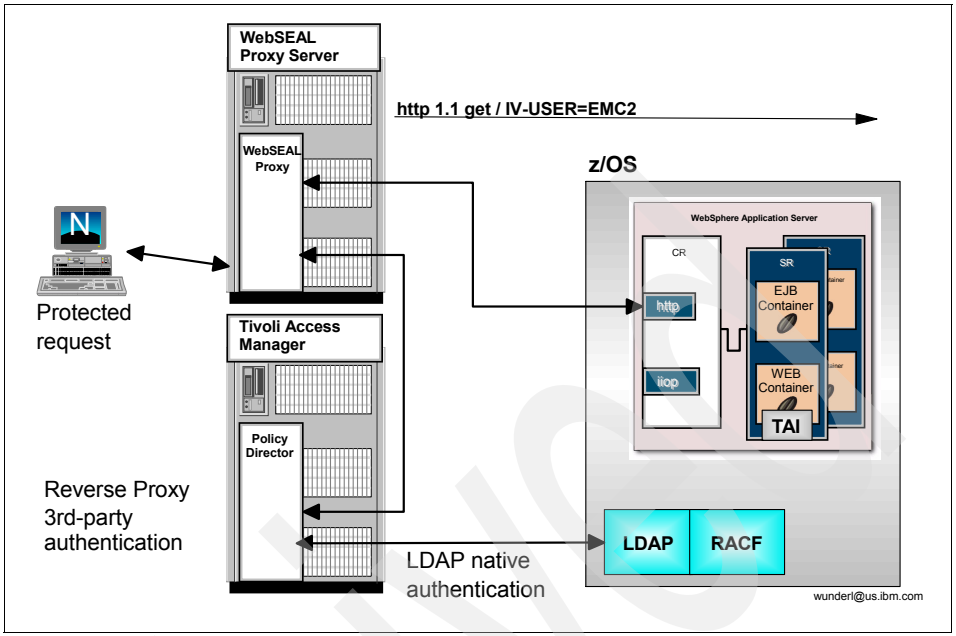


Figure 13-2 A typical trust association interceptor scenario

The implementation of the WebSEAL part is described in 13.6, “Sample scenario: Authentication with Tivoli Access Manager for e-business WebSEAL” on page 398.

Note: WebSEAL Versions 3.9 and later no longer pass user ID and password combinations to TAI. In this case, trust is established between WebSEAL and WebSphere through the use of SSL, and the TAI plug-in is no longer required.

The implementation of the TAI part is described in 13.8, “Sample scenario: Off-platform authentication with WebSEAL on Linux for zSeries using TAI” on page 409.

13.4 How to interact with a remote registry

The interaction with the remote registry happens seamlessly to the application and is done by WebSphere Application Server for z/OS Version 5. Nevertheless, it is the task of the system administration staff to set up and maintain the registry and to configure WebSphere accordingly.

WebSphere provides the functionality to use LDAP and CUR as remote registries, but how do you further use this technology? We offer several case scenarios for interacting with a remote registry including using AMWAS, WebSEAL and trust association interceptor, custom user registry, Tivoli Access Manager, and native authentication with RACF.

13.4.1 Tivoli Access Manager for e-business AMWAS module

How to integrate WebSphere Application Server for z/OS with Tivoli Access Manager for authentication and authorization is extensively described in Chapter 14, "IBM Tivoli Access Manager and WebSphere Application Server integration" on page 417.

The AMWAS module is an authorization tool that integrates the WebSphere security model with the access control capability found in Tivoli Access Manager for e-business. In this solution design, all role-to-user mappings are stored and maintained in the Access Manager ACL database. Access Manager provides administrative tools to maintain this data as well as the code that interacts with IBM WebSphere Application Server for z/OS, known as AMWAS. For a more detailed discussion, refer to the article *Integration of IBM Tivoli Access Manager 3.9 with WebSphere Application Server: A Complete Security Solution* at:

http://wilson.boulder.ibm.com/wsdd/techjournal/0206_miller/miller.html

Why should you enable this function?

- Centralized user and group management in Tivoli Access Manager for e-business user registry
- Centralized authorization management for J2EE applications
- Single sign-on capability for application servers

Programmatic security with AMWAS

Generic implementation of AMWAS code for fine-grained authorizations is done through AMWAS APIs within the application itself. For more information, check APAR OA02022. For more information about the programmatic interface, check the Web site above.

Declarative security with AMWAS

For authorization purposes, WebSphere accepts resource requests and determines which roles/groups are permitted to view these resources. Access Manager APIs are then called under the covers to determine whether or not the current user is in fact a member of that role or group. WebSphere receives this information from AMWAS and proceeds to either permit or deny access to resources accordingly.

The AMWAS credential check can be modified to meet the particular security needs of a given application. This allows an administrator to be as granular in ACL inheritance and permissions as necessary. However, these additional objects (such as server or host) do have to be created by the Tivoli Access Manager for e-business administrator. There is no pre-existing set of additional variables to be added to a credential for authorization.

Note: It is possible for AMWAS to work with a user registry that is not shared between the authentication server and WebSphere. In this case each WebSphere user would have to exist in both the Tivoli Access Manager for e-business user registry and the WebSphere user registry, or WebSphere needs to map the locally not available identities to available identities.

Typical infrastructure

A typical infrastructure with WebSphere and Tivoli Access Manager for e-business would look as depicted in Figure 13-3.

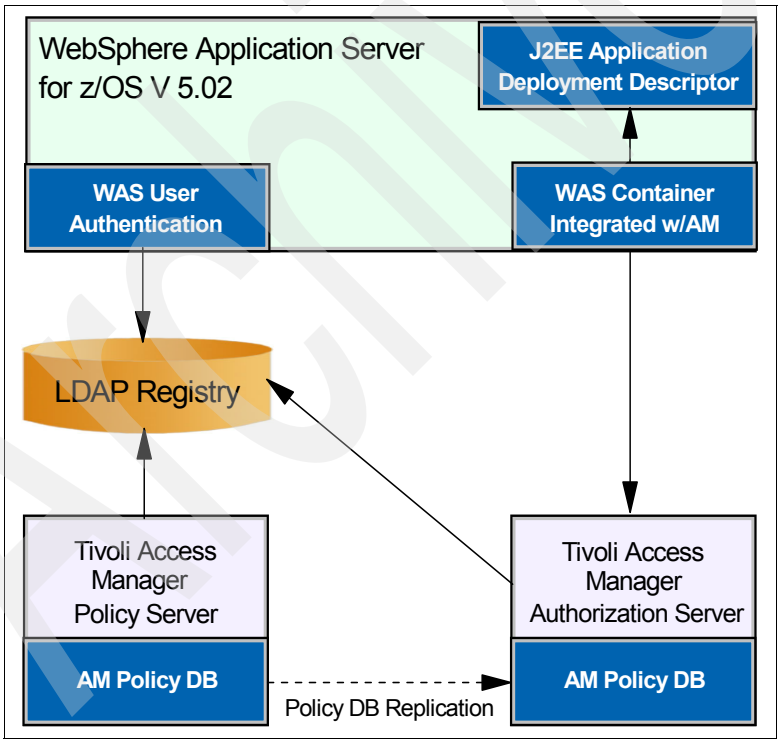


Figure 13-3 Tivoli Access Manager for e-business and WebSphere integration

13.4.2 Custom user registry

In WebSphere Application Server for z/OS Version 5, a user registry Java interface is provided to permit individual customers the ability to implement their own custom user registries as a means to secure WebSphere applications. The need to use this function could arise when a pre-existing security mechanism such as a database or user/group file is in place and it is not feasible to migrate this information to the local OS user registry. The user registry Java interface provided interacts with the pre-existing files to extract user identification and group information for authentication and authorization purposes.

Why should you enable the custom user registry interface?

- Custom user registries allows you to delegate the declarative and programmatic security requirements to a remote enterprise security registry. Thus, it allows WebSphere Application Server to integrate seamlessly into almost any security infrastructure.
- If you want to leverage a non-local registry for your non-local (Internet) clients.
- By providing the APIs, IBM enables you to develop a security solution that fits exactly to your needs.
- By providing this pluggable interface, IBM makes it possible for other vendors to enable their security solutions for WebSphere.

Typical custom user registry implementation

In a typical custom user registry solution, user and group data is pre-existing and is already in production in some cases. These registries could include a relational database, a flat file, or some other data repository outside of your z/OS operating system. The possibility of migrating or replicating data, or administering multiple registries is not always feasible. For these situations, WebSphere for z/OS Version 5 includes a service provider interface (SPI) that you can implement to interact with your configured registry. The SPI has a prepackaged set of methods that implement security for the application server.

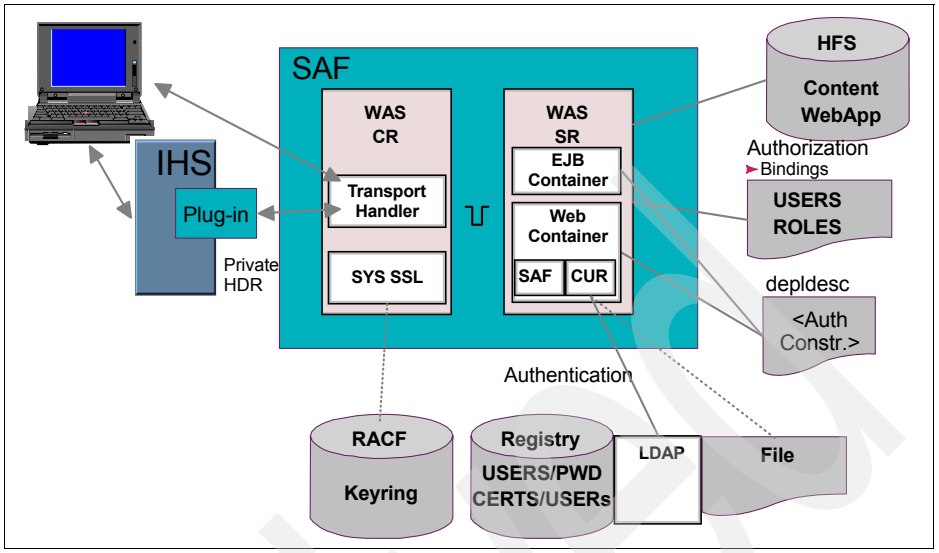


Figure 13-4 Custom user registry z/OS implementation

The configurable Lightweight Directory Access Protocol (LDAP) registry uses the same SPI, so most of the CUR considerations are valid for LDAP, too, except that you do not have to code the interface on your own. How to configure WebSphere to use LDAP as the active registry is described in “Configuring WebSphere to use LDAP as a registry” on page 393.

Custom user registry implementation hints

Consider the following custom user registry implementation hints:

- It is important to note that your custom user registry implementation is expected to be thread-safe[1].
- Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server components such as data sources, enterprise beans, and so on. If these dependencies exist, problems could arise due to the sequence of initializations in WebSphere. The security is initialized prior to the start of most other components. If this exists, make a code change to eliminate this dependency prior to implementing a custom user registry.

[1] Thread-safe means that a method that is called from multiple programming threads (or instances) executes without unwanted interaction between the threads. Thread-safe routines avoid the risk that one instance of a method will interfere and modify data elements of another instance with coordinated access to shared data.

- Be aware that the authenticated user ID does not get propagated into the EJB container when you implement the CUR interface. The principal will be the serverid, so the authorization table needs to be modified accordingly.

A typical file-based implementation consists of two files, one for users and one for groups. These files have to be stored and given accurate permissions in the HFS on z/OS, as well as being defined in the class path. Instructions about how to configure this setup are available in the case scenario depicted in Figure 13-5.

Typical file-based infrastructure

Figure 13-5 shows a typical file-based custom user registry.

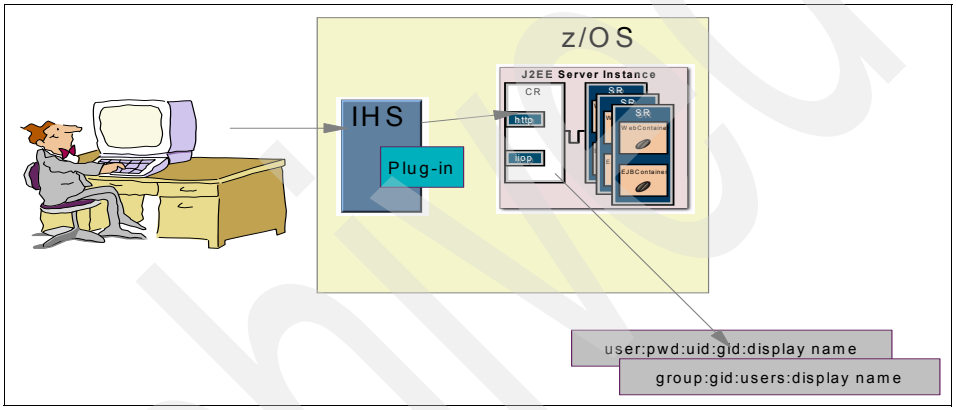


Figure 13-5 File-based custom user registry

The implementation of this scenario is described in 13.7, “Sample scenario: Implementing file-based custom user registry” on page 405.

13.5 Sample scenario: Using LDAP native authentication

LDAP native authentication (LNA) allows authentication to be done using RACF user IDs and passwords rather than storing user passwords in a DB2 table or a file in the HFS. Additionally, many companies that require full auditing capability such as banks or insurance companies, find that RACF IDs are required for all users to access secure data, and using this method of securing Web applications is easier. One would typically use the LNA with some LDAP-enabled authentication server as the front-end LDAP “client”. This configuration is also applicable for WebSphere servers running on distributed platforms, including Linux.

Note that LDAP on z/OS provides a special database schema called SDBM to work with the RACF database, but in order to work with an SDBM LDAP server you need an LDAP client application that is capable of working with SDBM. WebSphere and most HTTP Servers do not currently support SDBM. Instead, they work with the standard LDAP TDBM database schema. LDAP native authentication is based upon an LDAP TDBM database, so you need to implement TDBM, not SDBM, if you want to use LNA.

Another point to remember is that, although LNA provides the ability to authenticate using a RACF user ID and password, LDAP does not provide functions to manage all aspects of user ID and password management. If your RACF user ID is subject to a password expiry interval, when you sign-on directly to TSO or CICS, for example, using that user ID/password, the TSO or CICS subsystem provides messages and return codes to indicate the reason for a sign-on failure. When using LDAP and LNA, however, the reasons for a sign-on failure are not typically reported, so the user might not understand why the sign-on has failed.

In order to have more complete functions you need an LDAP authentication client with more function than that provided by basic authentication or form-based authentication, for example. You should consider Tivoli Access Manager to provide full function authentication to an LDAP server.

Why should you enable LDAP native authentication?

- You have a pre-existing LDAP server.
- You have the need for a central user registry (single sign-on).
- You want the ability to reuse RACF user IDs/passwords for intranet users, while using normal LDAP user IDs for Internet users.
- You are looking to front-end WebSphere Application Server for z/OS Version 5 with a security product like Tivoli Access Manager for e-business.

The following sample is what a typical LDAP user that is configured for native authentication might look like:

```
cn=sec1, o=itso
objectclass=top
objectclass=person
objectclass=ePerson
objectclass=ibm-nativeAuthentication
ibm-nativeId: SEC1USR
uid: SEC1USR
cn=sec1
sn=user1
```

Note that no userpassword attribute is stored in LDAP for RACF users. Notice also that we have chosen to use the objectclass `ePerson` under `person` and the

attributes `uid` or `ibm-nativeId`, or both, to set the RACF user ID associated with the LDAP user. When an LDAP user is defined as in this example, LDAP maps the LDAP common name (cn=) to a RACF user ID using the `uid` or `ibm-nativeId` attribute.

When using native authentication you can use either of the attributes `ibm-nativeId` or `uid` to define a unique name for the RACF user ID that you want to map to the LDAP user. However, note that WebSphere uses default searches based on `uid` and `cn`, so in practice, you must define the `uid` attribute for any LDAP native authentication users you want to use with WebSphere.

13.5.1 Enablement

Before you start we recommend that you obtain a tool that provides you with a graphical interface to the LDAP directory. You will find it hard to understand and see what is defined to LDAP without such a tool.

Try a search on the Internet for LDAP Browser or ldapbrowser. Following are some URLs you might find useful, but no recommendation is implied by providing these links:

```
http://www.iit.edu/~gawojar/ldap
http://www.pegacat.com/jxplorer
http://www.ldapadministrator.com
http://www.softwareshop.anl.gov/ldapbrowser.html
```

Installing LDAP on z/OS

LDAP on z/OS offers a configuration utility called *ldapcnf* to assist in the installation and customization of an LDAP server. To complete the installation process, complete the following instructions:

1. Create an HFS data set with about 5 tracks of space to hold the customized ldap.profile and the schema files you will customize and add to your LDAP server. Mount this HFS data set at etc/ldap.
2. Copy the following files from /usr/lpp/ldap/etc to /etc/ldap, which is the normal location for customized LDAP configuration files:
   - ldap.profile (this imbeds the following files when you run ldapcnf)
   - ldap.slapd.profile
   - ldap.db2.profile
   - ldap.racf.profile
3. Edit ldapcnf: Scroll to the bottom and change the lines that imbed ldap.db2.profile, ldap.slapd.profile, and ldap.racf.profile to reference the

/etc/ldap directory that contains the versions of these files that you will customize. You should change the lines to look like this:

```
${SOURCE_CMD} /etc/ldap/etc/ldap.slapd.profile
${SOURCE_CMD} /etc/ldap/ldap.db2.profile
${SOURCE_CMD} /etc/ldap/ldap.racf.profile
```

4. Customize the ldap.profile file to reflect your system and the configuration variables by following the detailed descriptions of each attribute in the profile.

Note: Some attributes in the ldap.profile file are required, but not given a default value. Make sure you read through the entire file, completing all required variables.

Most of the customization concerns data set names for parts of the z/OS system such as the libraries for language support or DB2, but there are some specific LDAP properties that you need to understand. These are discussed next.

TDBM_SUFFIX

To use LDAP native authentication you create a TDBM database, and so you need to choose and set the *suffix* that defines the *root* of the LDAP hierarchy. The suffix is normally in the form 'o=<your_organization>,c=<your_country>', so an IBM US system might code TDBM_SUFFIX='o=IBM,c=US'. (In the LDAP schema, o is an abbreviation for organization, and c for country.) If you want to create different *branches* under this organization, you can add organizationalunit (ou) under the organization. This is discussed in more detail later. In the examples below we refer to an LDAP with a suffix set to o=itso.

SDBM_SUFFIX

As explained earlier, LDAP native authentication uses the TDBM schema, so you do not have to define an SDBM database suffix. If you decide you want to use SDBM to provide an administrative interface to RACF from LDAP, you need to code the SDBM_SUFFIX parameter. You might code SDBM_SUFFIX cn=RACF,o=itso, for example.

ADMINDN and ADMINPW

You also need to define an LDAP administrator on variable ADMINDN and a password on ADMINPW. To start, you should define a purely LDAP user ID with an ADMINPW set. If you want to manage the LDAP administrator's user ID and password with RACF, you can add the ibm-nativeId attribute later, but if you plan to do this, it would probably be better to choose an LDAP administrator user ID that conforms to your RACF standards.

You might code: ADMINDN='cn=LDAP Administrator' for an administrator user ID managed by LDAP or ADMINDN='cn=LDAPADM' if you plan to manage this user ID through LNA.

OUTPUT_DATASET

This is the name of a z/OS partitioned data set (PDS) that will hold the customized configuration files and JCL to perform the LDAP configuration.

JOBCARDs

Customize the job cards for the jobs that will be created. Note that you need to escape certain special characters if you need to use them in the jobcards. For example, if you want one card to be a comment card (//*) you need to escape the asterisk (*) by adding a backward slash (\) before it, so it might look like this:

```
APF_JOBCARD_3="//\*'
```

5. Customize the ldap.db2.profile.

   Consult with your DB2 system administrator over the values you should set for the variables in ldap.db2.profile.

   Pay particular attention to the following variables in ldap.db2.profile:

   ```
   TDBM_DB2_LOCATION='LOC1'
   ```

   Set this to the correct LOCATION name for your DB2 system.

   ```
   TDBM_DB2_DNTRUNCSIZE='64'
   ```

   This variable is set to 32 by default but setting it to 64 can help performance when you have large numbers of entries in the database.

6. Customize the ldap.slapd.profile. Pay particular attention to the following variables in ldap.slapd.profile:

   ```
   LDAP_HOSTNAME=' '
   ```

   This variable is left blank by default and you must set the correct host name or IP address of your LDAP server.

   ```
   PORT='3389'
   ```

   And:

   ```
   SECUREPORT='6689'
   ```

   These are the ports LDAP will listen on (SECUREPORT listens for requests over SSL). You should also reserve the ports you choose (associate them with the LDAP started task name in the TCPIP profile).

7. Customize the ldap.racf.profile. Variables you set here will be used to set the GID and UID for the LDAP started task user ID.

8. Run the ldapcnf utility from the command line in z/OS OMVS. This utility is in /usr/lpp/ldap/sbin, so you should add this directory to your PATH or invoke the utility specifying the full path. For example:

   ```
   > /usr/lpp/ldap/sbin/ldapcnf -i /etc/ldap/ldap.profile
   ```

   The ldapcnf utility will generate a set of jobs into the MVS partitioned data set that was specified in the property OUTPUT_DATASET within the ldap.profile.

   It can take some time for ldapcnf to finish, especially if the default service class for the OMVS workload in WLM Workload Classification Rules has been given a low priority.

   The following steps refer to members within that customized data set.

9. Copy the LDAP server started task procedure from the output data set into the system PROCLIB. The default name for this started task is GLDSRV. If you change the name of the LDAP server started task, you need to update the job named ACF in the OUTPUT_DATASET to set the correct started task name in the commands that define the RACF STARTED profile for the LDAP server.

10. Copy the member named PROGxx to the system PARMLIB (normally SYS1.PARMLIB). (The job APF in the OUTPUT_DATASET will later run a SETPROG command to update the system APF library list, so you need member PROGxx in your system PARMLIB.)

11. Check that the LDAP SGLDLNK and DB2 SDSNLOAD libraries are APF-authorized and program-controlled. This is required when you want to use LDAP native authentication. Failure to do this will cause RACF errors such as this:

    ```
    ICH420I PROGRAM GLDSLAPD FROM LIBRARY GLD.SGLDLNK CAUSED THE ENVIRONMENT
    TO BECOME UNCONTROLLED.
    BPXM023I (LDAPSRV) 265
    ```

    You can program-control these libraries with RACF commands like the following:

    ```
    RALT PROGRAM * ADDMEM('GLD.SGLDLNK'//NOPADCHK)
    RALT PROGRAM * ADDMEM('DSN710.SDSNLOAD'//NOPADCHK)
    SETROPTS WHEN(PROGRAM) REFRESH
    ```

12. Review each of the following jobs. Pay particular attention to the jobs APF, RACF, and PGRMCNTL to ensure that the commands are correct for your environment. Your DB2 administrator should check job DBCLI and the SPUFI commands in member DBSPUFI before they are executed. After a careful review, run the jobs in the following sequence, remembering to check all of the output for successful return codes.

    a. RACF.

    b. APF.

c. DBCLI, make sure DB2 is started before submitting this job.

d. PGRMCTRL (if required).

13. Use the DB2 SPUFI tool to run the DB2 DDL statements in the member DBSPUFI.

14. Check that the SLAPDCNF member reflects your environment. (SLAPDCNF is the member referenced on the //CONFIG DD card of the GLDSRV started task JCL and contains the LDAP configuration options.)

15. Start the LDAP server using the GLDSRV started task. From SDSF you can start the server by entering /S GLDSRV.

16. When you see the phrase `slapd is ready for requests`, your LDAP server has started successfully.

Initializing the LDAP Directory

The next steps will assist you in building the LDAP schema and loading the LDAP directory with your suffix and the users you need to define.

1. Copy the following files to your /etc/ldap directory:
   - /usr/lpp/ldap/etc/schema.user.ldif
   - /usr/lpp/ldap/etc/schema.IBM.ldif
2. Edit *both* the above files by changing the line "cn=schema,<suffix>" to reflect the suffix that is defined on variable TDBM_SUFFIX in your ldap.profile. For example:

   ```
   dn: cn=schema,o=itso
   ```

3. Use the **`ldapmodify`** command to load the schema files into the directory:

   ```
   ldapmodify -h x.x.x.x -p 3389 -D "cn=LDAP Administrator" -w secret -f
   /etc/ldap/schema.user.ldif
   ldapmodify -h x.x.x.x -p 3389 -D "cn=LDAP Administrator" -w secret -f
   /etc/ldap/schema.IBM.ldif
   ```

   Note that `x.x.x.x` represents the host name or IP address of the LDAP server and the `cn=` and `-w` options set the user and password of the LDAP administrator defined earlier.

> **Tip:** LDAP offers an **`ldapmodify`** command or an **`ldapadd`** command. You can also use **`ldapmodify`** to add new entries by using the `-a` option or by coding `changetype:add` as a line in the LDIF file.

4. Create a file containing LDAP configuration definitions, which will add an organization entry (and maybe a country entry) for your suffix into the directory. (These definitions are called LDIF definitions and the file is called an “LDIF file.”) This LDIF file can contain any definitions you like, each block of related statements separated by a blank line, but at this stage, it is best just to add the suffix. For example, if your suffix is o=itso, you might create a file called suffix.ldif that looks like this:

```
dn: o=itso
objectclass: organization
objectclass:top
o: itso
```

5. Use **ldapadd** to add the suffix o=itso from the suffix LDIF file to the directory:

```
ldapadd -h x.x.x.x -p 3389 -D “cn=LDAP Administrator” -w secret -f
suffix.ldif
```

6. Execute the following **ldapsearch** command as an IVP to check that LDAP is set up correctly:

```
ldapsearch -h x.x.x.x -p 3389 -V 3 -s base -b ““ “objectclass=*”
```

Enabling LDAP native authentication

To enable LDAP native authentication, complete the following steps:

1. If you are running z/OS 1.3 or later, the schema.IBM.ldif will already contain the necessary attributes to support native authentication. Look in the comments at the start of your schema.IBM.ldif to see if it includes NativeAuthentication.ldif. If not, check whether you are looking at the correct version of /usr/lpp/ldap/etc before performing the next step.
2. If your schema.IBM.ldif file contains definitions for native authentication, skip this step. Otherwise, copy the following file to your /etc/ldap directory:

   /usr/lpp/ldap/etc/NativeAuthentication.ldif

   Edit this file by changing the line “cn=schema,<suffix>” to reflect the suffix that is defined in your configuration file.

   Use the **ldapmodify** command to load the schema file into the directory:

```
ldapmodify -h x.x.x.x -p 3389 -D “cn=LDAP Administrator” -w secret -f
/etc/ldap/NativeAuthenication.ldif
```

3. Modify the LDAP configuration file to include the following properties in the TDBM section. (The LDAP configuration file will be the member called SLAPDCNF within the output data set customized by ldapcnf.)

```
useNativeAuth selected
nativeAuthSubtree o=itso
nativeUpdateAllowed on
```

These definitions say that you want to enable LDAP native authentication only for users defined within the LDAP hierarchy under o=itso. Because o=itso is the suffix in our examples here, the definitions above have the same effect as nativeAuthSubtree **all**.

If you want to set up LNA for a part of your total organization, consider creating an organizationalunit under o=itso. For example, you might create `ou=WAS` under o=itso and then set `nativeAuthSubtree ou=WAS,o=itso`.

An LDIF file to do this might look like this:

```
dn: ou=WAS,o=itso
objectclass: top
objectclass: organizationalUnit
ou: native
```

4. Create a file containing entries that will update the ACL protecting the nativeAuthSubtree so that each native authentication user ID will be authorized to change their own password. For example, if your nativeAuthSubtree is set to o=itso, you might create a file called newaclcnthis that looks like this:

```
dn:o=itso
changetype:modify
add:x
aclEntry:access-id:cn=this:critical:rwsc
aclPropagate:TRUE
```

5. Use the **ldapmodify** command to update the ACL:

```
ldapmodify -h x.x.x.x -p 3389 -D "cn=LDAP Administrator" -w secret -f
/etc/ldap/newaclcnthis.ldif
```

6. At this point it is a good idea to start/stop LDAP and use an LDAP browser to check that the LDAP structure looks the way you want it to look.

Tip: Your LDAP server's TDBM database obviously has a dependency on the availability of DB2, so you need to make sure that the automation that starts all the system tasks after an IPL does not try to start the LDAP server until DB2 has completed initialization. The following error messages in LDAP are typical when LDAP is started before DB2 has finished initialization:

```
GLD0154E Error code -1 from odbc string: "SQLAllocConnect "  .
GLD0155E ODBC error, SQL data is:  native return code=-99999, SQL
state=58004, SQL message={DB2 for OS/390}{ODBC Driver}  SQLSTATE=5
8004  ERRLOC=2:170:9
  CAF "CONNECT" failed using DB2 system:DSN1
   RC=08 and REASON=00f30002
```

7. If you want to add users that were previously defined in another LDAP directory, you should export those entries from that LDAP server now. The LDAP Browser/Editor tools usually provide an export function, but you should read the documentation for your existing LDAP directory because it might provide utilities to do this, too.

8. If you have an LDIF file of user definitions exported from another LDAP V3 directory, you could run ldapadd against that LDIF file now to add the users to the LDAP z/OS directory. If these users are defined as *persons*, you will need to add the object classes ePerson and ibm-nativeAuthentication to each entry. You should also set the attributes ibm-nativeId and uid to the RACF user ID you want to associate with each LDAP user. To modify existing entries, you could use an LDIF file called nativeupdate.ldif that might look like this:

```
dn: cn=testu1, o=itso
changetype: modify
add: x
uid: TESTU1
ibm-nativeId: TESTU1
objectclass: ePerson
objectclass: ibm-nativeAuthentication
```

   Update users to be LNA users by using an **ldapmodify** command that might look like this:

```
ldapmodify -h x.x.x.x -p 3389 -D "cn=LDAP Administrator" -w secret -f
/etc/ldap/nativeupdate.ldif
```

9. If you are not importing existing user definitions, you should define the users you need to LDAP now. You will probably want to make sure the users are defined to LDAP as LNA users. You can optionally choose to add other person attributes such as telephone numbers.

   Create an LDIF file that looks something like this:

```
dn: cn=John Doe,o=itso
cn: John Doe
sn: Doe
telephonenumber: (03) 9335 2114
objectclass: ibm-nativeAuthentication
objectclass: ePerson
objectclass: person
objectclass: top
ibm-nativeId: JDOE
uid: JDOE
```

   For WebSphere Application Server for z/OS, you also need to define LDAP users for the WebSphere control region user ID, the servant region user ID, and the default guest user ID. These can be defined as LNA users because when you configure a WebSphere node these user IDs are normally defined as RACF users. Create an LDIF file called newserverids.ldif, for example,

similar to the one shown in Example 13-1, where V5P1CRU is the RACF user ID of the WebSphere control region, V5P1SRU is the RACF user ID of the WebSphere servant region, and V5P1GST is the WebSphere Guest user ID.

Example 13-1 The newserverids.ldif file

```
dn: cn=V5P1CRU,o=itso
changetype:add
objectclass: ibm-nativeAuthentication
objectclass: ePerson
objectclass: person
cn: V5P1CRU
sn: V5P1CRU
uid: V5P1CRU
ibm-nativeId: V5P1CRU

dn: cn=V5P1SRU,o=itso
changetype:add
objectclass: ibm-nativeAuthentication
objectclass: ePerson
objectclass: person
cn: V5P1SRU
sn: V5P1SRU
uid: V5P1SRU
ibm-nativeId: V5P1SRU

dn: cn=V5P1GST,o=itso
changetype:add
objectclass: ibm-nativeAuthentication
objectclass: ePerson
objectclass: person
cn: V5P1GST
sn: V5P1GST
uid: V5P1GST
ibm-nativeId: V5P1GST
```

10. Use **ldapadd** to add the WebSphere region user IDs to the directory:

```
ldapadd -h x.x.x.x -p 3389 -D "cn=LDAP Administrator" -w secret -f
/etc/ldap/newserverids.ldif
```

11. Now define your WebSphere administrators and the WebSphere configuration group to LDAP.

    You will probably want to make most WebSphere administrator user IDs LNA user IDs. During initialization, the WebSphere servant region checks that the user group that contains the WebSphere configuration users is defined to LDAP and that the WebSphere administrator user IDs are members of that group. So when you define the WebSphere administrator user ID you should also define the configuration user group in object class groupOfNames.

Note that you need to add the control region and servant region started task user IDs as members of the WebSphere administration group.

Later, when you configure WebSphere to use the LDAP User Registry, you need to specify a *server ID* on the LDAP User Registry Configuration panel. Whatever you specify as the server ID must also be a WebSphere administrator.

For example, assuming your WebSphere administrator user ID is WASADMIN, the shell WebSphere administrator user ID is WSADMSH, the LDAP server ID is WSSERV, and the configuration group is WASCFG, and that you are using nativeAuthSubtree o=itso, create an LDIF file called newadminuser-ldif that looks like the file shown in Example 13-2.

Example 13-2 LDIF file

```
dn: cn=WASADMIN,o=itso
cn: WASADMIN
sn: WASADMIN
objectclass: ibm-nativeAuthentication
objectclass: ePerson
objectclass: person
objectclass: top
uid: WASADMIN
ibm-nativeId: WASADMIN

dn: cn=WSADMSH,o=itso
cn: WSADMSH
sn: WSADMSH
objectclass: ibm-nativeAuthentication
objectclass: ePerson
objectclass: person
objectclass: top
uid: WSADMSH
ibm-nativeId: WSADMSH

dn: cn=WASSERV,o=itso
cn: WASSERV
sn: WASSERV
objectclass: ibm-nativeAuthentication
objectclass: ePerson
objectclass: person
objectclass: top
uid: WASSERV
ibm-nativeId: WASSERV

dn: cn=WASCFG,o=itso
objectclass: top
objectclass: groupOfNames
```

```
member: cn=WASADMIN,o=itso
member: cn=WSADMSH,o=itso
member: cn=WASSERV,o=itso
member: cn=V5P1CRU,o=itso
member: cn=V5P1SRU,o=itso
```

Use **ldapadd** to add the administrator user IDs and the WebSphere configuration user group to LDAP:

```
ldapadd -h x.x.x.x -p 3389 -D "cn=LDAP Administrator" -w secret -f
/etc/ldap/newadminuser.ldif
```

12. Now, you need to consider the user requirements of your applications. Your applications might make use of roles, JAAS aliases, or both.

    If your application uses run-as roles, you need to define to LDAP the users that you want to associate with those roles.

    If your application uses a J2CA Resource Adaptor to communicate with enterprise information systems, such as DB2, CICS, or IMS, the connection factories might be configured to use JAAS authentication aliases. If JAAS authentication aliases will be used, you need to define the user ID and password associated with the JAAS authentication alias to LDAP.

    Consider the following information:

    – When using a non-LocalOS user registry (that is, non-SAF), if your application contains an EJB or servlet that uses J2CA to connect to an EIS system, and if that EJB or servlet is defined with a run-as role, note that at Service Level W501000, the user associated with the role will not be passed to the J2CA connector.

      This limitation should be lifted by future service. In the meantime, any roles that your application uses still need to be mapped to users and those users defined as LDAP users. You could define a role user as an LDAP native authentication user, but then you will need to define the user ID to RACF with a non-expiring password.

    – If you intend to use any JAAS authentication aliases (J2C authentication data entries), the user IDs associated with the JAAS alias can be defined to LDAP as native authentication user IDs and with a password maintained in your SAF-based user registry. If you use a JAAS authentication alias for a local mode connection to CICS from WebSphere Application Server for z/OS, this password does not need to be correct, because only the user ID is passed to CICS by the CICS Transaction Gateway. But if you are using WebSphere on non-z/OS platforms, you need to code the correct password when you define the alias. With the IMS Connector, you always need to code the correct password.

– With a non-z/OS WebSphere and a container-managed J2CA connection, the server ID is flowed to the EIS. If you use a container-managed JAAS authentication alias, the alias user ID and password are flowed to the EIS instead. If you want to pass some identity that represents the authenticated user, your application must take responsibility for this, so you need to use a component-managed (also known as application-managed) connection (res-auth=application in the deployment descriptor XML). This is selected on the Authentication option of the resource reference for the J2CA connection in the application's deployment descriptor.

  For example, your application could look at the authenticated user ID using getUserPrincipal, and then use that user ID (or choose some other valid RACF user ID) to set on the user property of the ConnectionSpec object when making a J2CA request. With a non-z/OS WebSphere, you must also provide a password on the ConnectionSpec so that the EIS can authenticate the user. There is one exception to this rule. If you are connecting to CICS using the CICS Transaction Gateway and deploy the CICS Transaction Gateway daemon on the z/OS system, you can choose to configure the CICS Transaction Gateway daemon so that it does not to re-verify the user ID/password on the ECI request. You would also need to establish a trust relationship between the CICS Transaction Gateway and WebSphere, because, in effect, this configuration would amount to a signon without password to CICS. That trust could be established by configuring the CICS Transaction Gateway daemon to perform SSL client authentication of the WebSphere server. Trust between the CICS Transaction Gateway daemon and CICS can be established using MRO BIND security (RACF DFHAPPL class profiles).

Note: WebSphere on z/OS can pass the Java thread security identity to the J2CA connector. This is known as *thread identity support* for J2CA. However, it is currently not possible to pass that user ID to the J2CA connector when using a non-RACF user registry such as LDAP. Currently, (fix level W502000 and later) thread identity only works in WebSphere Application Server for z/OS when you use a local OS registry (RACF). Even though you might successfully authenticate with a valid RACF user ID/password through LDAP native authentication and the user ID you obtain using getUserPrincipal is correct, the J2CA connector will say that the credential for the user is invalid.

With WebSphere Application Server for z/OS, your application can use a J2CA resource reference with Authentication=Application (component-managed sign-on) and pass the user ID and password to the J2CA adaptor. But, as mentioned earlier, the password will not be flowed to CICS by the CICS Transaction Gateway. The IMS Connector will flow both the user ID and password, and these will be authenticated.

Create an LDIF file called newroles.ldif, for example, which might look like that shown in Example 13-3, to define role users SERVMAN and EJBMAN. This example also adds a JAAS authentication alias user called JAASA.

In this example, suppose that the servlets run with a role associated with cn=SERVMAN and EJBs run with role cn=EJBMAN. These users will be defined as LDAP native authentication users. However, remember the earlier discussion about the fact that WebSphere Application Server for z/OS currently cannot propagate the current thread identity to a J2CA connector, so at present, there is little value in defining role users as LNA users when using a container-managed sign-on (res-auth=container) on a J2CA connection.

However, if your J2CA connection factory is using a JAAS authentication alias, the alias user ID can successfully be defined to LDAP as a native authentication user ID and that user ID or user ID/password *will* be passed to the J2CA connector. WebSphere will authenticate the JAAS user ID/password.

Example 13-3 LDIF file

```
dn:cn=SERVMAN,o=itso
changetype:add
objectclass:top
objectclass:person
objectclass:ePerson
objectclass:ibm-nativeAuthentication
ibm-nativeId:SERVMAN
uid:SERVMAN
```

```
sn:SERVMAN
cn:SERVMAN

dn:cn=EJBMAN,o=itso
changetype:add
objectclass:top
objectclass:person
objectclass:ePerson
objectclass:ibm-nativeAuthentication
ibm-nativeId:EJBMAN
uid:EJBMAN
sn:EJBMAN
cn:EJBMAN

dn:cn=JAASA,o=itso
changetype:add
objectclass:top
objectclass:person
objectclass:ePerson
objectclass:ibm-nativeAuthentication
ibm-nativeId:JAASA
uid:JAASA
sn:JAASA
cn:JAASA
```

Use **ldapadd** to add the user IDs for the roles and the JAAS alias to LDAP:

```
ldapadd -h x.x.x.x -p 3389 -D "cn=LDAP Administrator" -w secret -f
/etc/ldap/newroles.ldif
```

13. If your application uses run-as roles, you need to bind the roles to a real LDAP user ID. You can do this in the deployment descriptor of the application (application.xml) on the Bindings tab. See Figure 13-6 on page 390 for an example of how to do this using the AAT. Alternatively, you can map roles to user IDs/passwords at application deployment time.

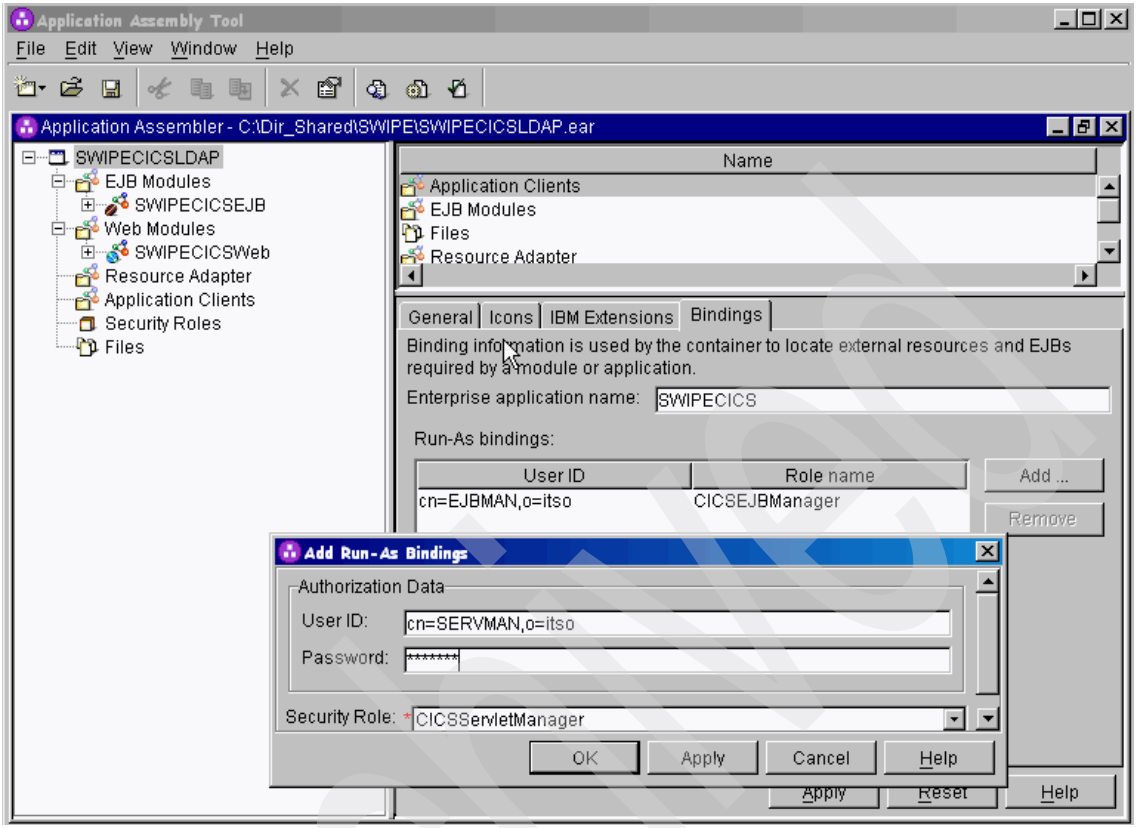


Figure 13-6 Using the AAT to set role to User ID and Password mappings

14. If you have updated the application deployment descriptor, you will now need to update the application in WebSphere. Use the WebSphere administrative console and navigate to **Application → Enterprise Applications**. Select your application and click **Stop**. When the application is stopped, select the application and click **Update** to redeploy the application.

    Pay attention to steps 7 and 8. In step 7, you might want to have the **All Authenticated Users** option selected for each role to allow any authenticated user to run in that role. For example, step 7 should look like Figure 13-7 on page 391. This is a screen shot from a system with user IDs defined under ou=WAS,o=IBM,c=US. In step 8, there are user IDs associated with each role and so these always appear under the Mapped Users column of step 7.

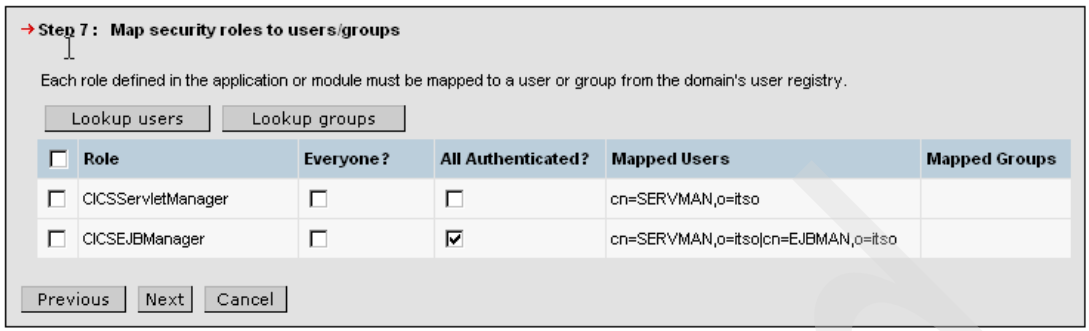


Figure 13-7 Step 7 of application deployment: Check who is authorized to the roles

If you want to permit specific users rather than check the All Authenticated box, click **Lookup Users** or **Lookup Groups**. This will open a window like that shown in Figure 13-8 on page 392. You can search the LDAP directory for all users or groups, highlight the ones you want to permit to the role, and then click the **>>** arrows in the center of the panel to move the users to the Selected list on the right. When you have selected all the users you want to permit to the role, click **OK**.

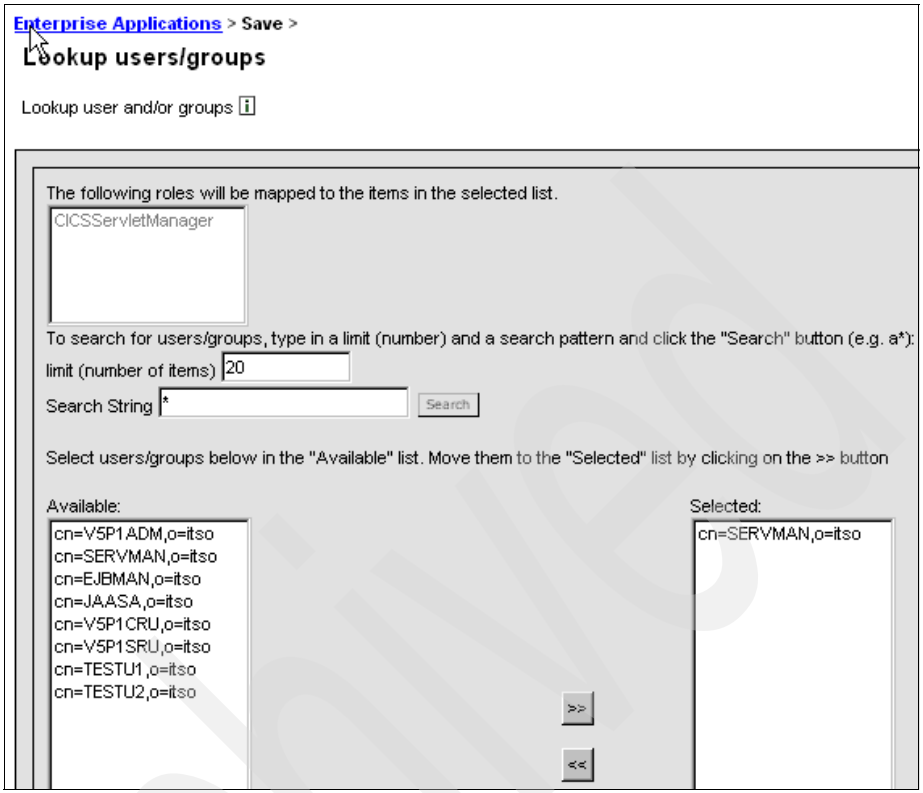


Figure 13-8 Step 7 of application deployment: Specific authorization to roles

15. In step 8, you need to check that the correct user ID is associated with the role. Because we are using an LDAP registry, the user ID will appear as an LDAP distinguished name, even if you have defined that user ID to be an LNA user ID.

    You can change the role to user mapping or the password for the role user ID by selecting the role and clicking the button to apply role mappings. Any user IDs you have bound to roles must also be defined to LDAP. You must make sure the password set in the deployment descriptor matches that defined to LDAP (or RACF when using native authentication).

    If you change the role to user ID mapping in step 8, you might have to go back to step 7 to ensure that the new user ID you have associated with the role is authorized to the role.

Configuring WebSphere to use LDAP as a registry

Enabling LDAP as the User Registry for WebSphere Application Server on all platforms is done through the WebSphere administrative console.

Before enabling global security in WebSphere, the first thing you need to do is configure an authentication mechanism. Next choose a user registry and configure it. Then, enable global security.

1. To configure the authentication mechanism, from the navigation bar of the WebSphere administrative console, click **Security** → **Authentication Mechanism** → **LTPA**. With an LDAP User Registry you will normally want to us LTPA.

   Set a password (and verify password) that will be used to encrypt the LTPA keys. At this time you do not need to click **Generate Keys** or set the name of the keyfile for import/export. When you click **OK**, WebSphere generates keys because this is the first time you are enabling LTPA.

   If you are enabling LTPA for a Network Deployment cell, you should also select and configure the single sign-on (SSO) options.

   Then click **OK**. You should be passed to the Enable Global Security panel where you should just click **OK** at this time. The Global Security panel will be configured after you have finished configuring the LDAP User Registry.

2. To select LDAP as a User Registry, from the navigation bar of the WebSphere administrative console, click **Security** → **User Registries** → **LDAP**.

   Fill in the required information. The fields you need to complete are:

   – Server Id: `cn=WASSERV, o=itso`
   – Server Password: `secret`
   – Type: **Secureway**
   – Host: `9.1.1.1`
   – Port: `3389`
   – Base Dn: `o=itso`
   – Bind Dn: `cn=LDAP Administrator`
   – Bind Password: `secret`
   – Reuse Connection: Select
   – Ignore Case: Select

Note: Note that `Server Id` (`WASSERV` above) can be any LDAP user. The user ID/password is checked during server initialization when WebSphere contacts the LDAP User Registry. It is possible to use an LDAP user ID that maps to a RACF user ID using ibm-nativeId, and in that case you must define WASSERV to RACF and provide the correct RACF password on the LDAP User Registry configuration panel. If that user ID is subject to password expiry or if the password is incorrect, your WebSphere server will fail to initialize. For this reason, if you want to set a RACF user ID for server ID here, you need to choose a user ID with a password that is non-expiring.

You might think that server ID suggests that you should use the user ID of the WebSphere server's control region, but the RACF user ID for the WebSphere Application Server for z/OS control region is set up without a password, so if you decide to use that user ID in server ID, it will not work if the user ID is defined to LDAP as an LNA user ID. Earlier both the WebSphere control region and servant region user IDs were defined to LDAP, so one approach is to set the WebSphere control region user ID as the server ID but to define this user ID to LDAP as an LDAP user with an LDAP password.

Usually, if you want to use LNA for users, it is probably best to use an LNA user ID/password for the server ID. Therefore, we recommend that you define a new specific RACF user ID/password for server ID rather than use the WebSphere control region or administrator user ID in the Server Id field.

Earlier you should have added this user ID (WASSERV in our example) to LDAP. Check that in LDAP you can see WASSERV defined as an LNA user ID and that it is a member of the WebSphere configuration group. Later, we will use the WebSphere administrative console to define WASSERV as an administrator.

3. Click **OK**. This should cause a link to the Global Security panel. If you do not get linked to the Global Security page, from the navigation bar, select **Security** → **Global Security**.

   Looking under the Configuration tab, set the value for Active User Registry to LDAP.

   Click **OK**.

4. Now again navigate to **Security** → **User Registries** → **LDAP** and click the link to Advanced LDAP Settings at the bottom of the page under Additional Properties. On Advanced LDAP Settings you should not need to change anything, but it is important to understand the masks for the User and Group search filters.

   When using Secureway LDAP server as a user registry, WebSphere will search for users using uid with the objectclass ePerson. And WebSphere

searches for groups using cn and objectclass groupOfNames. You should probably leave the search filters like this to take advantage of Secureway search optimizations based on these attributes. This means that when you define users or groups to LDAP you need to ensure that users are defined with the uid attribute and objectclass ePerson, and groups are defined with a cn and objectclass groupOfNames. (This is why the earlier examples used these attributes and object classes.)

If you use the default search filters for a Secureway LDAP server and have defined an LNA user with the uid attribute, LDAP will correctly locate the user during authentication and will check the password with RACF. Assuming authentication succeeds, the value specified for the uid will become the current Java thread identity.

If everything is correct, you should be able to log in to WebSphere when prompted using just the LDAP user ID rather than the full name (for example, cn=testu1,o=itso). If this fails, check the WebSphere servant joblog for trace messages relating to the failure.

5. Click **OK**. If you are placed in the Global Security page, click **Apply** or **OK**.
6. Save your configuration.
7. Configure the LDAP server ID you defined when configuring the LDAP user registry as a WebSphere administrator. (We used the LDAP server ID *wasserv* in the example earlier).

   Navigate to **System Administration** → **Console Users**, and then add WASSERV as an administrator by clicking **Add** and completing the next panel. Click **OK**.
8. Configure the LDAP server ID you defined when configuring the LDAP user registry so it has authority to write and delete resources in the CosNaming server. (We used the LDAP server ID wasserv in the example earlier). This is required because certain operations relating to naming take place when servers are started and stopped so that they run under the server ID.

   Navigate to **System Administration** → **CORBA Naming Service Users**.

   There appear to be no users defined on the next panel, but click the small down arrow and you should see an entry for your server ID granted Cos Naming Read. See Figure 13-10 on page 396.

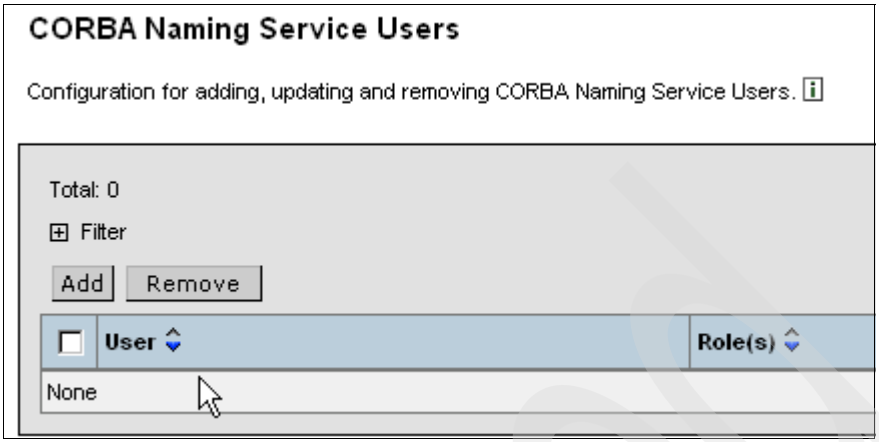


Figure 13-9 Expanding list to see users granted COSNaming roles

After clicking the small down arrow, you should see a list like this:

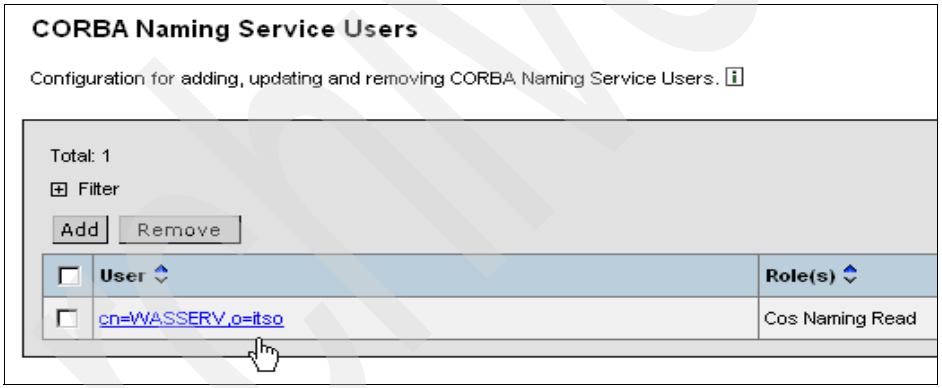


Figure 13-10 The expanded list of users granted CosNaming roles

Select the box next to the entry for cn=WASSERV,o=itso and click **Remove**.

Now, click **Add**.

Enter the user cn=WASSERV,o=itso and then highlight all the roles shown by clicking the first role in the list (Cos Naming Read) and then clicking the last role while pressing Shift. The display should look like Figure 13-11 on page 397.

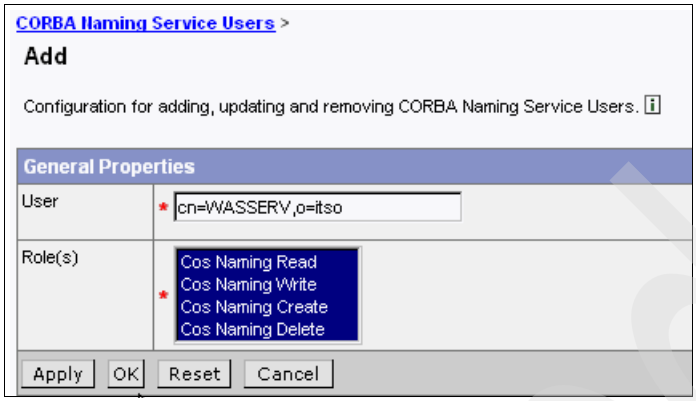


Figure 13-11 Adding all roles to the server ID (we use cn=WASSERV,o=itso)

Click **OK**.

9. Save your configuration and restart the server.

> **Note:** The need to specifically authorize users to the CosNaming server for any operations other than read was introduced with fix level W502000. Check that other WebSphere administrator user IDs have full write and delete access.

13.5.2 Verification

There are two tests you need to perform to verify that WebSphere is successfully configured to use LDAP.

First, go to the WebSphere administrative console. A login with password should now be required. In the examples above our WebSphere administration user ID WASADMIN was defined to LDAP as an LNA user ID, so WASADMIN should be used as the login ID with its RACF password. If the WebSphere administrative console comes up, native authentication is now protecting WebSphere for z/OS. This test also checks that the WebSphere administration user is correctly defined to LDAP as an LNA user with the correct attributes to match the LDAP search filters that WebSphere will use (as set on the Advanced LDAP Settings panel.)

Use SDSF to check the job log of the WebSphere Application Server for z/OS control region and server region for any trace error messages. If your login fails and you see a message like the following:

```
BBOO0220E SECJ0336E: Authentication failed for user WASADMIN...
.... No user WASADMIN found
```

This probably indicates that the WASDMIN LDAP user was not given the uid attribute or you changed the User Filter mask on the LDAP Advanced Settings panel to something other than uid=%v.

If the login to the administrative console worked and you have defined all the user IDs you need to LDAP (WebSphere control region, servant region, LDAP server ID, WebSphere administrators and configuration group, application users, user IDs for roles, and user IDs for JAAS aliases), you are ready to test your applications.

Try to start a secure Web application in WebSphere Application Server for z/OS Version 5. WebSphere should prompt you for a user ID and password for authentication and authorization. When you deployed the application, provided you allowed “any authenticated” to be authorized to any roles the application uses, the requested page should be returned.

13.6 Sample scenario: Authentication with Tivoli Access Manager for e-business WebSEAL

How to integrate WebSphere Application Server for z/OS with Tivoli Access Manager for authentication and authorization is extensively described in Chapter 14, “IBM Tivoli Access Manager and WebSphere Application Server integration” on page 417. In this section, Tivoli Access Manager is installed on Linux on zSeries and LDAP on z/OS.

Tivoli Access Manager for e-business is the base software that is required to run applications in the IBM Tivoli Access Manager product suite. It enables the integration of IBM Tivoli Access Manager applications that provide a wide range of authorization and management solutions. Delivered as an integrated solution, these products provide an access control management solution that centralizes network and application security policy for e-business applications.

For more information about Tivoli Access Manager for e-business, refer to:

http://www.tivoli.com/products/index/access-mgr-e-bus/

Note: Tivoli Access Manager for e-business is the new name of the previously released software entitled Tivoli SecureWay® Policy Director. Also, for users familiar with the Tivoli SecureWay Policy Director software and documentation, the *management server* is now referred to as the *policy server*.

For this book, we will install Tivoli Access Manager on our Linux for zSeries environment, and use the functionality of Tivoli Access Manager to secure

access to resources on our WebSphere Application Server for z/OS application server. Due to the fact that all of our servers are located in an internal network, we have chosen not to use SSL communication between the Linux environment and IBM WebSphere Application Server for z/OS. If you are planning on setting up this scenario in your own environment, you might still want to consider using SSL.

We chose to use the SecureWay Directory Server for z/OS as our LDAP directory because we wanted to consolidate as much security information as possible in a single directory. The Tivoli Access Manager instance on Linux for zSeries is therefore configured to use LDAP on z/OS as its directory.

Why should you enable this scenario?

Implementing Tivoli Access Manager and WebSEAL or the Access Manager plug-in for WebSphere Edge Server in conjunction with WebSphere on z/OS enables you to:

- Define the users in the z/OS security product, such as RACF or ACF2.
- Have end users enter their MVS user ID and password when they access a URL that requires authentication.
- Validate this user ID and password by WebSEAL or Access Manager plug-in for Edge Server on a distributed platform, but validate against the password stored in the z/OS security product.
- Have the authenticated servlet run in WebSphere on the z/OS platform run with the principal derived from the user's authentication.

In our scenario we use a single Tivoli Access Manager instance. If you want to create an environment that is scalable and highly available, you will need to create several Tivoli Access Manager Linux instances and use a sprayer mechanism to distribute the workload across the various instances. Such a mechanism could be a round-robin DNS or it could be a Linux cluster using Linux Virtual Server (LVS).

If your back-end application servers are configured for scalability and high-availability, that is, you have several application server instances, your WebSEAL junctions can be configured to point to several hosts. To assure unified service, these hosts should be configured with exactly the same services. In such a scenario, WebSEAL will act as sprayer to send requests to your various application server instances in an intelligent manner. Additionally, if all of the HTTP listeners that WebSEAL points to are located on a z/OS-based server, high availability can be achieved by utilizing WLM monitoring functions through the use of Sysplex Distributor. More information about this product and its interaction with WebSphere for z/OS is available at:

`http://www.ibm.com/support/docview.wss?uid=tss1wp100312`

If you need to add session awareness into your workload distribution process, you might configure a WebSphere HTTP plug-in between your WebSEAL instance and the WebSphere Application server instance.

13.6.1 Enablement

In this section, we discuss enablement.

Installing and configuring LDAP on z/OS

LDAP on z/OS is configured as a TDBM directory server, with native authentication enabled. Native authentication enables you to map LDAP users to RACF IDs without storing a RACF password in the database. Packaged with z/OS V1R4, LDAP includes the ability to enable password changes to RACF IDs to be made through this native authentication configuration. Now, as user passwords expire, LDAP will automatically prompt the user for a new password and the change will be reflected in the RACF database. Refer to “Installing and configuring LDAP on z/OS” on page 400 for in-depth z/OS LDAP installation instructions.

After the LDAP server is configured, Tivoli Access Manager can be installed and connected directly to the LDAP server on z/OS. Any existing LDAP users can then be imported to Tivoli Access Manager through the pdadmin utility.

Additional users are defined to Tivoli Access Manager by using the standard Tivoli Access Manager commands, creating a user in the LDAP server. For example, the user defined to Tivoli Access Manager will have a “dn” value such as cn=P12345,ou=pok-racf,o=ibmpok, but be defined to Tivoli Access Manager as the user P12345. The user is also defined to the z/OS Security Server with an ibm-nativeId of P12345.

Installing Tivoli Access Manager for e-business

Tivoli Access Manager requires the installation of both a Global Security Toolkit (GSKIT) and an LDAP client. The Global Security Toolkit includes Secure Sockets Layer (SSL) with encryption strengths up to Triple DES.

We will install these two packages, gsk5bas-5.0-5.46.s390.rpm and ldap-clientd-4.1-1.s390.rpm.

The basic Tivoli Access Manager package for Linux on zSeries contains a number of RPM packages. For our particular use in this redbook, only the following two packages were necessary for installation, PDMgr-PD-4.1.0-2.s390.rpm, which is the Policy Director, and PDRTE-PD-4.1.0-2.s390.rpm, which is the Policy Director runtime.

We performed the installation using standard rpm, bottom-line commands:

```
rpm -ivh gsk5bas-5.0-5.46.s390.rpm
rpm -ivh ldap-clientd-4.1-1.s390.rpm
rpm -ivh PDMgr-4.1.0-2.s390.rpm
rpm -ivh PDRTE-4.1.0-2.s390.rpm
```

After a successful packet installation, we used the **pdconfig** command to configure the runtime and the Policy Server to point to the LDAP server running on z/OS.

Configuring the Tivoli Access Manager for e-business runtime

During the configuration of the Tivoli Access Manager runtime, you are prompted for the following information:

- Registry selection: Click to select the type of registry you configured for Tivoli Access Manager. Note that LDAP registry is the only supported choice.
- LDAP server hostname: Specifies the fully qualified host name or TCP/IP address of the LDAP server. We used 9.12.6.16, the TCP/IP address of our z/OS image.
- LDAP server port number: Specifies the port number on which the LDAP server listens. We used 3389, the port number where the z/OS LDAP server is listening.

Configuring the Tivoli Access Manager Policy Server

During the configuration of the policy server you are prompted for the following information:

- LDAP administrative user DN: Specifies the distinguished name of the LDAP administrator. We used `LDAP Administrator`.
- LDAP administrative user password: Specifies the password associated with the LDAP administrator ID. We used `secret`.
- Enable SSL communication between the Access Manager Policy Server and the LDAP server: Specifies whether SSL should be enabled, `yes` or `no`. We specified `no`, because we were running inside one single machine using HiperSockets™ for inter-image communication. If you want to avoid sending user ID/password combinations, you can specify `yes`, and configure for SSL using certificates. Refer to *Securing Linux for zSeries with a Central z/OS (RACF) LDAP Server*, REDP0221 for guidelines to encrypt the LDAP communication between your Linux virtual machines and the z/OS LDAP server.

The results of the above configuration activities are stored in the /opt/PolicyDirector/etc directory in three different files: pd.conf, ivmgrd.conf and ldap.conf.

Installing WebSEAL for Tivoli Access Manager

The basic Tivoli Access Manager WebSEAL package contains a number of rpm packages. For our particular use we only installed the PDWeb-PD-4.1.0-2.s390.rpm, using the standard rpm bottom line command:

```
rpm -ivh PDWeb-PD-4.1.0-2.s390.rpm
```

After successful installation of the package we used the **pdconfig** command to configure the Tivoli Access Manager WebSEAL instance, pointing to the z/OS LDAP instance on the same machine; see Example 13-4.

Example 13-4 The Access Manager for e-business Setup Menu

```
Access Manager for e-business Setup Menu
1. Configure Package
2. Unconfigure Package
3. Display Configuration Status
x. Exit
Please select the menu item [x]:
```

1. Type `1` for Configure Package.
2. On the next screen, type `1` for Access Manager WebSEAL Configuration.
3. On the following screen, type `1` for Configure and press Enter.

A prompt opens requesting you to enter the password of the Tivoli Access Manager administrator. Enter the password for sec_master. In this project we used `secret`.

You are now prompted to enable SSL communication between the WebSEAL server and the LDAP server. Type `y` (yes) or `n` (no) to enable SSL communication between the Access Manager server and the LDAP server.

The result of the configuration action is stored in /opt/pdweb/etc/webseald.conf.

Tip: If you repeatedly enter an incorrect password, you might see the error message:
Error: `This account has been temporarily locked out due to too many failed login attempts.`
If this occurs, obtain the correct password, wait five minutes for the lock to clear, and then restart `pdconfig`.

Note: In this book we do not use SSL because we are connecting through an internal network. If you decide not to allow passing of user ID/password, you can use SSL, even on an internal network. Refer to *Securing Linux for zSeries with a Central z/OS (RACF) LDAP Server*, REDP0221 for a full description and guidelines on how to encrypt the LDAP communication between your Linux virtual machines and the z/OS LDAP server.

Configuring the WebSEAL junction

To prepare for the test of this function, we define two WebSEAL junctions, one to connect to the HTTP server on our z/OS image and one to connect to WebSphere Application Server on z/OS.

These definitions are done on the Linux for zSeries image where Tivoli Access Manager is installed, using the **pdadmin** bottom-line command:

```
# pdadmin
pdadmin> login
Enter User ID: sec_master
Enter Password:
pdadmin> server task webseald-lnxsu4.itso.ibm.com create -t tcp
-h 9.12.6.16 -p 80 /test390
pdadmin> server task webseald-lnxsu4.itso.ibm.com create -t tcp
-h 9.12.6.16 -p 9087 /was390
```

- WebSEAL server identity: `webseald-lnxsu4.itso.ibm.com`
- WebSEAL junction name to IBM HTTP Server: test390
- WebSEAL junction name to transport handler: `was390`
- TCP/IP address of the z/OS image: `9.12.6.16`
- IBM HTTP Server on z/OS port: 80
- WebSphere Transport Handler port: 9087

The Linux for zSeries image is on TCP/IP address 9.12.9.38.

Authentication flow for this solution

Figure 13-12 on page 404 illustrates the authentication flow for this implementation.

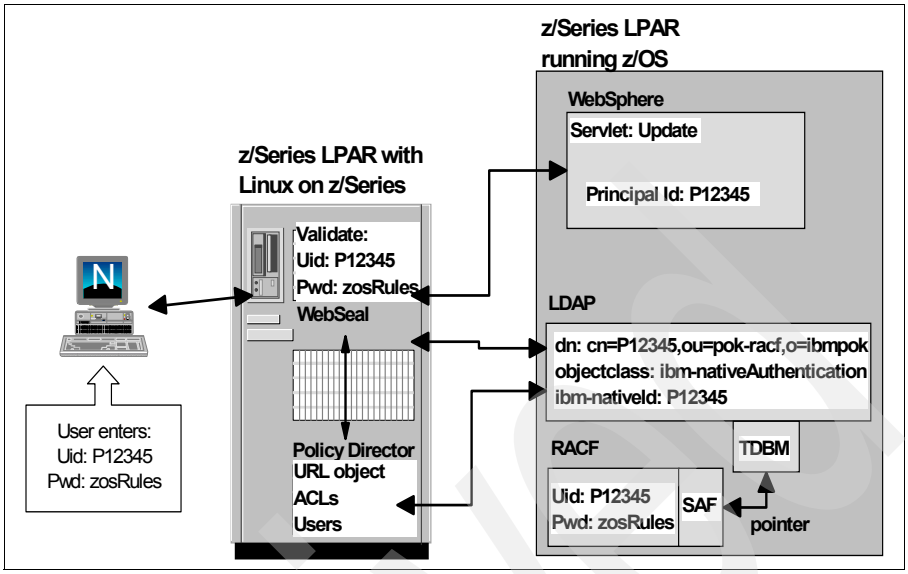


Figure 13-12 Authentication and URL authorization with WebSEAL

The following steps describe the authentication flow for this solution:

1. The user enters a URL that is directed to WebSEAL.
2. WebSEAL checks with Tivoli Access Manager to see if the URL requires authentication.
3. Because, for this URL, Tivoli Access Manager had been configured to specify that authentication is required, WebSEAL sends a request back to the end user requesting their user ID and password.
4. The end user enters the user ID of P12345 and a password.
5. WebSEAL receives the authentication information and then requests the LDAP server on the z/OS image to validate the user and password.
6. Because the user definition in the LDAP server has been configured to use native authentication, the LDAP extracts the value specified in the ibm-nativeId attribute of the user, which specifies the z/OS Security Server user ID.
7. LDAP can use the SDBM interface to the z/OS Security Server to have the Security Server validate the user ID and password.
8. The result of the validation is returned to WebSEAL.
9. If the validation fails, the end user has to retry.

10. If the validation succeeds, the WebSEAL checks with Tivoli Access Manager to see if the user that has been authenticated is authorized to access the URL.
11. Assuming the user is authorized, then the request flows to WebSphere on the z/OS LPAR.
12. WebSphere would receive the request. If the deployment descriptor specifies that the servlet requires authentication, WebSphere endeavors to obtain authentication details so that it can set the principal ID for the servlet to run under.
13. Regardless of whether the Access Manager plug-in for Edge Server or WebSEAL were used, the servlet runs in WebSphere with a principal ID that is a user ID defined to the z/OS Security Server. The end result is that this configuration provides very tight integration of processes that occur across multiple platforms.

13.6.2 Verification

To call the WebSphere Application Server on z/OS through WebSEAL, and execute application IBMEBizWeb/EJBCaller, the following URL is used:

```
http://9.12.9.38/was390/IBMEBizWeb/EJBCaller
```

When trying this URL, WebSEAL prompts for a user ID and password, which will be forwarded to LDAP on z/OS and verified, either directly in LDAP, or, depending on user ID LDAP settings, against the RACF database on z/OS.

Tip: The latest versions of Tivoli Access Manager and LDAP, used in this setup, will also handle the expired password situation.

13.7 Sample scenario: Implementing file-based custom user registry

For this redbook, we implemented a file-based custom user registry as a means to secure IBM WebSphere Application Server for z/OS. The file-based implementation is very simplistic and easy to use as a basic proof of concept. For more information about developing a database-oriented custom user registry, refer to the *WebSphere Application Server Information Center* at:

```
http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp
```

13.7.1 Implementation

Enabling CUR as the authentication/authorization mechanism for WebSphere on z/OS is done through the WebSphere administrative console. Complete the following steps:

1. From the navigation bar, click **Security** → **User Registries** → **Custom**.
2. Fill in the required information. Following is an example of what values we used:
   - Server User Id: `wsadmin`
   - Server User Password: `secret`
   - Custom Registry Classname: `com.ibm.websphere.security.FileRegistrySample`
   - IgnoreCase: `enabled`

   Click **Apply**.

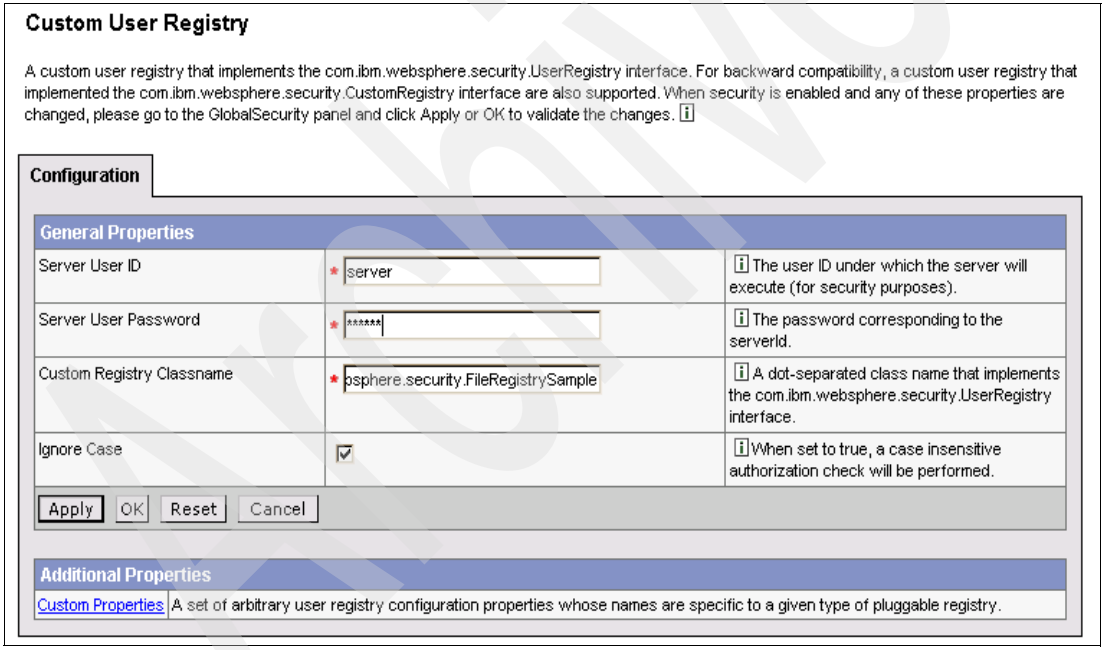


Figure 13-13 Custom User Registry properties

Put the custom registry classname, typically com.ibm.websphere.security.FileRegistrySample, in the class path of the server you are working on. This can be done through the UNIX Systems Services shell on your z/OS machine.

3. Add a server identity and a user ID, this ID should correspond to a user ID defined in the users.prop file.
4. Define environment variables for the user and group files:
   - userFile: /u/wunderl/users.prop
   - groupFile: /u/wunderl/groups.prop

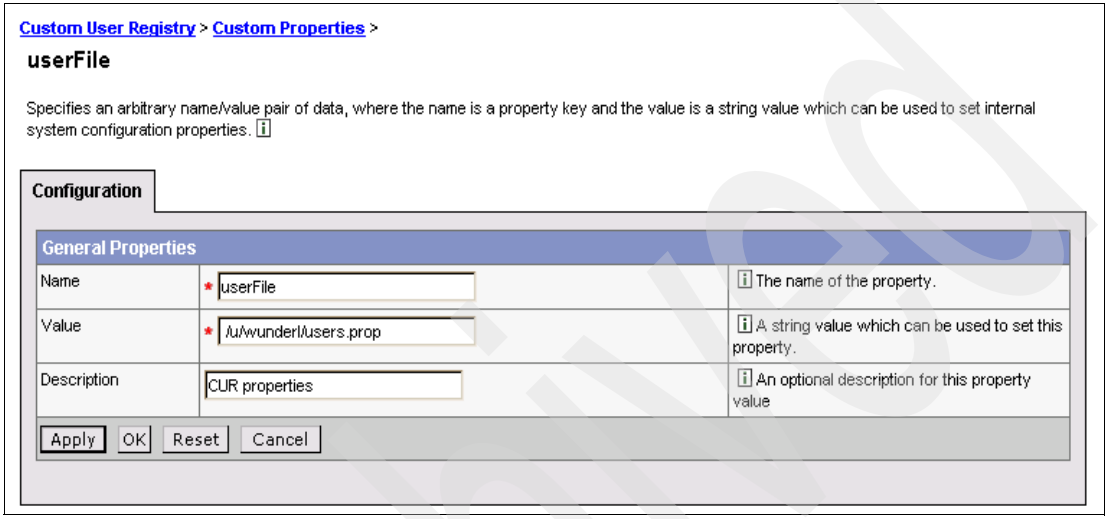


Figure 13-14 Defining Custom User Registry variables

5. Create user and group files and copy them into the specified directory.

Example 13-5 A sample user file

```
# Format:
# name:passwd:uid:gids:display name
# where name = userId/userName of the user
# passwd = password of the user
# uid = uniqueId of the user
# gid = groupIds of the groups that the user belongs to
# display name = a (optional) display name for the user.
WSADMIN:secret:112:666:WSADMIN
Holger:secret:123:666:
Eva:s1nner:124:777
Wunderl:strange:125:125,789:wunderlicht
```

Example 13-6 A sample group file

```
# Format:
# name:gid:users:display name
# where name   = groupId of the group
#       gid    = uniqueId of the group
#       users = list of all the userIds that the group contains
#       display name = a (optional) display name for the group.
admins:666:WSADMIN,wunderl,holger:Administrative group
users:777:eva:
```

6. Define an administrator who is also in your user file.

> **Tip:** The role-based authorizer for administration will use RACF EJBROLE profiles. So if you run with CUR enabled, you just need to add your RACF admin users to the file-based registry, and everything will work fine.

7. From the navigation bar, select **Security** → **Global Security**.
8. Under the Configuration tab, set the value for Active User Registry to CUSTOM, and click **Apply**.
9. Save your configuration and restart the server.

13.7.2 Verification

There are two ways to verify that WebSphere is successfully configured to use a custom user registry for securing a Web application on z/OS.

First, go to the WebSphere administrative console. If your server was running without security, a logon should now be required. Following the above installation instructions, WSADMIN should be used as the logon ID and “secret” as the password. If the administrative console comes up, your IBM WebSphere Application Server for z/OS is now being protected through a custom user registry.

Second, go to a secure Web application in WebSphere on z/OS. WebSphere should prompt you for a user ID/password for authentication and authorization. Again, if the above instructions were followed, bob should be used as the logon ID and password. As long as the resource is protected by “any authenticated,” the requested page should be returned.

13.8 Sample scenario: Off-platform authentication with WebSEAL on Linux for zSeries using TAI

For this book we chose to implement a TAI solution using WebSEAL on Linux for zSeries and WebSphere Application Server for z/OS Version 5. WebSEAL is shipped with a portion of TAI already present, simplifying the implementation process. WebSEAL is configured to use LDAP on z/OS as its user registry. Installation and configuration instructions for both products are found earlier in this chapter.

13.8.1 Implementation

In this section, we discuss the implementation.

Installing and configuring WebSEAL

WebSEAL should be configured on Linux on zSeries and a junction should already be created that will connect WebSEAL to the WebSphere Application Server Protocol Handler or the IBM HTTP Server with the WebSphere HTTP plug-in. For additional instructions on creating WebSEAL junctions, see “Configuring the WebSEAL junction” on page 403.

Enabling TAI for WebSphere

Enabling and Configuring TAI for WebSphere can be done through the WebSphere administrative console.

1. Click **Security** → **Authentication Mechanisms** → **LTPA** on the navigation panel.
2. Under Additional Properties, click **Trust Association** and select the **Trust Association Enabled** option.
3. Again, under Additional Properties, click **Interceptors**.
4. The WebSEAL interceptor should be provided in the drop-down box: com.ibm.ws.security.web.WebSealTrustAssociationInterceptor.
5. Under Additional Properties, click **Custom Properties.**
6. Add the following name value pairs:
   - com.ibm.websphere.security.trustassociation.types = WebSEAL
   - com.ibm.websphere.security.webseal.loginId = <WebSEAL server ID>
   - com.ibm.websphere.security.webseal.id = iv-user
   - com.ibm.websphere.security.webseal.hostnames = <host name or IP >
   - com.ibm.websphere.security.webseal.ports = <port number>

– com.ibm.websphere.security.webseal.ignoreProxy = true

7. Click **OK** and save the master configuration.

13.8.2 Verification

Enable global security. Then save, stop, and restart all cells, nodes, and application servers for your changes to take effect.

Try to access a secure Web resource through WebSEAL using the following format for the Web address:

```
http://<webseal-ipaddress>/<junction-name>/<securewebapp>
```

An authentication challenge should be initiated by WebSEAL and the secure resource should be returned upon receipt of a valid user ID/password.

13.9 Sample scenario: Implementing a custom TAI

We now describe how to implement a custom trust association interceptor (TAI).

For this purpose, we use a sample TAI developed by ITSO available with the Web Material. The ITSO sample TAI in this archive file is in the AdditionalMaterial\Swipe\ directory and it is called pokItsoTai1.jar.

The cProxy TAI is packaged in a Java archive file that contains several TAIs. The ITSO sample TAI Java archive is named pokItsoTai1.jar. The compiled TAI class that we use here is named cProxy.class. The TAI source code is named cProxy.java. This cProxy TAI uses a property file called DistribUserid.properties. Here are the choices that have been made for this sample cProxy TAI:

- isTargetInterceptor reads through the HTTP headers looking for the “Host” header. When found, it checks if the values extracted matches the values coded for WS390Host in the DistribUserid.properties file. If it is a match, processing continues to the validateEstablishedTrust method. If it is not a match, a value of false is returned, which tells WebSphere that this TAI will not process this request.
- validateEstablishedTrust reads through the HTTP headers looking for the “Via” header. When found, it extracts the value and checks that the string specified in the WS390Via field from DistribUserid.properties file. The purpose of this check is to verify that the request came from a known system, in this case, a proxy server. If this is successful, processing continues in the getAuthenticatedUsername method. If it is not, an exception is thrown and the request is rejected.

- getAuthenticatedUsername reads through the HTTP headers looking for the $wsru header, and then extracts the value associated with this header, which is the user ID that was validated by the Remote Proxy Security Server (RPSS). Having obtained the user ID, which should correspond to a valid user registry ID, we just pass this value back to WebSphere as the user ID of the servlet Java Principal. WebSphere then runs the servlet as that user ID. If no match is found, an exception is thrown and the request is rejected.

In an HTTP request, the Host header field specifies the Internet host of the resource being requested. This is the host name of our z/OS LPAR in our example. The Via header field is used by gateways and proxies to indicate on the request the intermediate protocols and recipients between the user and the server. The $wsru header field is added by the RPSS and contains the a valid user registry ID.

Here again, the WebSphere TAI on z/OS does not implement the trust relationship between the authenticating server and the WebSphere Application Server itself. We recommend that you establish a level of trust by enforcing a trusted connection between these two parties.

To set up WebSphere Application Server for z/OS to enable your custom trust association interceptor (TAI):

1. If your TAI uses the “init (String propsfile)” method, it means that your TAI needs a properties file. In this case, you first need to modify your .jar file to include the properties file. You can use software such as WinZip to modify the properties file from the .jar file. In our example, we customize the DistribUserid.properties file from pokItsoTai1.jar.

   If your TAI uses the “init (Properties props)” method, your TAI will read properties from the WebSphere Application Server configuration, and you do not need to set any property at this point.
2. Transfer your TAI in binary to UNIX System Services. Put it in a directory that is in the WebSphere Application Server class path. You can put it in <was_install_root>/appserver/lib if your TAI is packaged in a .jar file or in <was_install_root>/appserver/classes if your TAI is simply a compiled class file.
3. Give the file the right UNIX permissions so that WebSphere Application Server can read it.
4. Log on to the WebSphere Application Server administrative console.
5. Navigate to **Security** → **Authentication mechanisms** → **LTPA** → **Trust Association**. Select the **Trust Association Enabled** option and click **Apply**.
6. Navigate to **Additional Properties** → **Interceptors** and click **New**.

7. Enter your TAI class name and press **OK**. Ask your developer for the class name. In our example, it is `itso.edge.cProxy`.
8. Click your TAI class name. Then, select **Custom Properties**.
9. If your TAI uses the “init (String propsfile)” method, you now need to specify the name of your property file. For this purpose, you need to set up the following variable:

   ```
   com.ibm.websphere.security.trustassociation.initPropsFile <file_name>
   ```

   Notice that you have to remove the “.properties” extension. In our example, the property file is DistribUserid.properties. Therefore, we set up the following variable:

   ```
   com.ibm.websphere.security.trustassociation.initPropsFile DistribUserid
   ```

   Figure 13-15 shows the administrative console display at this point.

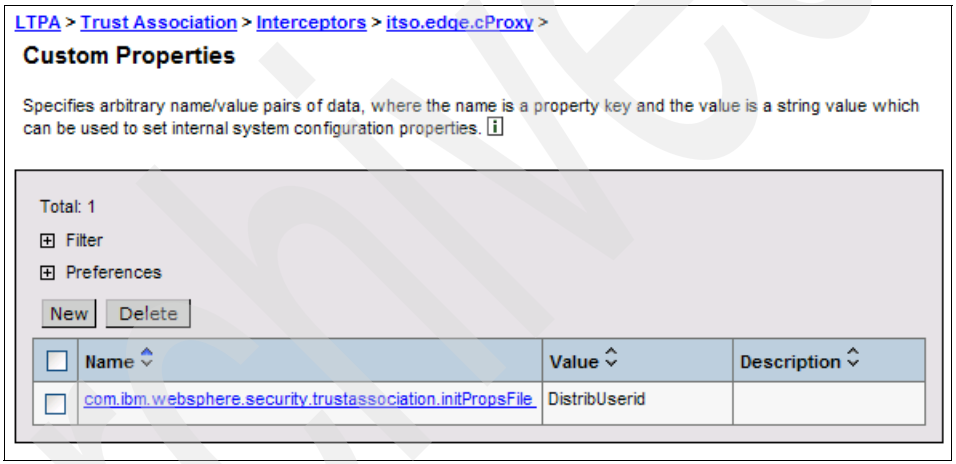


Figure 13-15 Trust Association Interceptor Custom Properties

10. If your TAI uses the “init (Properties props)” method, you now need to add any property that is necessary for your custom trust association interceptor to initialize. You can find properties your TAI is looking for by reading its “init” method. For example, if your “init” method contains the following items, you need to create a com.ibm.websphere.security.myTai.id custom property:

    ```
    String s = (String)properties.get("com.ibm.websphere.security.myTai.id");
    ```

11. Click **Save** and then **Save** again to the master configuration.

12. Stop and restart your server. You should see in the servant region output a message similar to the following:

```
BBOO0222I SECJ0121I: Trust Association Init class itso.edge.cProxy
loaded successfully
BBOO0222I SECJ0122I: Trust Association Init Interceptor signature:
WS390Interceptor 1.1  IBM Corporation (C)opyright  1999, 2000, 2001
```

Our TAI is designed to write a message to stdout when the initialization completes successfully. Therefore, we see the following message:

```
cpTai --> Exiting initialization: SUCCESS
```

You can now test your TAI.

Using our TAI, we see the messages shown in Example 13-7.

Example 13-7 Our TAI messages

```
cpTai--> isTargetInterceptor: HttpHdr: authorization Basic dXNy
cpTai--> isTargetInterceptor: HttpHdr: via HTTP/1.1 tot188:443
cpTai--> isTargetInterceptor: HttpHdr: user-agent Mozilla/4.0 (
cpTai--> isTargetInterceptor: HttpHdr: iv_server_name default-w
cpTai--> isTargetInterceptor: HttpHdr: host wtsc59.itso.ibm.com
cpTai --> isTargetInterceptor: Check for Host=wtsc59.itso.ibm.c
cpTai --> matched host, will action
cpTai: in validateEstablishedTrust
cpTai: --> validateEstablishedTrust: HttpHdr: authorization Basic dXNyd29ya
cpTai: --> validateEstablishedTrust: HttpHdr: via HTTP/1.1 tot188:443
cpTai --> validateEstablishedTrust: will acceptvia
cpTai: --> validateEstablishedTrust: HttpHdr: user-agent Mozilla/4.0 (compa
cpTai: --> validateEstablishedTrust: HttpHdr: host wtsc59.itso.ibm.com:9085
cpTai: --> validateEstablishedTrust: HttpHdr: accept image/gif, image/x-xbi
application/vnd.ms-powerpoint, application/msword, application/x-shockwave-
cpTai: --> validateEstablishedTrust: HttpHdr: connection close
cpTai: --> validateEstablishedTrust: HttpHdr: accept-language en,en-us;q=0.
cpTai: --> validateEstablishedTrust: HttpHdr: $wsru usrwork
ITSO cpTai --> mapping: usrwork to: usrwork
```

13.10 Registry choices summary table

Table 13-1 on page 414 summarizes most of the registry options you have today with WebSphere Application Server for z/OS Version 5.

Table 13-1 Registry options for authentication and authorization

	Authentication		**Authorization**	
Registry configured	**Web container**	**EJB container**	**Web container**	**EJB container**
Local OS, container	**Transport handler:** ► SAF **with TAI:** ► anywhere ► Reverse Proxy ► WebSEAL ► Tivoli Access Manager CP plug-in -> map to SAF **Mutual SSL:** ► SAF ► IBM HTTP Server HTTP PI (RACF plex) =>map to SAF USERID (1) ► IBM HTTP Server HTTP PI (in a nonshr registry) =>map to SAF USERID (2)	► z/SAS ► CSIv2	► EJBROLES, GEJBROLES ► or bindings	► EJBROLES, GEJBROLES ► or bindings
Programmatic options with Local OS	► JAAS against SAF	► JAAS against SAF	► getUserPrincipal sUserInRole ► Tivoli Access Manager Java APIs against remote registry	► getCallerPrincipal sCallerInRole ► Tivoli Access Manager Java APIs against remote registry

	Authentication		Authorization	
Custom user registry & LDAP, container	**Transport handler** ► Gets resolved in supplied registry (TCP/IP or local file) **with TAI:** ► Anywhere ► Reverse Proxy ► WebSEAL ► Tivoli Access Manager CP plug-in -> map to CUR **mutual SSL:** ► CUR ► IBM HTTP Server HTTP PI (same registry than CUR) => map to CUR USER (3) ► IBM HTTP Server HTTP PI (in a nonshr registry, like the RACFplex) =>map to CUR USERID (4)	► Not available today Note: container needs to be protected against connecting IIOP clients when running in CUR mode ► With WebSphere Application Server V5.02 against supplied registry ► CSIv2 ► zSAS not supported	► Bindings, checking against CUR/LDAP authenticated user	► Bindings, checking against SAF server ID ► With WebSphere Application Server V5.02 checking bindings against CUR/LDAP authenticated user
Programmatic options with custom user registry & LDAP	► JAAS against CUR interface (TCP/IP or local file)	► JAAS against CUR interface (TCP/IP or local file)	► getUserrPrincipal isUserInRole ► Tivoli Access Manager Java APIs against remote registry	► getCallerPrincipal sCallerInRole ► Tivoli Access Manager Java APIs against remote registry
AMWAS container (WebSphere Application Server V5.02 and later)	► Tivoli Access Manager	► Tivoli Access Manager	► Tivoli Access Manager ACL	► Tivoli Access Manager ACL

	Authentication		Authorization	
programmatic options with AMWAS (APAR OA02022	► JAAS	► JAAS	► getUserrPrincipal isUserInRole ► Tivoli Access Manager Java APIs against remote registry	► getUserrPrincipal isUserInRole ► Tivoli Access Manager Java APIs against remote registry

The following steps explain the corresponding numbers in Table 13-1 on page 414:

1. After the handshake was successful, you can be sure that the certificate is already in SAF and can be mapped to a valid identity.
2. If IBM HTTP Server WebSphere HTTP plug-in uses a different registry, the certificate that is forwarded to WebSphere Application Server might not be available in the SAF registry. The certificate needs to be mapped to an SAF user ID.
3. After the handshake was successful, you can be sure that the certificate is already in CUR and can be mapped to a valid identity.
4. The SSL handshake, based on a certificate that is in RACF, needs to be mapped to a non-RACF user available in CUR.

Note: The ability to exploit thread identity support in WebSphere Application Server for z/OS and OS/390 is currently available only when using a local OS registry (SAF). That is, a J2CA connector can pass the user ID associated with the current Java thread on a container-managed connection only when the user registry is LocalOS. With other registries, the user ID of the servant region is propagated to the EIS on a container-managed J2CA connection unless a container-managed authentication alias or mapping-configuration alias is specified.

14

IBM Tivoli Access Manager and WebSphere Application Server integration

This chapter discusses IBM Tivoli Access Manager as a solution to secure access to WebSphere Application Server for z/OS. We focus on using the highest level of integration between IBM Tivoli Access Manager and WebSphere Application Server for z/OS. It enables cross-platform centralized authentication and J2EE roles authorization management.

In this chapter, we discuss the following topics:

- Introducing IBM Tivoli Access Manager for WebSphere
- Configuring Tivoli Access Manager and WebSphere Application Server integration
- Using Tivoli Access Manager for WebSphere

14.1 Introducing IBM Tivoli Access Manager for WebSphere

In this section, we introduce the Tivoli Access Manager features and components, and we describe scenarios involving Tivoli Access Manager and WebSphere Application Server for z/OS.

14.1.1 Tivoli Access Manager features and components

Tivoli Access Manager is an authentication and authorization solution for enterprise Web, client/server, and existing applications. Tivoli Access Manager enables you to control user access to protected information and resources. By providing a centralized, flexible, and scalable access control solution, Tivoli Access Manager enables you to build secure and easy-to-manage network-based applications and e-business infrastructure.

Tivoli Access Manager features

Tivoli Access Manager supports authentication, authorization, data security, and resource management capabilities.

The Tivoli Access Manager network security management solution provides and supports the following core technologies:

- Authentication: The first step a user must take when making a request for a resource that is protected by Tivoli Access Manager. During authentication, a user's identity is validated. Tivoli Access Manager enables a highly flexible approach to authentication through the use of the authorization API.
- Authorization: Enforces the security policy by determining what objects a user can access and what actions a user can take on those objects and then granting appropriate access to the user. Tivoli Access Manager handles authorization through the use of the following features:
  - Tivoli Access Manager authorization service
  - Access control lists (ACLs), protected object policies (POPs), and authorization rules for fine-grained access control
  - Standards-based authorization API, using the aznAPI for C language applications, and the Java Authentication and Authorization Service (JAAS) for Java language applications
  - External authorization service capability

- Quality of protection: The degree to which Tivoli Access Manager protects any information transmitted between client and server. The quality of data protection is determined by the combined effect of encryption standards and modification-detection algorithms. Quality of protection levels include standard Transmission Control Protocol (TCP) communication (no protection) and data integrity and data privacy provided by the Secure Sockets Layer (SSL) communication protocol.
- Scalability: The ability to respond to increasing numbers of users who access resources in the domain. Tivoli Access Manager uses the following techniques to provide scalability: replication of services, front-end replicated servers, back-end replicated servers, optimized performance by allowing the off-loading of authentication and authorization services to separate servers, and scaled deployment of services.
- Accountability: Tivoli Access Manager provides a number of logging and auditing capabilities. Log files capture any error and warning messages generated by Tivoli Access Manager servers. Audit trail files monitor Tivoli Access Manager server activity.
- Centralized management: Three methods are provided for managing security policy and the Tivoli Access Manager servers:
  - The **pdadmin** command line utility
  - Web Portal Manager graphical user interface (GUI)
  - Administration API

  You can accomplish most tasks using any of these methods. However, some tasks cannot be performed using the Web Portal Manager.

Tivoli Access Manager handles the programmatic security and the declarative security approaches.

As far as programmatic security is concerned, Tivoli Access Manager supports JAASLogin for authentication (PDLogin Java class) and PDPermission for authorization (PDPermission class). The PDLogin class knows how to authenticate to Tivoli Access Manager. The PDPermission class knows how to locate the current JAAS Subject, extract the authentication information, and contact Tivoli Access Manager to see if the current Subject is allowed the particular access to the particular resource. Non-JAAS applications are also supported.

Declarative security is based on the J2EE Web or EJB containers. It uses role-to-method mappings defined at the application deployment descriptor level. It also uses subject-to-role mappings defined at the Tivoli Access Manager level.

Tivoli Access Manager components

The computing environment on which Tivoli Access Manager enforces your security policies for authentication, authorization, and access control is called a secure domain. Figure 14-1 illustrates the secure domain components.

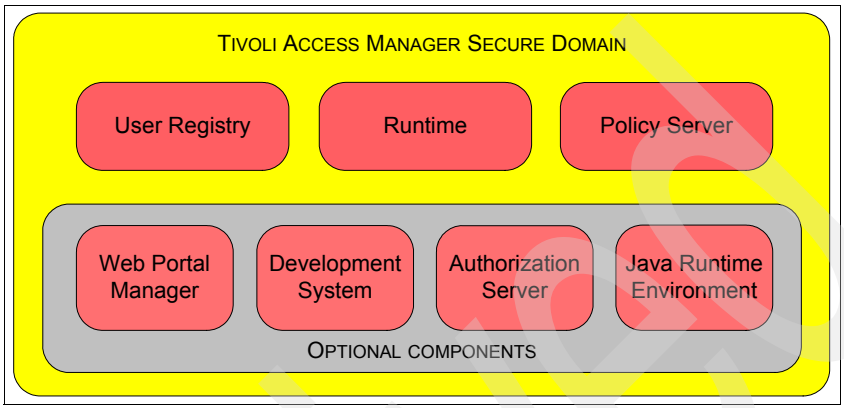


Figure 14-1 IBM Tivoli Access Manager secure domain components

Tivoli Access Manager requires a *user registry* to support the operation of its authorization functions. The registry provides a database of the user identities known to Tivoli Access Manager. It also provides a representation of groups in Tivoli Access Manager roles that might be associated with users. The Microsoft Windows version of Tivoli Access Manager comes with Tivoli Directory Server. Other LDAP servers, such as IBM z/OS Security Server LDAP server, are also supported. You can find the extensive lists of supported LDAP servers in *IBM Tivoli Access Manager for e-businessWeb Security Installation Guide Version 5.1*, SC32-1361-00.

The Tivoli Access Manager *policy server* maintains the master authorization database for the secure domain. This server is key to the processing of access control, authentication, and authorization requests. It also updates authorization database replicas and maintains location information about other Tivoli Access Manager servers in the secure domain. There can be only one instance of the policy server and its master authorization database in any secure domain at one time. For availability purposes, a standby server can be configured to take over policy server functions in the event of a system failure.

The *authorization server* offloads access control and authorization decisions from the policy server. It maintains a replica of the authorization policy database and functions as the authorization decision-making evaluator. A separate authorization server also provides access to the authorization service for third-party applications that use the Tivoli Access Manager authorization API in remote cache mode.

The Tivoli Access Manager *Java Runtime Environment* offers a reliable environment for developing and deploying Java applications in a Tivoli Access Manager secure domain. You can use it to add Tivoli Access Manager authorization and security services to new or existing Java applications. Note that if you plan to install the Web Portal Manager interface, this component is required.

The Tivoli Access Manager *runtime* contains runtime libraries and supporting files that applications can use to access Tivoli Access Manager servers. You must install the Tivoli Access Manager runtime or the Tivoli Access Manager Java Runtime Environment on every system in your secure domain.

The *Web Portal Manager* is a Web-based graphical user interface (GUI) used for Tivoli Access Manager administration. Similar to the **pdadmin** command line interface, this GUI provides management of users, groups, roles, permissions, policies, and other Tivoli Access Manager tasks. A key advantage is that you can perform these tasks remotely, without requiring any special network configuration.

14.1.2 Why use Tivoli Access Manager with WebSphere?

Tivoli Access Manager enables centralized security management for a wide variety of resources. For example, it can secure access to WebSphere application servers, non-WebSphere applications servers, WebSphere MQ queues, operating systems, and so on. Therefore, WebSphere Application Server can participate in a broad, large, and cohesive security architecture. The Tivoli Access Manager comprehensive, policy-based approach addresses z/OS and non-z/OS resources.

Externalized security also enables single sign-on solutions across applications or resources access.

More specifically, Tivoli Access Manager offers the following key features to WebSphere Application Server:

- Early user authentication with WebSEAL or WebSphere Edge Server Tivoli Access Manager plug-in
- A J2EE authorization module (AMWAS) that replaces built-in WebSphere Application Server J2EE security
- Lightweight Third Party Authentication (LTPA) support for single sign-on
- WebSphere trust association interceptor support for single sign-on
- Shared user registry capability
- Java security classes for programmatic use

- Single sign-on to forms-based login

Integrating Tivoli Access Manager with WebSphere Application Server has the following advantages:

- Container-based security: EJB/servlet/JSP developers focus on business.
- Central administration: Manage WebSphere and non-WebSphere environments.
- Flexibility: More flexible user-to-role policies are possible.
- Transparency: Added value, yet no changes to J2EE code.
- Standards-based: J2EE 1.3.
- Dynamic: Make user-to-role changes without restarting WebSphere Application Server.

14.1.3 Scenario 1: Tivoli Access Manager authentication and LocalOS authorization for WebSphere

Tivoli Access Manager has the ability to use a z/OS LDAP server to store its user registry. Additionally, z/OS LDAP can be configured for native authentication. Native authentication enables authentication to be done using RACF user IDs and passwords. To support full auditing capability, many businesses, such as banks or insurance companies, require that RACF is used to control access to secure data; native authentication provides this support.

Figure 14-2 on page 423 shows a scenario using WebSphere Application Server for z/OS, Tivoli Access Manager for authentication, and z/OS LDAP native authentication.

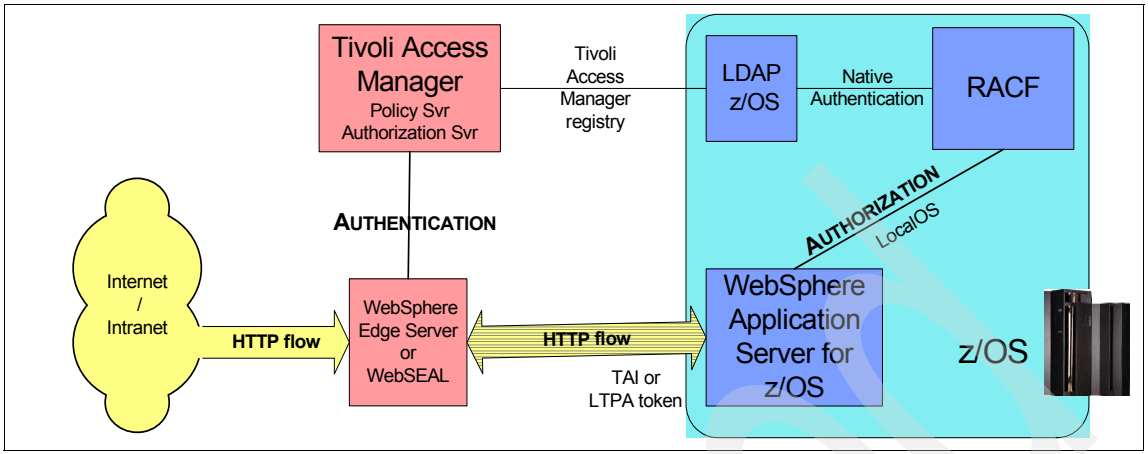


Figure 14-2 Scenario 1: Tivoli Access Manager authentication and LocalOS authorization

In this scenario, IBM Tivoli Access Manager is only used for authentication. This authentication is done against LDAP z/OS, which checks passwords against RACF. WebSphere Application Server for z/OS uses the LocalOS user registry (RACF) for authorization.

This is a single sign-on (SSO) scenario. Authentication occurs at the WebSphere Edge Server or WebSEAL level. The identity of the user is then retrieved by WebSphere Application Server using a trust association interceptor (TAI) or using LTPA tokens. TAI or LTPA tokens implementation is described in 14.2.6, “Configure single sign-on between WebSEAL and WebSphere” on page 452.

The WebSphere Edge Server has a Tivoli Access Manager plug-in to act as a Remote Proxy Security Server (RPSS). WebSEAL is an RPSS. WebSphere Edge Server is the preferred solution for high-availability configuration to balance workload to WebSphere Application Server clusters, with support for dynamic configuration.

Why should you set up scenario 1?

- WebSphere Application Server for z/OS uses LocalOS (RACF) for authorization.
- Tivoli Access Manager enables cross-platform, centralized authentication management.
- Native authentication enables user registry passwords to be stored inside RACF where they are not accessible from the outside.
- Native authentication enables end users to enter their MVS user ID and password when they access a URL that requires authentication.
- LDAP can be a central user registry for single sign-on solutions.
- The TAI or LTPA tokens enable single sign-on between WebSEAL and WebSphere Application Server.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL enables early authentication in the DMZ.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL protects URIs.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL hides WebSphere Application Server from the exposed network.
- Because a LocalOS user registry is used by WebSphere Application Server for z/OS and OS/390, even though authentication takes place outside WebSphere Application Server in Tivoli Access Manager, the user ID passed to the TAI by Tivoli Access Manager can then be propagated to an EIS on a container-managed J2CA connection.

This scenario setup is extensively described with WebSphere Application Server for z/OS V4.01 in the IBM Redbook *Tivoli and WebSphere Application Server for z/OS*, SG24-7062. LDAP native authentication setup is described in 13.5, “Sample scenario: Using LDAP native authentication” on page 374.

14.1.4 Scenario 2: Tivoli Access Manager authentication and authorization for WebSphere

WebSphere Application Server has the ability to interface with Tivoli Access Manager for authentication and J2EE role-based authorization purposes. The role-to-user mapping is stored in the Tivoli Access Manager ACL database. The access decision can be further constrained by the particular cell, host, or server on which the application instance is running.

Figure 14-3 shows a scenario using WebSphere Application Server for z/OS, Tivoli Access Manager for authentication, and J2EE role-based authorization.

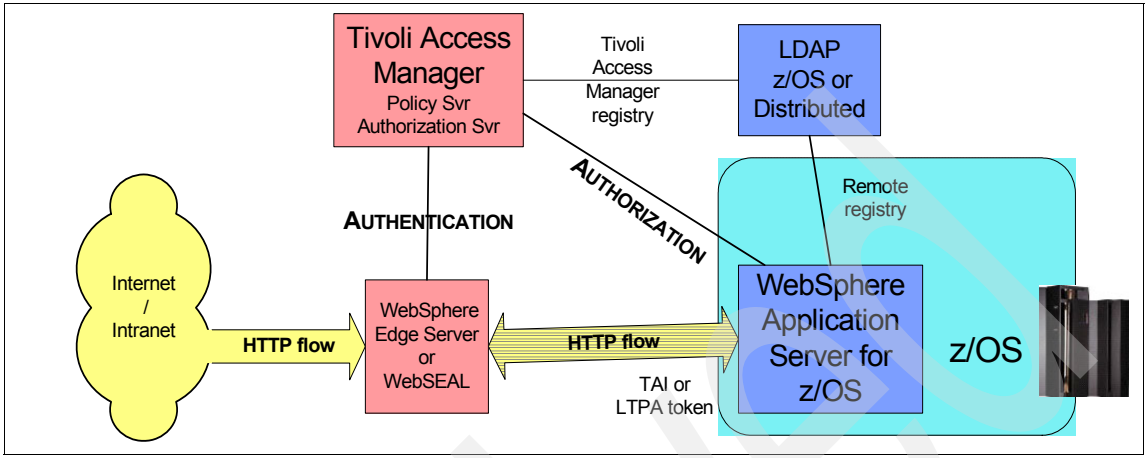


Figure 14-3 Scenario 2: Tivoli Access Manager authentication and authorization

In this scenario, IBM Tivoli Access Manager is used for authentication and authorization. WebSphere Application Server for z/OS uses the LDAP remote registry and Tivoli Access Manager for J2EE role-based authorization.

This is a single sign-on (SSO) scenario. Authentication occurs at the WebSphere Edge Server or WebSEAL level. Then, the identity of the user is retrieved by WebSphere Application Server using a trust association interceptor (TAI) or using LTPA tokens. TAI or LTPA tokens implementation is described in 14.2.6, “Configure single sign-on between WebSEAL and WebSphere” on page 452.

The WebSphere Edge Server has a Tivoli Access Manager plug-in to act as a Remote Proxy Security Server (RPSS). WebSEAL is a RPSS. WebSphere Edge Server is the preferred solution for high-availability configuration to balance workload to WebSphere Application Server clusters, with support for dynamic configuration.

Why should you set up scenario 2?

- Tivoli Access Manager enables cross-platform, centralized authentication and authorization management.
- Tivoli Access Manager enables cross-platform, centralized users access to J2EE roles management.
- LDAP can be a central user registry for single sign-on solutions.
- The TAI or LTPA tokens enable single sign-on between WebSEAL and WebSphere Application Server.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL enables early authentication in the DMZ.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL protects URIs.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL hides WebSphere Application Server from the exposed network.

We describe this scenario in this chapter.

14.1.5 Scenario 3: Tivoli Access Manager authentication, authorization, and native authentication for WebSphere

This scenario is scenario 2 with LDAP native authentication in addition.

Figure 14-4 on page 427 shows a scenario using WebSphere Application Server for z/OS, Tivoli Access Manager for authentication, J2EE role-based authorization, and LDAP native authentication.

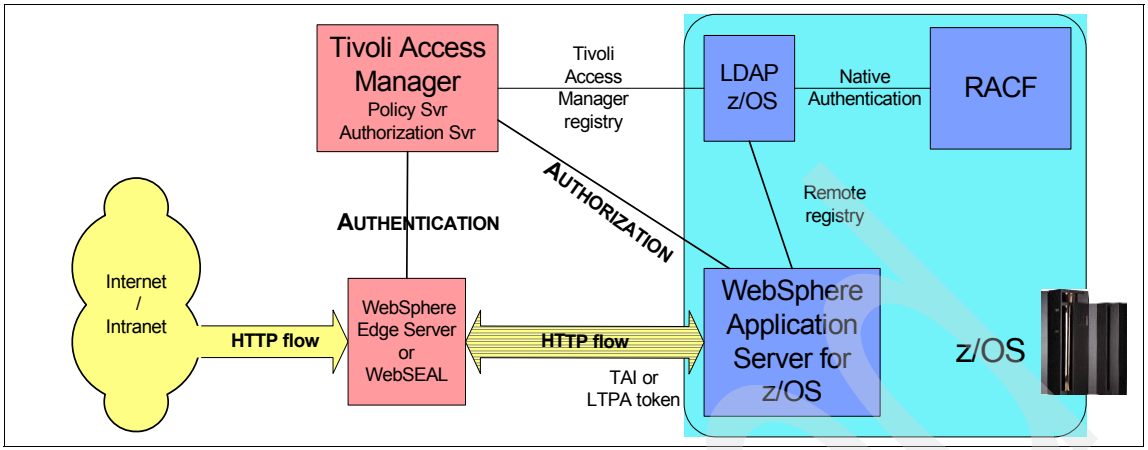


Figure 14-4 Scenario 3: Tivoli Access Manager authentication, authorization, and native authentication

In this scenario, Tivoli Access Manager is used for authentication and authorization. WebSphere Application Server for z/OS uses the LDAP remote registry and Tivoli Access Manager for J2EE role-based authorization. The authentication is done against LDAP z/OS, which checks passwords against RACF.

Why should you set up scenario 3?

- Tivoli Access Manager enables cross-platform, centralized authentication and authorization management.
- Tivoli Access Manager enables cross-platform, centralized users access to J2EE roles management.
- Native authentication enables user registry passwords to be stored inside RACF where they are not accessible from the outside.
- Native authentication enables end users to enter their user ID and MVS password when they access a URL that requires authentication.
- LDAP can be a central user registry for single sign-on solutions.
- The TAI or LTPA tokens enable single sign-on between WebSEAL and WebSphere Application Server.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL enables early authentication in the DMZ.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL protects URIs.
- WebSphere Edge Server with Tivoli Access Manager plug-in or WebSEAL hides WebSphere Application Server from the exposed network.

To set up this scenario, follow instructions in this chapter and set up native authentication as explained in 13.5, “Sample scenario: Using LDAP native authentication” on page 374.

14.2 Configuring Tivoli Access Manager and WebSphere Application Server integration

In this section, we describe how to set up scenario 2, explained in 14.1.4, “Scenario 2: Tivoli Access Manager authentication and authorization for WebSphere” on page 424. This scenario uses the AMWAS module that enables container-based security to be managed by Tivoli Access Manager. Both programmatic and declarative security can be used with this setup. This configuration process consists of the following steps:

1. Configuring the Tivoli Access Manager secure domain.
2. Configuring Tivoli Access Manager specifically for WebSphere Application Server for z/OS.
3. Configuring WebSphere Application Server for interacting with Tivoli Access Manager.

4. Enabling WebSphere Application Server global security.
5. Migrating WebSphere Application Server security to Tivoli Access Manager.
6. Configuring single sign-on between WebSEAL and WebSphere Application Server.

14.2.1 Tivoli Access Manager setup in our environment

In our environment, we installed IBM Tivoli Access Manager Version 5.1 and the IBM Directory Server (LDAP) on a Microsoft Windows Server 2003 machine.

If you install Tivoli Access Manager on Linux on zSeries, refer to 13.6, “Sample scenario: Authentication with Tivoli Access Manager for e-business WebSEAL” on page 398 where this installation is described.

This book does not describe how to install Tivoli Access Manager, but how to integrate Tivoli Access Manager with WebSphere Application Server for z/OS.

Nevertheless, here are the steps we went through and tips that we provide to install Tivoli Access Manager on a Microsoft Windows Server 2003:

1. Install the Java Runtime Environment (JRE). Launch the IBM JRE 1.3.1 install program from Tivoli Access Manager 5.1 Base CD. Be careful to choose the option to select this installation as the default system JVM.
2. Create a DB2 administrator user ID in Windows user accounts. We created the db2admin user in our environment. Add this user to the administrators group.
3. Install IBM Tivoli Directory Server (LDAP). Run **`install_ldap_server.exe`**. It is critical to use the default “key4ssl” SSL keyfile password if you use the default keyfile am_key.kdb. Otherwise, you will end up with an LDAP server that is functional in every respect except that LDAP will not listen on the SSL port.

   Issue a **`netstat -na`** command at a command prompt to check that both the non-SSL and the SSL ports are opened. Also, check the ibmslapd.log to make sure that LDAP started with no errors.

4. Add dependencies to the Windows registry to make services start in a specific order. The “ibmdiradm” service needs to start after the “ibmslapd” service which needs to start after the “DB2ADMIN” service. For this purpose, add the following values to keys in the Windows registry using **regedit** or **regedt32**:

```
Key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ibmdiradm
Value Name: dependOnService
Data Type: REG_MULTI_SZ
Value: ibmslapd

Key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ibmslapd
Value Name: dependOnService
Data Type: REG_MULTI_SZ
Value: DB2ADMIN
```

   If you do not add these registry keys prior to installing the Tivoli Access Manager policy server, LDAP will fail to autostart following the system restarts, which are required during Tivoli Access Manager installs, and because of this, you will get unexpected error messages during the Tivoli Access Manager installs that follow.

5. Install Tivoli Access Manager policy server. Run **install_ammgr_exe**.
6. Install Tivoli Access Manager authorization server. Run **install_ammgr.exe**.
7. Install Tivoli Access Manager Web Portal Manager. Be careful, the Web Portal Manager install executable for Windows Server 2003 is different from the executable for Windows 2000 Server. There was no difference in executables for the LDAP server, the policy server, or the authorization server.

   WebSphere Application Server should not be already installed. It appears that WebSphere Application Server should be installed and configured under the control of the Web Portal Manager wizard. Uninstall WebSphere if necessary.

   Then, run **install_amwpm.exe** for Windows Server 2003.

   If the configuration fails, run **pdconfig**, select **Access Manager Web Portal Manager**, and configure it.

8. Install WebSEAL. Run **install_amweb.exe**.

   If the configuration fails, run **pdconfig**, select **Access Manager WebSEAL**, and configure it.

   To avoid port conflicts with the HTTP server, you might want to stop the HTTP server that was installed with WebSphere Application Server. You might experience port conflicts if you choose 80 as the WebSEAL HTTP access port or 443 as the WebSEAL HTTPS access port.

9. Check that all Tivoli Access Manager components are configured using the **pdconfig** utility.

   Figure 14-5 shows the **pdconfig** output window running on a Microsoft Windows Server 2003.

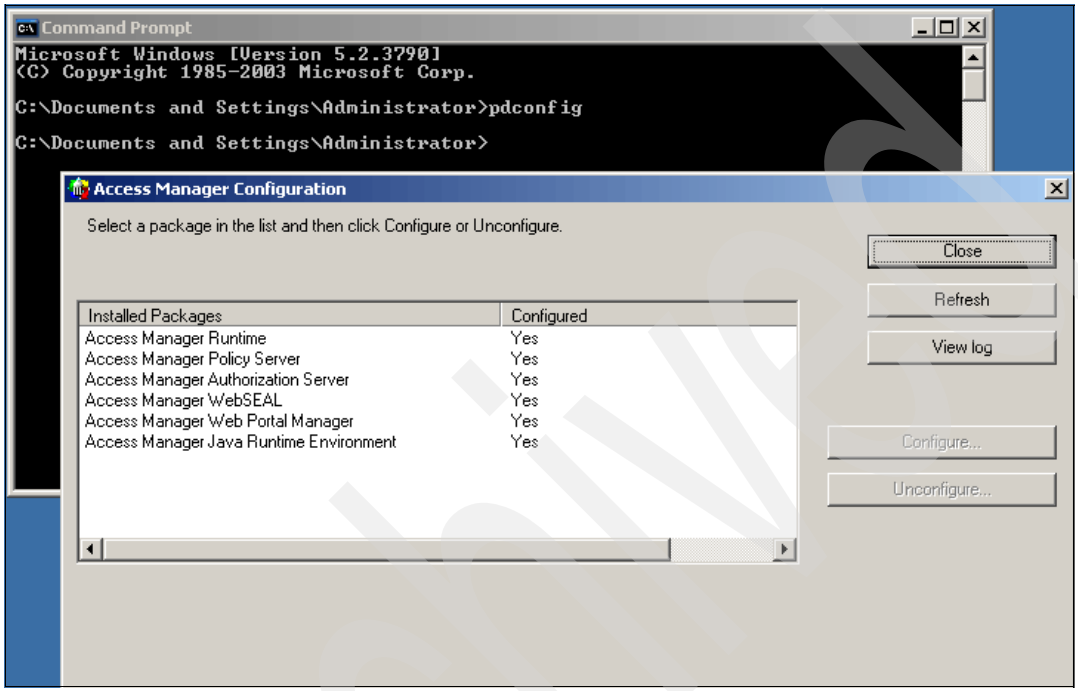


Figure 14-5 IBM Tivoli Access Manager pdconfig window

14.2.2 Configure Tivoli Access Manager for WebSphere

This section describes how to set up Tivoli Access Manager Version 5.1 for WebSphere Application Server for z/OS Version 5.1. In our case, WebSphere Application Server global security is not turned on yet and the Tivoli Access Manager installation is fresh.

Initial verifications

Verify that you have a Tivoli Access Manager policy server, a Tivoli Access Manager authorization server, a supported registry server, and WebSEAL configured in your environment. You can check this by running the **pdconfig** command.

Figure 14-5 shows the **pdconfig** output window running on a Microsoft Windows Server 2003.

Verify that WebSphere Application for z/OS Version 5.1 is installed and running. You can check that you can use the administrative console. By default, the administrative console is at the following URL:

```
http://<WAS_hostname>:9090/admin/
```

The version and the service level of your WebSphere Application Server appear in the About frame on the Welcome page. Figure 14-6 shows the administrative console welcome page. Note the About frame on the right side.

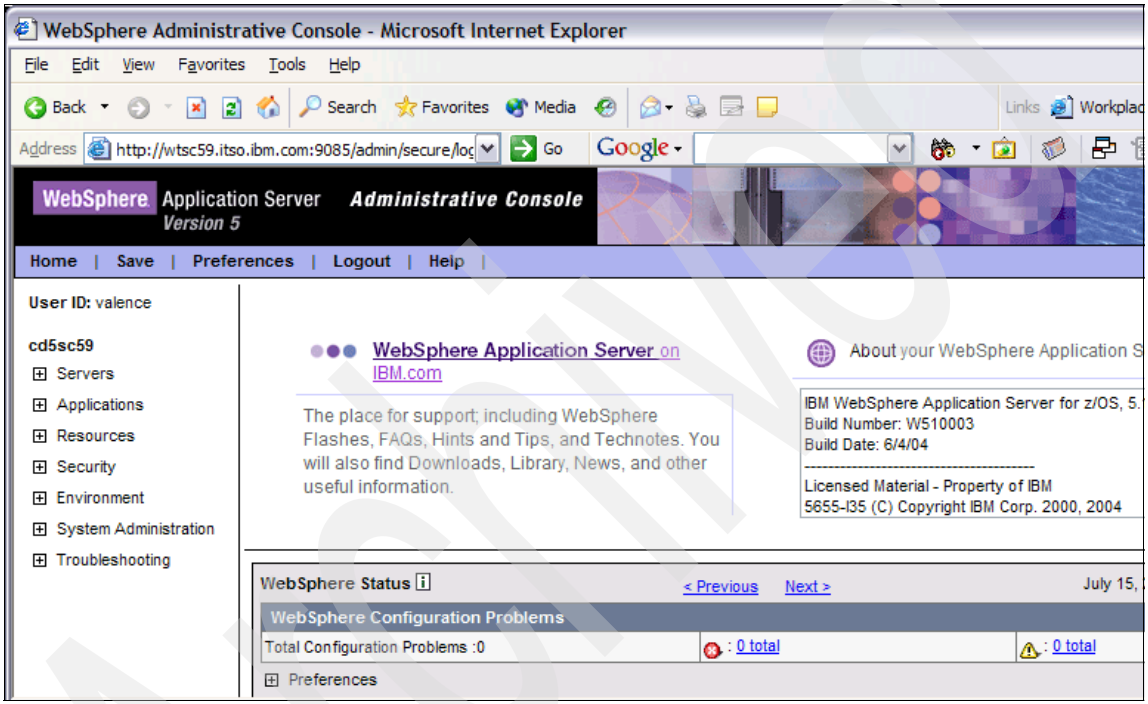


Figure 14-6 WebSphere Application Server Administrative Console

Note: If you configured your Tivoli Access Manager policy server against an existing WebSphere Application Server remote registry, you have to import users and groups to the Tivoli Access Manager registry. This can be done with the Web Portal Manager interface or with the **pdadmin** command line utility. For more information about how to do this, refer to *IBM Tivoli Access Manager Base Administration Guide, V5.1*, SC32-1360-00.

Creating the Tivoli Access Manager user for WebSphere

You now have to create a Tivoli Access Manager user that will perform tasks specific to WebSphere Application Server.

For any Tivoli Access Manager operation, you have the possibility to use the Web Portal Manager interface or the **pdadmin** command line utility.

To operate using the Web Portal Manager (WPM), you need to have it installed, and then point to the following URL:

```
http://<WPM_hostname>:9080/pdadmin/
```

Figure 14-7 shows the welcome window.

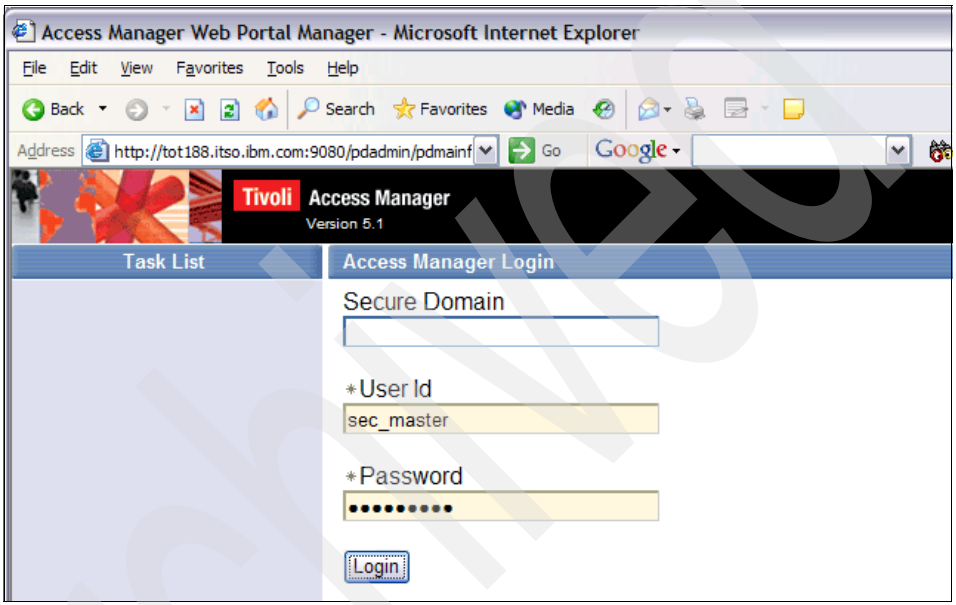


Figure 14-7 Web Portal Manager interface welcome window

Enter the Tivoli Access Manager administrator user ID and password and then click **Login**.

On the left side in the Task List, select **User** and then **Create User**. Here is the user we create:

- User Id: `wasuser`
- First Name: `was`
- Last Name: `user`
- Password: `wasuser2004`
- Description: `TAM administrative user for WAS`
- Registry UID: `cn=wasuser,o=itso`

The registry UID must have as a suffix the suffix you defined when configuring the IBM Directory Server (LDAP). The “Is GSO User” option allows global

sign-on (GSO) capabilities, which enables access to multiple Web resources through a single login.

Figure 14-8 shows the Web Portal Manager window when creating a new user.

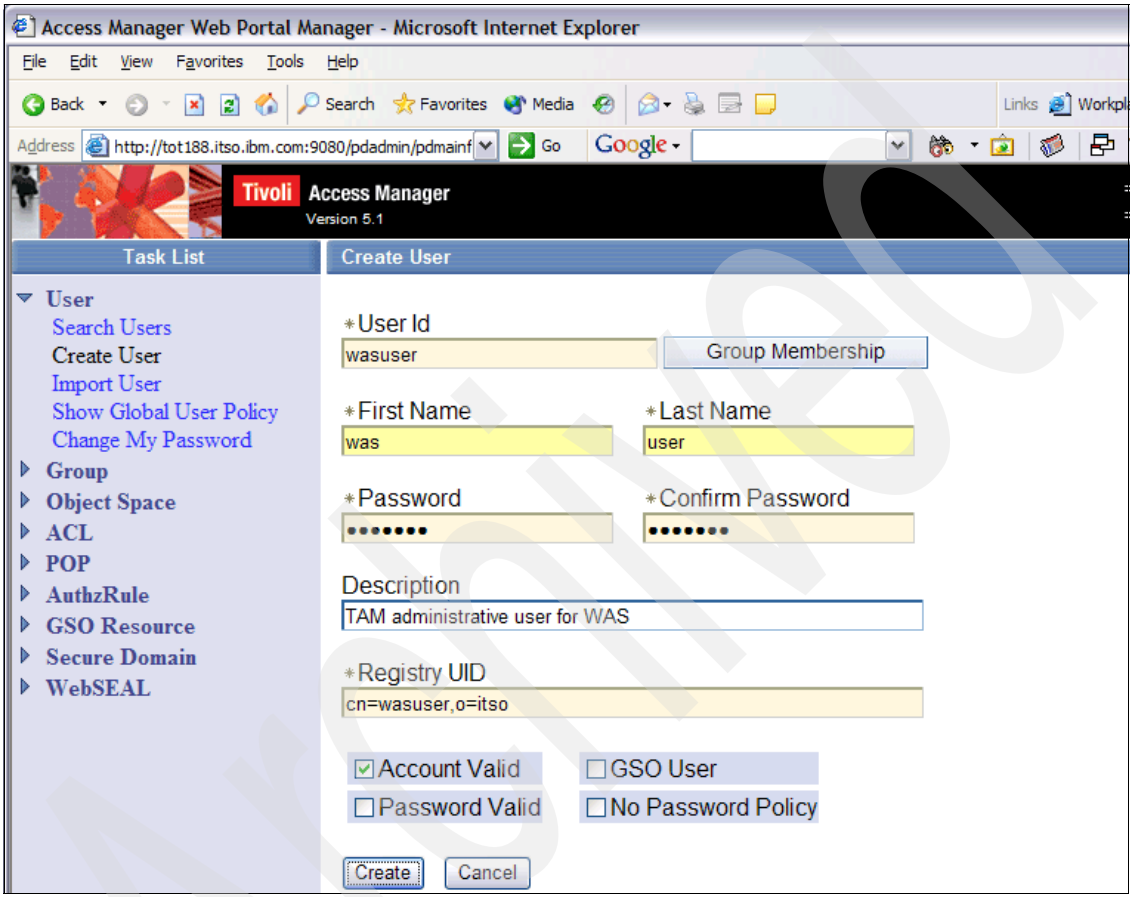


Figure 14-8 Web Portal Manager Create User window

Click **Create**. You should now see a “The user was created successfully” message.

If you choose to use the **pdadmin** command line utility, type commands similar to the following commands:

```
pdadmin -a sec_master
user create wasuser cn=wasuser,o=itso was user wasuser2004
user modify wasuser account-valid yes
user show wasuser
exit
```

14.2.3 Configure WebSphere for Tivoli Access Manager

We now describe how to configure WebSphere Application Server to interact with the Tivoli Access Manager policy server and authorization server. This consists of two steps:

1. Configuring the Tivoli Access Manager Java Runtime Environment component
2. Configuring WebSphere Application Server to use Tivoli Access Manager

Configuring the Tivoli Access Manager Java Runtime Environment component

The Tivoli Access Manager Java Runtime Environment (JRE) component is part of WebSphere Application Server for z/OS. It contains the core classes needed to make Java applications be able to communicate with the Tivoli Access Manager policy server and authorization server. It enables Java applications to manage and use Tivoli Access Manager security.

In this section, we use the **pdjrtecfg** utility. It configures and reconfigures the logon to UNIX System Services with a user that has read and write access to the $WAS_HOME directory. This can be either with Telnet or with OMVS. Complete the following steps:

1. Go to the <WAS_Config_Dir>\appserver\bin directory and run the following command to set up the environment variables:

   ```
   . ./setupCmdLine.sh
   ```

   Important: The dot at the beginning of the above command is very important so that environment variables are set for your current shell.

   You can check if the environment variables have been set properly using the **env** command. This command lists all the environment variables for your current shell.

2. If your WAS_HOME environment variable is not already set, set it with a command similar to the following command:

   ```
   export WAS_HOME=<WAS_Config_Dir>/appserver
   ```

3. Create a UNIX script file called PDJrteCfg.sh. Example 14-1 on page 436 describes the file contents.

Example 14-1 PDJrteCfg.sh sample script

```
command="$JAVA_HOME/bin/java -Dfile.encoding=ISO8859-1
                -Dws.output.encoding=CP1047
                -Xnoargsconversion
                -Dpd.home=$WAS_HOME/java/jre/PolicyDirector
                -cp $WAS_HOME/java/jre/lib/ext/PD.jar
                 com.tivoli.pd.jcfg.PDJrteCfg
                -action config
                -cfgfiles_path $WAS_HOME/java/jre
                -host <TAM_hostname>
                -was"
$command
```

Customize the <TAM_hostname> value to fit with your environment.

4. Change the UNIX permission bit to be able to execute this script with the following command:

   ```
   chmod +x PDJrteCfg.sh
   ```

5. Run the script with the following command:

   ```
   ./PDJrteCfg.sh
   ```

   You should now see the following output:

   ```
   Configuration of Access Manager Java Runtime Environment is in progress.
   This might take several minutes.
   The /usr/lpp/java/J1.4/PolicyDirector directory does not exist.
   Creating the directory.
   Configuration of Access Manager Java Runtime Environment completed
   successfully.
   ```

The **PDJrteCfg** utility configures or reconfigures the Tivoli Access Manager JRE component. The Tivoli Access Manager JRE component enables Java applications to manage and use Tivoli Access Manager security. This utility validates the Java JRE location then creates some configuration files in the $WAS_HOME/java/jre/PolicyDirector directory.

Note: You must do this operation for any JDK that is used by WebSphere Application Server. Therefore, in a Network Deployment environment, you must configure the JDK used by the deployment manager. For this purpose, repeat the previous steps and set the WAS_HOME environment variable to <WAS_Config_Dir>/DeplMgr.

Configuring WebSphere Application Server to use Tivoli Access Manager

In this section, you will run two utilities to configure WebSphere Application Server to use Tivoli Access Manager for authentication.

The first utility is the **SvrSslCfg** utility. It creates a user account representing the WebSphere Application Server in the Tivoli Access Manager user registry. It also creates a configuration file and a Java key store file, which securely stores a client certificate locally on the application server. This client certificate permits callers to make authenticated use of Tivoli Access Manager services.

The second utility is the **pdwascfg** utility. It configures the Tivoli Access Manager for WebSphere component to the default security authorization provider for WebSphere Application Server. During this process, this component also joins to the Tivoli Access Manager domain. Complete the following steps:

1. Log on to UNIX System Services with a user that has read and write access to the $WAS_HOME directory. This can be either with telnet or with OMVS.
2. Go to the <WAS_Config_Dir>\appserver\bin directory and run the following command to set up the environment variables:

   ```
   . ./setupCmdLine.sh
   ```

   Important: The dot at the beginning of the above command is very important so that environment variables are set for your current shell.

   You can check if the environment variables have been set properly using the **env** command. This command lists all the environment variables for your current shell.
3. If your WAS_HOME environment variable is not already set, set it with a command similar to the following command:

   ```
   export WAS_HOME=<WAS_Config_Dir>/appserver
   ```
4. Create a UNIX script file called SvrSslCfg.sh. Example 14-2 describes the content of this file.

Example 14-2 SvrSslCfg.sh sample script

```
CLASSPATH=$WAS_HOME/java/jre/lib/ext/PD.jar:$WAS_CLASSPATH
command="java
     -cp $CLASSPATH
     -Dpd.cfg.home=$WAS_HOME/java/jre
     -Dfile.encoding=ISO8859-1
     -Dws.output.encoding=CP1047
     -Xnoargsconversion
      com.tivoli.pd.jcfg.SvrSslCfg
```

```
    -action config
    -admin_id sec_master
    -admin_pwd <tam_password>
    -appsvr_id <app_srv_id>
    -policysvr <tam_policy_srv_hostname>:7135:1
    -port 7135
    -authzsvr <tam_authorization_srv_hostname>:7136:1
    -mode remote
    -cfg_file $WAS_HOME/java/jre/PdPerm.properties
    -key_file $WAS_HOME/java/jre/key_store.kdb
    -cfg_action create"
$command
```

Customize the <tam_password>, <app_srv_id>, <tam_policy_srv_hostname>, and <tam_authorization_srv_hostname> values to fit with your environment. The <app_srv_id> value is an identifier for the application server you are configuring. We choose BS05 (base server 05) in our configuration. This name will appear in the Tivoli Access Manager users list.

5. Change the UNIX permission bit to be able to execute this script with the following command:

```
chmod +x SvrSslCfg.sh
```

6. Run the script with the following command:

```
./SvrSslCfg.sh
```

You should now see the following output:

```
The configuration completed successfully.
```

After running the **SvrSslCfg** utility successfully on WebSphere Application Server, a user account and server entries, representing WebSphere Application Server, are created in the Tivoli Access Manager user registry.

You can check the user account creation with the Web Portal Manager interface. Figure 14-9 on page 439 shows the created entry for our environment.

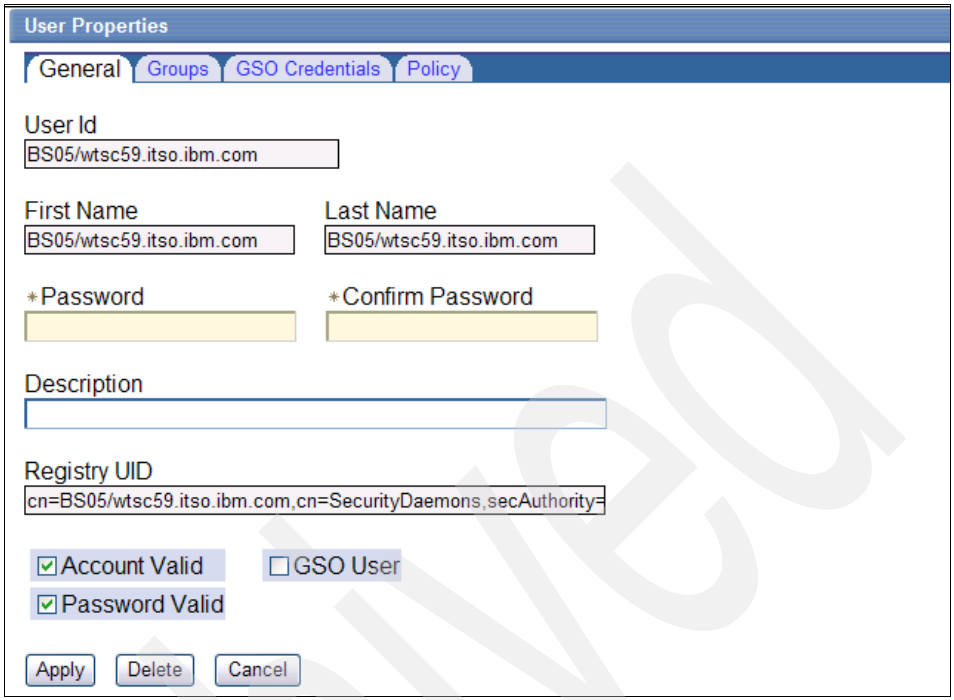


Figure 14-9 Web Portal Manager app_srv_id entry

In addition, you can check this entry using the **pdadmin** command line utility using commands similar to the following commands:

```
pdadmin -a sec_master
show BS05/wtsc59.itso.ibm.com
server list
exit
```

7. Go to the $WAS_HOME/java/jre directory and make sure that the WebSphere Application Server controller and servant users have read permissions to the PdPerm.properties and key_store.kdb files. For this purpose, you can change the permissions or change the owner of the files. We choose to use a command similar to the following command:

```
chown WSADMIN:WASADM PdPerm.properties key_store.kdb
```

8. Create a UNIX script file called pdwascfg.sh. Example 14-3 on page 440 describes the contents of this file.

Example 14-3 pdwascfg.sh sample script

```
export PDWAS_HOME=$WAS_HOME
export JDK_DIR=$JAVA_HOME
command="$WAS_HOME/bin/pdwascfg
     -action configWAS5
     -remote_acl_user <app_srv_id>
     -sec_master_pwd <tam_password>
     -pdmgrd_host <tam_policy_srv_hostname>
     -pdacld_host <tam_authorization_srv_hostname>
     -embedded true
     -was_home $WAS_HOME
     -amwas_home $PDWAS_HOME
     -action_type local"
$command
```

Customize the <app_srv_id>, <tam_password>, <tam_policy_srv_hostname>, and <tam_authorization_srv_hostname> values to fit with your environment. We choose the same <app_srv_id> value (BS05) for the remote ACL user. This user is employed for all communication with the Tivoli Access Manager servers.

9. Change the UNIX permission bit to be able to execute this script with the following command:

   ```
   chmod +x pdwascfg.sh
   ```

10. Run the script with the following command:

    ```
    ./pdwascfg.sh
    ```

    No message tells you if the configuration is successful or not. Nevertheless, if any error is encountered, you will get an error message.

    The **pdwascfg** utility configures WebSphere Application Server to use Tivoli Access Manager for WebSphere as the authorization vendor.

    For a confirmation that this utility worked correctly, you can make sure that the $PDWAS_HOME/etc/PDWAS.properties and $PDWAS_HOME/config/PD_WAS.prop files have been created.

    Moreover, the remote_acl_user is added to the remote-acl-users group. You can check this using the Web Portal Manager interface. Figure 14-10 on page 441 shows the created entry for our environment.

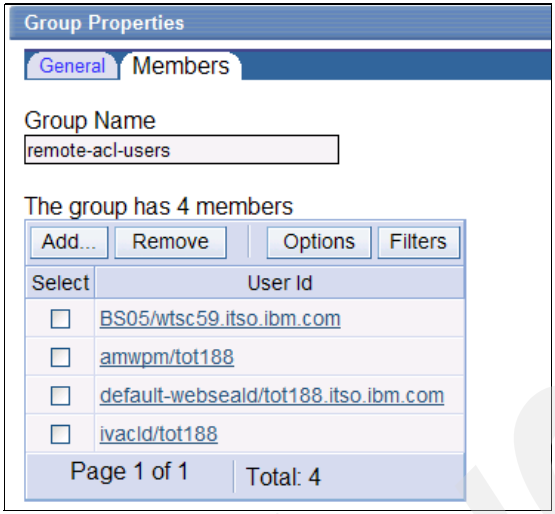


Figure 14-10 Web Portal Manager remote-acl-users group members

In addition, you can check this entry using the **pdadmin** command line utility using commands similar to the following commands:

```
pdadmin -a sec_master
group show-members remote-acl-users
exit
```

Note: If you are using a deployment manager configuration, you must run the above two scripts (SvrSslCfg.sh and pdwascfg.sh) twice: one time for the base configuration and again to configure the deployment manager. For this purpose, repeat the previous steps and set the `WAS_HOME` environment variable to `<WAS_Config_Dir>/DeplMgr`.

During the second run for the deployment manager and the **SvrSslConfig** utility, you need to change the value of `appsvr_id`, for example adding "`_nd`" to the end, or else the utility will fail, because the Tivoli Access Manager server will think that the application server has already been configured.

14.2.4 Enable WebSphere Application Server security to use Tivoli Access Manager

We now enable WebSphere Application Server security to use Tivoli Access Manager for authorization, to use the Tivoli Access Manager user registry, and to use Tivoli Access Manager for account policies.

Complete the following steps:

1. We recommend that you back up your configuration. Type and execute commands similar to the following commands:

```
cd <was_config_dir>/appserver/bin
. ./setupCmdLine.sh
./backupConfig.sh /u/valence/was_config.backup -user wsadmin -password
wsadmin -nostop
```

   You should see messages similar to the following messages:

```
ADMU0116I: Tool information is being logged in file
/WebSphere/BS05/appserver/logs/backupConfig.log
ADMU5001I: Backing up config directory /WebSphere/BS05/appserver/config
to file /u/valence/was_config.backup
.....................................................................
.................
ADMU5002I: 89 files successfully backed up
```

2. Start WebSphere Application Server if it is not already started.

3. Add a grant codeBase definition statement for the PD.jar file in the server.policy file. Tivoli Access Manager core classes in the PD.jar file need permissions when Java security is enabled. Use the Policy Tool application (.../java/jre/bin/policytool.exe) or an editor to modify the $WAS_HOME/properties/server.policy file by appending the following statement to the bottom of the file:

```
grant codeBase "file:${was.install.root}/java/jre/lib/ext/PD.jar" {
permission java.security.AllPermission;
};
```

4. Log on to the WebSphere administrative console. By default, the administrative console is at the following URL:

```
http://<WAS_hostname>:9090/admin/
```

5. If global security is already turned on, disable it. Navigate to **Security** → **Global Security** and clear the **Enabled** options. Then, restart the server.

6. Add the PDDEFAULTCONFIG and pd.cfg.home variables to the Java virtual machine (JVM) configuration for the control process. These variables must be added to the deployment manager controller if applicable and to the base server controller. Navigate to **Servers** → **Application Servers** → **<server_name>** → **Process Definition** → **Control** → **Java Virtual Machine** → **Custom Properties** → **New**.

   Add the following two variables:

   – `PDDEFAULTCONFIG`: Specify the fully qualified file name for the configuration file that you specified when you ran the **`SvrSslCfg`** utility. This file name is specified using the **`-cfg_file`** option. For example, in our environment, it is /WebSphere/BS05/appserver/java/jre/PdPerm.properties.

– `pd.cfg.home`: Specify the Tivoli Access Manager configuration home directory that you specified when you ran the **`PDJrteCfg`** utility. This file name is specified using the **`-cfgfiles_path`** option. For example, in our environment, it is /WebSphere/BS05/appserver/java/jre.

Figure 14-11 shows the WebSphere administrative console when adding the `PDDEFAULTCONFIG` variables.

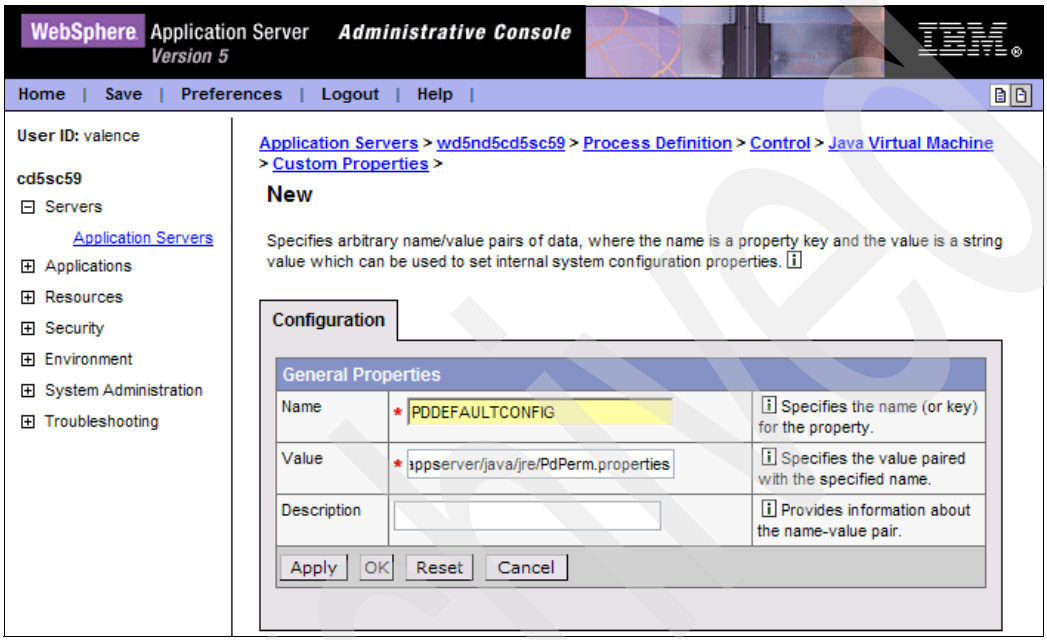


Figure 14-11 WebSphere Administrative Console and PDDEFAULTCONFIG

7. We now do the same operation for the servant process. Add the PDDEFAULTCONFIG and pd.cfg.home variables to the Java virtual machine (JVM) configuration for the servant process. These variables must be added to the deployment manager servant if applicable and to the base server servant. Navigate to **Servers** → **Application Servers** → **<server_name>** → **Process Definition** → **Servant** → **Java Virtual Machine** → **Custom Properties** → **New**.

   Add the following two variables:

   – `PDDEFAULTCONFIG`: Specify the fully qualified file name for the configuration file that you specified when you ran the **`SvrSslCfg`** utility. This file name is specified using the **`-cfg_file`** option. For example, in our environment, it is /WebSphere/BS05/appserver/java/jre/PdPerm.properties.

– `pd.cfg.home`: Specify the Tivoli Access Manager configuration home directory that you specified when you ran the **`PDJrteCfg`** utility. This file name is specified using the **`-cfgfiles_path`** option. For example, in our environment, it is WebSphere/BS05/appserver/java/jre.

Figure 14-12 shows the administrative console at this point.

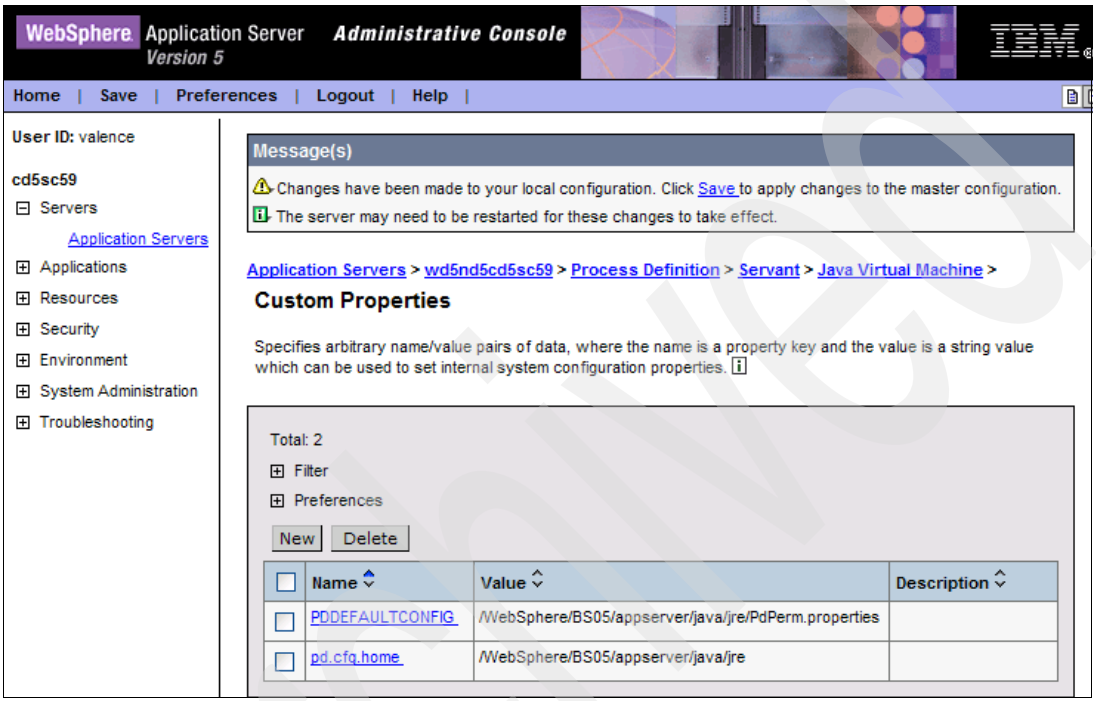


Figure 14-12 WebSphere Administrative Console and JVM variables

8. You can now click **Save**, propagate the changes to all nodes if applicable, and click **Save** to save the master configuration.
9. We now configure the LDAP user registry. Navigate to **Security** → **User Registries** → **LDAP**. Configure LDAP properties similar to the following properties:
   – Server User ID: `cn=wasuser,o=itso` (user defined in “Creating the Tivoli Access Manager user for WebSphere” on page 432)
   – Server User Password: `wasuser2004`
   – Type: `IBM_Directory_Server`
   – Host: `tot188.itso.ibm.com` (IBM Directory Server LDAP server host name)
   – Port: `389`
   – Base Distinguished Name (DN): `o=itso (suffix DN)`

- Bind Distinguished Name (DN): `cn=root (administrator DN)`
- Bind Password: `secret`
- Search Timeout: `120`
- Reuse Connection: Selected
- Ignore Case: Selected
- SSL Enabled: Not selected
- SSL Configuration: `cellname/DefaultSSLSetting`
- Use Tivoli Access Manager for Account Policies: Not selected (this option will be selected later)

Then, click **OK**.

10. You can now click **Save**, propagate the changes to all nodes if applicable, and click **Save** to save the master configuration.
11. We now configure LTPA. Navigate to **Security** → **Authentication Mechanisms** → **LTPA**. Specify a new Password used to encrypt and decrypt the LTPA keys. Then enter your password again in Confirm Password (secret in our environment). Then, click **Apply**.
12. Click **Single Signon (SSO)** under Additional Properties. From the Single Signon (SSO) page, select **Enable single signon** and click **Apply**.
13. , (Optional) If you want to use single sign-on, enter the single sign-on Domain Name System (DNS), and click **OK**.
14. We now enable global security. Navigate to **Security** → **Global Security**. Configure the following properties and leave the other properties unchanged:
    - Enabled: Selected
    - Use domain qualified user IDs: Selected
    - Active authentication mechanism: **LTPA**
    - Active user registry: **LDAP**

    Then, click **OK**.

    Figure 14-13 on page 446 shows the administrative console at this point.

Figure 14-13 WebSphere Administrative Console and Enabled option selected

15. Click **Save**, propagate the changes to all nodes if applicable, and click **Save** to save the master configuration.
16. Click **Logout** to log out of the WebSphere Application Server administrative console.
17. Open the administrative console and log in again.
18. Navigate to **Security** → **User Registries** → **LDAP**. Select the **Use Tivoli Access Manager for Account Policies** option and click **OK**.
19. Click **Save**, propagate the changes to all nodes if applicable, and click **Save** to save the master configuration.
20. Click **Logout** to log out of the WebSphere Application Server administrative console.

Do not restart WebSphere Application Server yet. Simply proceed to the following section to continue the configuration.

14.2.5 Migrate WebSphere Application Server security settings

After enabling WebSphere Application Server security to use Tivoli Access Manager, you have to migrate the security definitions to Tivoli Access Manager. We use the **migrateEAR5** utility to convert security role definitions to Tivoli Access Manager protected objects.

The **migrateEAR5** utility must be run against the following policy definition files:

- The adminconsole.ear file, which contains the application.xml file that includes the administrative console Web address (resource to protect) and the administrative roles required to use the administrative console
- The admin-authz.xml file, which contains users and their assigned administrative roles
- The naming-authz.xml file, which contains roles to control access to the WebSphere name space

If you are using the Network Deployment configuration, replace “appserver” with “DeplMgr” in the following steps. Complete the following steps:

1. Go to the <WAS_Config_Dir>\appserver\bin directory and run the following command to set up the environment variables:

   ```
   . ./setupCmdLine.sh
   ```

   You can check if the environment variables have been set properly using the **env** command. This command lists all the environment variables for your current shell.

2. If your WAS_HOME environement variable is not already set, set it with a command similar to the following command:

   ```
   export WAS_HOME=<WAS_Config_Dir>/appserver
   ```

3. Create a UNIX script file called migrateEAR5AdminCon.sh. Example 14-4 shows the contents of this file.

Example 14-4 migrateEAR5AdminCon.sh sample script

```
export CLASSPATH=$WAS_HOME/java/jre/lib/ext/PD.jar:$WAS_CLASSPATH
export JDK_DIR=$JAVA_HOME
export PDWAS_HOME=$WAS_HOME
command="$WAS_HOME/bin/migrateEAR5
    -j $WAS_HOME/installedApps/<cell_name>/adminconsole.ear
    -e adminconsole
    -a sec_master
    -p <tam_password>
    -w wsadmin
    -d <ldap_suffix>
    -c file:$WAS_HOME/java/jre/PdPerm.properties"
$command
```

Customize the <cell_name>, <tam_password>, and <ldap_suffix> (o=itso in our example) values to fit with your environment.

4. Change the UNIX permission bit to be able to execute this script with the following command:

```
chmod +x migrateEAR5AdminCon.sh
```

5. Run the script with the following command:

```
./migrateEAR5AdminCon.sh
```

You should now see the following output:

```
AWXWS0021I   Logging all activity to the file .//pdwas_migrate.log.
AWXWS0051E   The migrate tool has successfully completed.
```

6. Create a Tivoli Access Manager WASADM group. This step is necessary so that you do not get any errors when migrating the admin-authz.xml file. For this purpose, you can use the Web Portal Manager (WPM) interface or the **pdadmin** command line utility.

From the WPM interface, select **Group** → **Create Group**. Enter the following information:

```
Group Name: <auth_group>
Registry GID: cn=<auth_group>,<ldap_suffix>
```

The <ldap_suffix> value is o=itso in our environment. To know the name of the group, you can open the $WAS_HOME/config/cells/<cell_name>/admin-authz.xml file, which is in ASCII. Then, look for the following XML tag: <groups xmi:id="GroupExt_1" name="WASADM"/>. The group is WASADM in our environment.

Figure 14-14 on page 449 shows the WPM interface at this point.

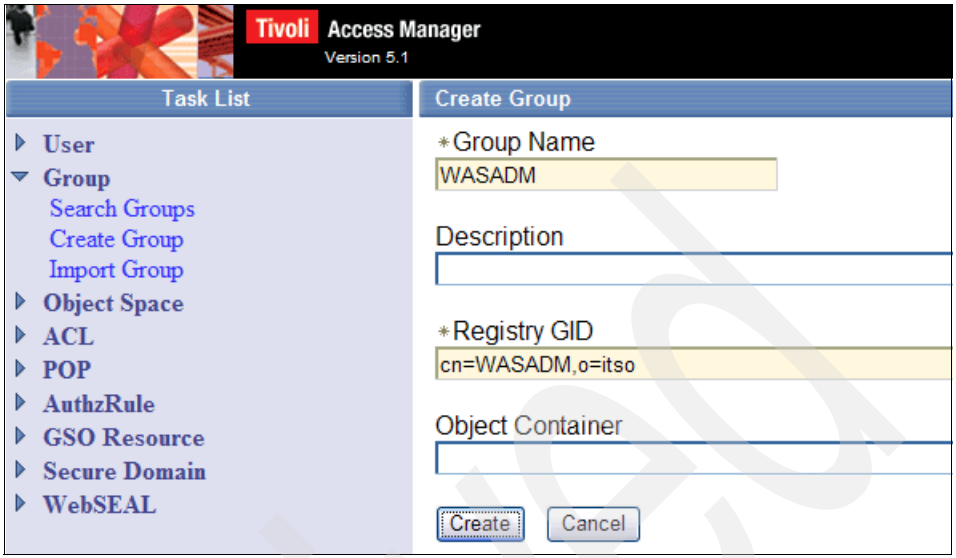


Figure 14-14 Tivoli Access Manager WPM Create Group

Then, click **Create**.

In addition, you can create this group using the **pdadmin** command line utility using commands similar to the following commands:

```
pdadmin -a sec_master
group create WASADM cn=WASADM,o=itso WASADM
group list * 200
exit
```

7. Create a UNIX script file called migrateEAR5AdminAuth.sh. Example 14-5 shows the contents of this file.

Example 14-5 migrateEAR5AdminAuth.sh sample script

```
export CLASSPATH=$WAS_HOME/java/jre/lib/ext/PD.jar:$WAS_CLASSPATH
export JDK_DIR=$JAVA_HOME
export PDWAS_HOME=$WAS_HOME
command="$WAS_HOME/bin/migrateEAR5
     -j $WAS_HOME/config/cells/<cell_name>/admin-authz.xml
     -a sec_master
     -p <tam_password>
     -w wsadmin
     -d <ldap_suffix>
     -c file:$WAS_HOME/java/jre/PdPerm.properties"
$command
```

Customize the <cell_name>, <tam_password>, and <ldap_suffix> (o=itso in our example) values to fit with your environment.

8. Change the UNIX permission bit to be able to execute this script with the following command:

```
chmod +x migrateEAR5AdminAuth.sh
```

9. Run the script with the following command:

```
./migrateEAR5AdminAuth.sh
```

You should now see the following output:

```
AWXWSO021I  Logging all activity to the file .//pdwas_migrate.log.
AWXWSO025W   The pdwas-admin group already exists, and its members are
Ýwsadmin~
.
AWXWSO051E   The migrate tool has successfully completed.
```

10. Create a UNIX script file called migrateEAR5NamAuth.sh. Example 14-6 shows the contents of this file.

Example 14-6 migrateEAR5NamAuth.sh sample script

```
export CLASSPATH=$WAS_HOME/java/jre/lib/ext/PD.jar:$WAS_CLASSPATH
export JDK_DIR=$JAVA_HOME
export PDWAS_HOME=$WAS_HOME
command="$WAS_HOME/bin/migrateEAR5
     -j $WAS_HOME/config/cells/<cell_name>/naming-authz.xml
     -a sec_master
     -p <tam_password>
     -w wsadmin
     -d <ldap_suffix>
     -c file:$WAS_HOME/java/jre/PdPerm.properties"
$command
```

Customize the <cell_name>, <tam_password>, and <ldap_suffix> (o=itso in our example) values to fit with your environment.

11. Change the UNIX permission bit to be able to execute this script with the following command:

```
chmod +x migrateEAR5NamAuth.sh
```

12. Run the script with the following command:

```
./migrateEAR5NamAuth.sh
```

You should now see the following output:

```
AWXWS0021I  Logging all activity to the file .//pdwas_migrate.log.
AWXWS0025W   The pdwas-admin group already exists, and its members are
Ýwsadmin~
.
AWXWS0051E   The migrate tool has successfully completed.
```

13. Define a Tivoli Access Manager wsadmin password. This can be done with the WPM interface or using the **pdadmin** command line utility. Select the **Password Valid** option.

    Figure 14-15 shows the WPM interface at this point.

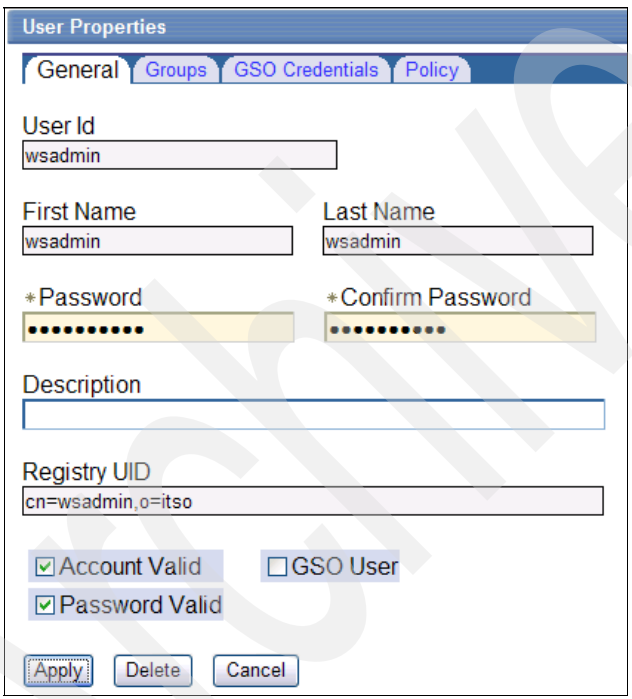


Figure 14-15 Tivoli Access Manager WPM wsadmin user

Then, click **Apply**.

In addition, you can create this group using the **pdadmin** command line utility using commands similar to the following commands:

```
pdadmin -a sec_master
user modify wsadmin password wsadmin2004
exit
```

14. Restart WebSphere Application Server to pick up changes.

    In the z/OS system log, you can see the WebSphere Application Server startup:

```
BBOO0222I SECJ0386I: Initializing registry to use Tivoli Access 712
Manager for authentication.
BBOO0222I SECJ0136I: Custom 713
Registry:com.ibm.ws.security.registry.ldap.TAMLdapRegistryImpl has been
initialized
```

You should now be able to log on to the administrative console using the wsadmin user and password. Check that you have 0 Total Configuration Problems.

14.2.6 Configure single sign-on between WebSEAL and WebSphere

Tivoli Access Manager WebSEAL can be used as a proxy server to provide access management and single sign-on capability to your Tivoli Access Manager for WebSphere protected applications. With such an architecture, WebSEAL authenticates users and forwards the collected credentials to WebSphere Application Server.

WebSphere Application Server for z/OS provides two methods to transfer the end-user identity:

- HTTP headers and the trust association interceptor (TAI)
- LTPA tokens

Single sign-on with trust association interceptor

In this section, we describe how to make WebSEAL authenticate users and forward the collected credentials to WebSphere Application Server in the form of an HTTP header. WebSphere trust association interceptors (TAIs) then intercept requests from WebSEAL, extract the end-user name from the iv-user HTTP header, and forward it to Tivoli Access Manager for WebSphere, which uses the information to construct the client credential information and authorize the user.

Important: The trust relationship between WebSEAL and WebSphere Application Server for z/OS is implemented through the use of a password encoded in a transport handler. Someone listening on the network could easily retrieve this password. This is the reason why we recommend that you encrypt the communication between WebSEAL and WebSphere with SSL, for example. In a trusted environment, you can use a non-encrypted connection between WebSEAL and WebSphere.

Figure 14-16 shows the authentication flow when using the trust association interceptor. Here is the explanation for this figure:

1. The browser makes a request for a secured WebSphere resource.
2. The WebSEAL server sends back a challenge, either an HTTP basic authentication or a form-based challenge.
3. A user name and password are supplied.
4. The WebSEAL product authenticates the user to Lightweight Directory Access Protocol (LDAP).
5. The modified request is forwarded by the WebSEAL product to WebSphere Application Server.
6. The plug-in TAI establishes trust between WebSphere Application Server and the WebSEAL server by using the negotiateAndValidateEstablishedTrust method. It uses the password from the incoming request and the user ID specified in the com.ibm.websphere.security.webseal.loginId variable.
7. The plug-in extracts the end-user credentials from the iv-user header field and passes it to WebSphere Application Server for authorization.

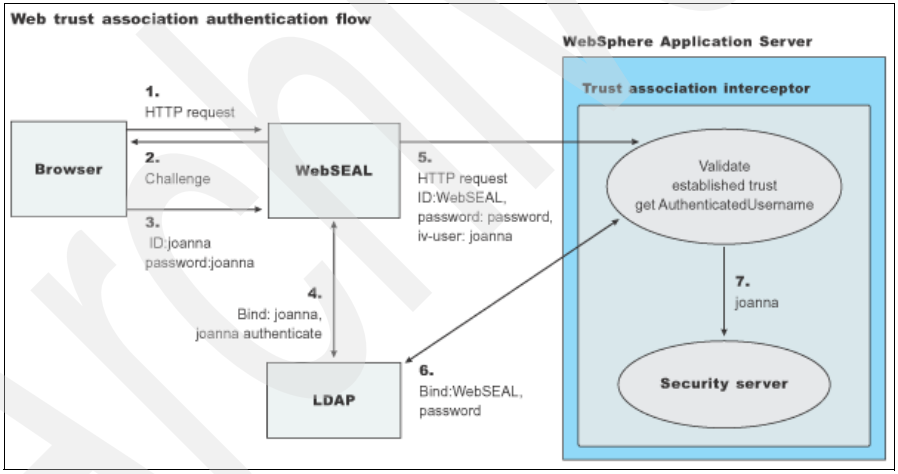


Figure 14-16 Web trust association authentication flow

We show here how to set up a non-SSL connection between WebSEAL and WebSphere. Keep in mind that we strongly recommend that you to use SSL for this connection if you are not in a trusted environment.

To set up a SSL connection between WebSEAL and WebSphere, you would need to specify the “-t ssl -B -U “username” -W “password”” options for the

WebSEAL junction and com.ibm.websphere.security.webseal.mutualSSL to true for the trust association interceptor. Complete the following steps:

1. One of the underlying security requirements of the TAI is the creation of a trusted user account in the Tivoli Access Manager user registry that WebSphere Application Server is configured to use. This is the ID and password that WebSEAL uses to identify itself to the WebSphere Application Server. To prevent potential vulnerabilities, do not use sec_master as the trusted user account and ensure that the password you use is unique. The trusted user account should be for the TAI only.

   You can use the Web Portal Manager interface to create this user or use a **pdadmin** command similar to the following command to create your WebSEAL user:

   ```
   user create usrwebseal "cn=usrwebseal,o=itso" "cn=usr" "sn=webseal"
   usrwebseal2004
   user modify usrwebseal account-valid yes
   user modify usrwebseal password-valid yes
   ```

2. Edit the WebSEAL configuration file Webseal_install_directory/etc/webseald-default.conf and set the SSO password for the user created in step 1 by setting the parameter basicauth-dummy-passwd=webseal_userid_passwd.

3. We now create the junction that will forward requests from WebSEAL to WebSphere Application Server. Although WebSEAL can be configured to pass the end-user identity in other ways, the iv-user header is the only one supported by the TAI. We recommend that communications over the junction use SSL for increased security.

   ```
   server task default-webseald-tot188.itso.ibm.com create -f -t tcp -h
   wtsc59.itso.ibm.com -p 9085 -b supply -c iv-user /wtsc59tai
   ```

   The `-b supply` junction will provide the original client user name and the webseal_userid_passwd in the basic authentication header of the forwarded request to WebSphere.

4. Log on to the WebSphere administrative console.

5. Navigate to **Security** → **Authentication mechanisms** → **LTPA** → **Trust Association**. Select the **Trust Association Enabled** option and click **Apply**.

6. Navigate to **Additional Properties** → **Interceptors**.

7. Click **com.ibm.ws.security.web.WebSealTrustAssociationInterceptor** to use the default WebSEAL interceptor.

8. Click **Custom Properties** under Additional Properties.

9. Click **New** to set up properties. Set up the following properties:
   - **com.ibm.websphere.security.webseal.hostnames**: List of any host names that are trusted. These are the hosts that the HTTP request go through before reaching WebSphere. In our environment, this is `tot188`.
   - **com.ibm.websphere.security.webseal.id**: List of HTTP headers that must be found in the incoming request to determine that the request comes from WebSEAL. It is `iv-user` in our environment.
   - **com.ibm.websphere.security.webseal.loginId**: WebSEAL user name defined in LDAP. This is `usrwebseal` in our environment.
   - **com.ibm.websphere.security.webseal.mutualSSL**: This configures the trust association interceptor so that trust with the reverse proxy is already validated using a mutually-authenticated Secure Sockets Layer (SSL) connection. If the value of the mutual SSL property is `true`, authentication is not performed for the single sign-on (SSO) user. In our environment, we do not use SSL between WebSEAL and WebSphere. Therefore, we need to make sure that authentication is performed for the SSO user. This value is `false` in our environment.
   - **com.ibm.websphere.security.webseal.ports**: List of ports that trusted hosts use. In our environment, WebSEAL is the only trusted host. Therefore, this list is `80,443` in our environment.

   Figure 14-17 shows the administrative console Custom Properties panel for the WebSEAL TAI.

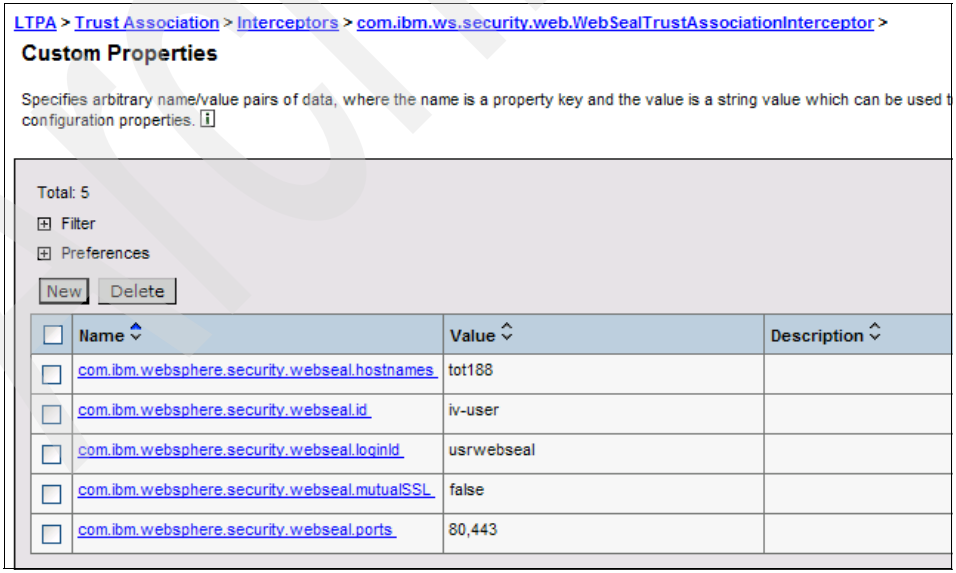


Figure 14-17 WebSphere administrative console WebSEAL TAI properties

10. Click **Save** and then click **Save** again to save the master configuration. Restart your server to initialize the trust association interceptor. In the syslog, you should read a message similar to the following message:

```
BBOOO222I SECJ0121I: Trust Association Init class 051
com.ibm.ws.security.web.WebSealTrustAssociationInterceptor loaded
successfully
BBOOO222I SECJ0122I: Trust Association Init Interceptor signature: 052
WebSeal Interceptor Version 1.1
```

You can now test your single sign-on solution.

We used the following URL to access SWIPE securely directly through the WebSphere HTTP transport handler:

```
http://wtsc59.itso.ibm.com:9085/IBMEBizWeb/secure/EJBCaller
```

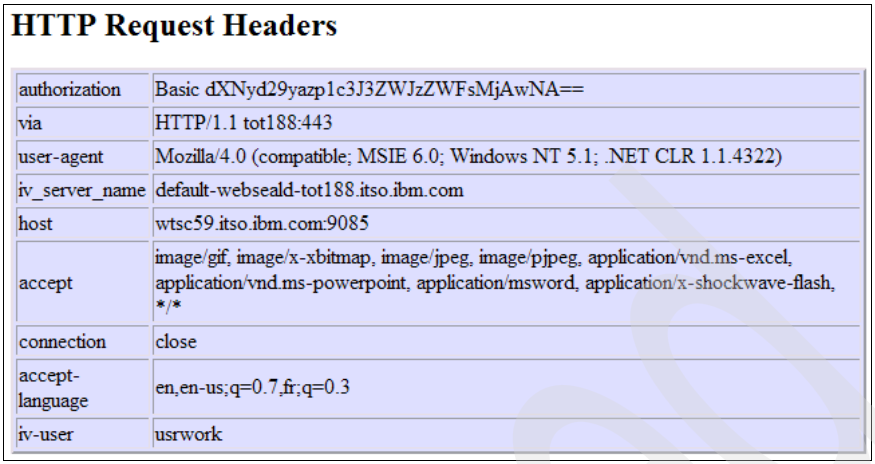
Now, we use the following address to access the same servlet through WebSEAL:

```
https://tot188.itso.ibm.com/wtsc59tai/IBMEBizWeb/secure/EJBCaller
```

`tot188.itso.ibm.com` is our WebSEAL DNS host name, and `/wtsc59tai` is our junction name.

You are first asked to authenticate at the WebSEAL level. Provide your user name and password (usrwork and usrwork2004 in our example). The user name is included in the forwarded request to WebSphere. Then, the TAI intercepts the request and provides the proper Java principal to WebSphere.

Using this EJBCaller servlet, it is now interesting to look at HTTP headers. Figure 14-18 on page 457 shows the HTTP headers in our environment.



HTTP Request Headers

authorization	Basic dXNyd29yazp1c3J3ZWJzZWFsMjAwNA==
via	HTTP/1.1 tot188:443
user-agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
iv_server_name	default-webseald-tot188.itso.ibm.com
host	wtsc59.itso.ibm.com:9085
accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
connection	close
accept-language	en,en-us;q=0.7,fr;q=0.3
iv-user	usrwork

Figure 14-18 SWIPE HTTP Headers using SSO with TAI

The authorization header shows that a basic authentication header is in the incoming request. The via header shows that the request comes from WebSEAL running on a machine with host name tot188. The incoming request possess the “iv-user” header added by WebSEAL. This header contains the user identity.

Single sign-on with LTPA tokens

WebSphere provides the cookie-based Lightweight Third Party Authentication (LTPA) mechanism. You can configure WebSEAL junctions to support LTPA and provide a single sign-on solution for clients.

When a user makes a request for a WebSphere resource, the user must first authenticate to WebSEAL. After successful authentication, WebSEAL generates an LTPA cookie on behalf of the user. The LTPA cookie, which serves as an authentication token for WebSphere, contains the user identity, key and token data, buffer length, and expiration information. This information is encrypted using a password-protected secret key shared between WebSEAL and the WebSphere server.

WebSEAL inserts the cookie in the HTTP header of the request that is sent across the junction to WebSphere. The back-end WebSphere server receives the request, decrypts the cookie, and authenticates the user based on the identity information supplied in the cookie. This cookie never reaches the end-user browser. It is included in the communication between WebSEAL and WebSphere only. WebSEAL maintains state by mapping incoming user information (SSL session ID, BA user ID) to the LTPA token.

Verify that all servers are configured as part of the same DNS domain. For example, if the DNS domain is specified as mycompany.com, SSO is effective with any Domino server or WebSphere Application Server on a host that is part of the mycompany.com domain, for example, a.mycompany.com and b.mycompany.com. Complete the following steps:

1. Log on to the WebSphere administrative console.
2. Navigate to **Security** → **Authentication mechanisms** → **LTPA** → **Single Signon (SSO)**.

   Select **Enable** if SSO is disabled.

   Enable the **Requires SSL** field if all of the requests are expected on HTTPS. In our test environment, we do not use HTTPS between WebSEAL and WebSphere.

   Enter the domain name in the Domain name field where SSO is effective. In our environment, WebSEAL runs on a machine with a DNS name of tot188.itso.ibm.com; WebSphere runs on a machine with a DNS name of wtsc59.itso.ibm.com. Therefore, our Domain name is `ibm.com`.

   Then, click **OK**.
3. Click **Save** to save your changes to the master configuration.

   Note: To save changes to the master configuration, we had to add the wasuser to the pdwas-admin group. Apparently, the wasuser makes changes to the security.xml file at this point. You can check this in the z/OS syslog.

4. Restart your entire WebSphere configuration (deployment manager, node agents, and J2EE servers).
5. Log on to the WebSphere administrative console.
6. Navigate to **Security** → **Authentication Mechanisms** → **LTPA**. Enter a file name including the full path in the **Key File Name** field. We choose `/WebSphere/BSO5/appserver/temp/ltpa.keys` in our environment. Make sure that the administrative console has write access to this directory. Then, click **Export Keys** to export LTPA keys in this file.
7. Transfer this file to your workstation. Make sure that transfers are made in binary format.
8. Edit this file and add a line containing the realm of your LTPA token creation server. The realm should be specified in the following manner:

   ```
   com.ibm.websphere.ltpa.Realm=<WebSEAL_hostname>:<port>
   ```

   Example 14-7 on page 459 shows the contents of our ltpa.keys file with the realm added.

Example 14-7 LTPA key file sample

```
#IBM WebSphere Application Server key file
#Wed Jul 21 22:09:19 GMT 2004
com.ibm.websphere.CreationDate=Wed Jul 21 22\:09\:19 GMT 2004
com.ibm.websphere.ltpa.PublicKey=ALRa2QCe3kFeWJ1aHpXVU2nXnpFBs...
com.ibm.websphere.ltpa.version=1.0
com.ibm.websphere.ltpa.Realm=tot188.itso.ibm.com:389
com.ibm.websphere.ltpa.3DESKey=ZowUUqWRG8BWFeZTsqE7Zen5BosoOUWzhbJvTvOVRrg\=
com.ibm.websphere.CreationHost=wtsc59.itso.ibm.com
com.ibm.websphere.ltpa.PrivateKey=9p1ROpnmhJJt3lzCeP1qNFbP9+Or...
```

9. Transfer this file to the WebSEAL server. Make sure that transfers are made in binary format.

10. Using the Web Portal Manager or the **pdadmin** command line utility, create a filter junction with LTPA cookies enabled. Specify the key file and a password for this key file. Use a **pdadmin** command similar to the following command:

    ```
    server task default-webseald-tot188.itso.ibm.com  create -t tcp -h
    wtsc59.itso.ibm.com -p 9085 -b filter -A -F "c:\ltpa.keys" -Z "secret"
    /wtsc59ltpa
    ```

    Figure 14-19 shows some of the fields you need to complete when creating the junction with the Web Portal Manager interface.

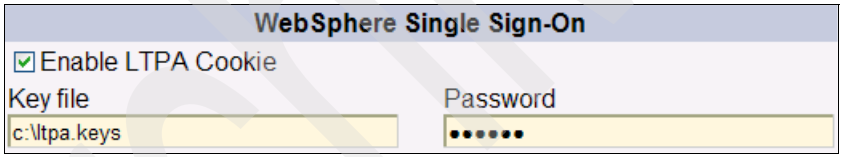


Figure 14-19 Junction creation LTPA Cookie enablement

You can now test your single sign-on solution.

We used the following URL to access SWIPE securely directly through the WebSphere HTTP transport handler:

```
http://wtsc59.itso.ibm.com:9085/IBMEBizWeb/secure/EJBCaller
```

Now, we use the following address to access the same servlet through WebSEAL:

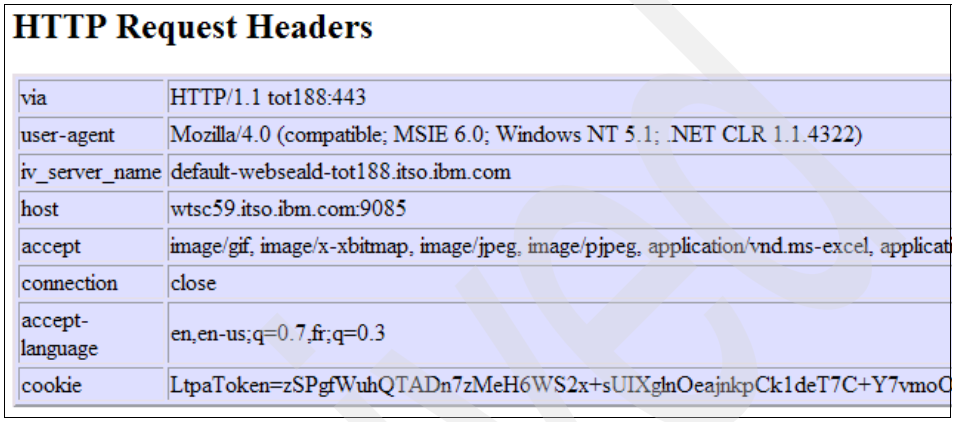
```
https://tot188.itso.ibm.com/wtsc59ltpa/IBMEBizWeb/secure/EJBCaller
```

`tot188.itso.ibm.com` is our WebSEAL DNS host name, and `/wtsc59ltpa` is our junction name.

You are first asked to authenticate at the WebSEAL level. Provide your user name and password (usrwork and usrwork2004 in our example). The user name

is included in an LTPA token (inside a cookie) in the forwarded to request to WebSphere. Then, WebSphere finds the cookie, reads it, and provides the proper Java principal.

Using this EJBCaller servlet, it is now interesting to look at HTTP headers. Figure 14-20 shows the HTTP headers in our environment.



HTTP Request Headers

via	HTTP/1.1 tot188:443
user-agent	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
iv_server_name	default-webseald-tot188.itso.ibm.com
host	wtsc59.itso.ibm.com:9085
accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, applicat
connection	close
accept-language	en,en-us;q=0.7,fr;q=0.3
cookie	LtpaToken=zSPgfWuhQTADn7zMeH6WS2x+sUIXglnOeajnkpCk1deT7C+Y7vmoC

Figure 14-20 SWIPE HTTP Headers using SSO with LTPA tokens

The via header shows that the request comes from WebSEAL running on a machine with host name `tot188`. The incoming request possess a cookie LTPA token added by WebSEAL. This LTPA token contains the user identity.

14.3 Using Tivoli Access Manager for WebSphere

In this section, we describe how to use Tivoli Access Manager and WebSphere Application Server when they are integrated together. We show how to:

- Create users or groups in such a configuration
- Create and secure J2EE roles
- Grant users or groups access to J2EE roles
- Deploy an application

14.3.1 Create users or groups with Tivoli Access Manager

In this section, we show how to create users or groups with Tivoli Access Manager. First, we describe how to create a new administrator who can use the administrative console with the Web Portal Manager interface. Second, we create users and groups for SWIPE using the **`pdadmin`** command line utility

Creating a new WebSphere administrator using the WPM interface

To create a new user who can access the WebSphere administrative console with an administrator role, complete the following steps:

1. Log on to the Tivoli Access Manager Web Portal Manager (WPM) interface.
2. Navigate to **User** → **Create User** to create a user. Figure 14-21 shows the user we create.

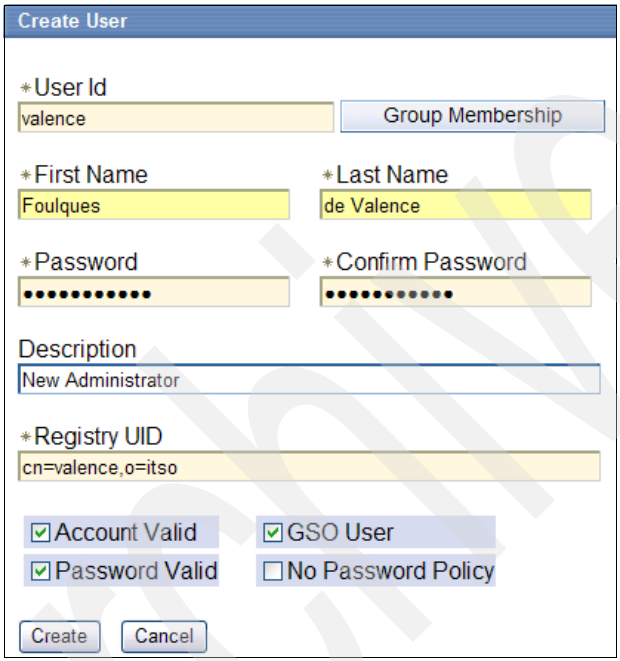


Figure 14-21 Tivoli Access Manager WPM interface Create User window

Then, click **Create**.

3. Navigate to **Group** → **Search Groups** → **pdwas-admin** → **Members** → **Add** to add your new user to the pdwas-admin group. Search for your user, select it, and then click **Add** and then **Done**. Figure 14-22 on page 462 shows the group members after the user has been added.

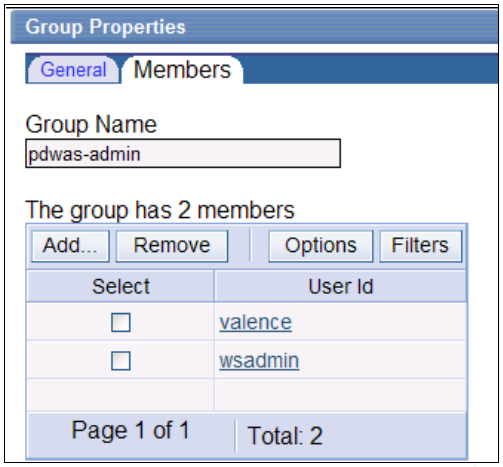


Figure 14-22 Tivoli Access Manager WPM interface Group Members window

Now, this new user can log on to the WebSphere administrative console with administrator privileges.

Why does it work? This new user is granted administrator privileges because it is a member of the pdwas-admin group. This pdwas-admin group is specified within the _WebAppServer_deployedResources_administrator_admin-authz_ACL and _WebAppServer_deployedResources_administrator_adminconsole_ACL ACLs with permissions to use the /WebAppServer/deployedResources/administrator/admin-authz and /WebAppServer/deployedResources/administrator/adminconsole protected objects.

Creating new SWIPE users and groups using pdadmin

The **pdadmin** command line utility enables you to do any operation you can do with the Web Portal Manager interface and even more.

To create users and groups for SWIPE using the **pdadmin** utility, complete the following steps:

1. Create a file called createusersgroups.txt containing the information in Example 14-8.

Example 14-8 The createusersgroups.txt file

```
user create usremp "cn=usremp,o=ITSO" "cn=usr" "sn=emp" usremp2004
user modify usremp account-valid yes
user modify usremp password-valid yes
user create usrwork "cn=usrwork,o=ITSO" "cn=usr" "sn=work" usrwork2004
user modify usrwork account-valid yes
```

```
user modify usrwork password-valid yes
user create mrsheen "cn=mrsheen,o=ITSO" "cn=mr" "sn=sheen" mrsheen2004
user modify mrsheen account-valid yes
user modify mrsheen password-valid yes
user create usrmgr "cn=usrmgr,o=ITSO" "cn=usr" "sn=mgr" usrmgr2004
user modify usrmgr account-valid yes
user modify usrmgr password-valid yes
user create usrceo "cn=usrceo,o=ITSO" "cn=usr" "sn=ceo" usrceo2004
user modify usrceo account-valid yes
user modify usrceo password-valid yes
user create rolemgr "cn=rolemgr,o=ITSO" "cn=role" "sn=mgr" rolemgr2004
user modify rolemgr account-valid yes
user modify rolemgr password-valid yes
user create roleceo "cn=roleceo,o=ITSO" "cn=role" "sn=ceo" roleceo2004
user modify roleceo account-valid yes
user modify roleceo password-valid yes

group create grpmgrs "cn=grpmgrs,o=itso" grpmgrs
group modify grpmgrs add (usrmgr rolemgr usrceo roleceo)
```

2. Run a command similar to the following command to run the createusersgroups.txt script:

   ```
   pdadmin -a sec_master createusersgroups.txt
   ```

   This command will request the sec_master password and then execute the contents of the createusersgroups.txt script.

   All the commands included in this script appear at execution time. There is no message saying that the execution is successful. If any errors occur, an additional error message is displayed.

3. Check that the users and groups have been added with commands similar to the following commands:

   ```
   user list * 200
   group list * 200
   ```

14.3.2 Create and secure roles with Tivoli Access Manager

In this section, we describe how to create and secure roles with Tivoli Access Manager.

A role is represented within Tivoli Access Manager by a protected object. You can see these protected objects by navigating with the Web Portal Manager interface to **Object Space** → **Browse Object Space** → **+/** → **+WebAppServer** → **+DeployedResources**. For example, /WebAppServer/deployedResources/Worker/SWIPE is a protected object that we will create for the Worker role of the SWIPE application.

Figure 14-23 shows the object space in our environment.

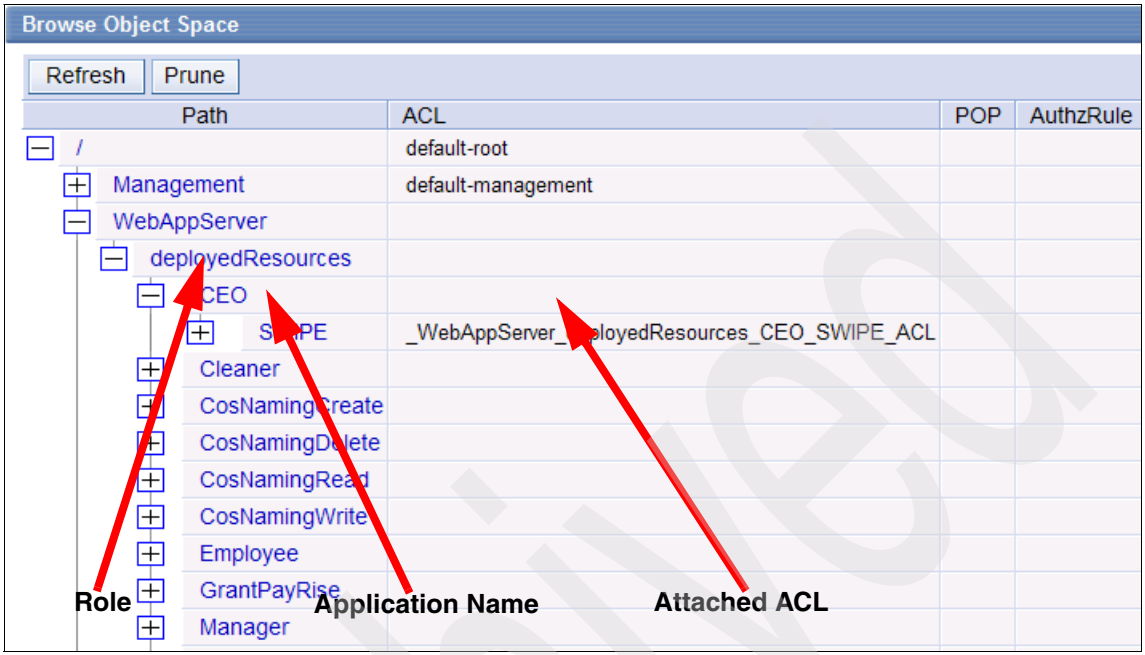


Figure 14-23 Tivoli Access Manager WPM interface Object Space window

The protected objects are protected within Tivoli Access Manager by access control lists (ACLs). An ACL controls what operations can be performed and who can perform these operations. For each protected object representing a role, you need to attach an ACL that specifies who can do what with this protected object. You can see these ACLs by navigating with the Web Portal Manager interface to **ACL** → **List ACL**. The ACL attached to the protected object for the Worker role will be _WebAppServer_deployedResources_Worker_SWIPE_ACL.

Therefore, creating and securing a role consists of three steps:

1. Create a protected object that represents the role.
2. Create an ACL to secure the protected object.
3. Attach the ACL to the protected object that you want to secure.

To create and secure roles for SWIPE using the **`pdadmin`** utility, complete the following steps:

1. Create a file called createsecureroles.txt containing the information shown in Example 14-9 on page 465.

Example 14-9 The createsecureroles.txt file

```
object create /WebAppServer/deployedResources/Employee/SWIPE "Created using
pdadmin" 0 ispolicyattachable yes
object create /WebAppServer/deployedResources/Worker/SWIPE "Created using
pdadmin" 0 ispolicyattachable yes
object create /WebAppServer/deployedResources/Manager/SWIPE "Created using
pdadmin" 0 ispolicyattachable yes
object create /WebAppServer/deployedResources/CEO/SWIPE "Created using pdadmin"
0 ispolicyattachable yes
object create /WebAppServer/deployedResources/Cleaner/SWIPE "Created using
pdadmin" 0 ispolicyattachable yes
object create /WebAppServer/deployedResources/GrantPayRise/SWIPE "Created using
pdadmin" 0 ispolicyattachable yes
object create /WebAppServer/deployedResources/PromoteWorkers/SWIPE "Created
using pdadmin" 0 ispolicyattachable yes

acl create _WebAppServer_deployedResources_Employee_SWIPE_ACL
acl create _WebAppServer_deployedResources_Worker_SWIPE_ACL
acl create _WebAppServer_deployedResources_Manager_SWIPE_ACL
acl create _WebAppServer_deployedResources_CEO_SWIPE_ACL
acl create _WebAppServer_deployedResources_Cleaner_SWIPE_ACL
acl create _WebAppServer_deployedResources_GrantPayRise_SWIPE_ACL
acl create _WebAppServer_deployedResources_PromoteWorkers_SWIPE_ACL

acl attach /WebAppServer/deployedResources/Employee/SWIPE
_WebAppServer_deployedResources_Employee_SWIPE_ACL
acl attach /WebAppServer/deployedResources/Worker/SWIPE
_WebAppServer_deployedResources_Worker_SWIPE_ACL
acl attach /WebAppServer/deployedResources/Manager/SWIPE
_WebAppServer_deployedResources_Manager_SWIPE_ACL
acl attach /WebAppServer/deployedResources/CEO/SWIPE
_WebAppServer_deployedResources_CEO_SWIPE_ACL
acl attach /WebAppServer/deployedResources/Cleaner/SWIPE
_WebAppServer_deployedResources_Cleaner_SWIPE_ACL
acl attach /WebAppServer/deployedResources/GrantPayRise/SWIPE
_WebAppServer_deployedResources_GrantPayRise_SWIPE_ACL
acl attach /WebAppServer/deployedResources/PromoteWorkers/SWIPE
_WebAppServer_deployedResources_PromoteWorkers_SWIPE_ACL
```

2. Run a command similar to the following command to run the createsecureroles.txt script:

   ```
   pdadmin -a sec_master createsecureroles.txt
   ```

   This command will request the sec_master password and then execute the contents of the createsecureroles.txt script.

All the commands included in this script appear at execution time. There is no message saying that the execution is successful. If any errors occur, an additional error message is displayed.

3. Check that the protected objects and ACLs have been created with commands similar to the following commands:

```
object show /WebAppServer/deployedResources
acl list
acl show _WebAppServer_deployedResources_Worker_SWIPE_ACL
```

Optionally, a cell name, a host name, or even a J2EE server name can be specified to differentiate and configure security differently for separate J2EE servers, for example. These are optional levels of granularity.

14.3.3 Grant user access to J2EE roles with Tivoli Access Manager

In this section, we describe how to grant access to a role with Tivoli Access Manager.

We assume here that you read 14.3.3, “Grant user access to J2EE roles with Tivoli Access Manager” on page 466.

After a protected object representing a role has been created, after an ACL has been created, and after the ACL has been attached to the protected object to protect it, you need to specify who can do what with this protected object. This is done with ACL entries. ACL entries define permissions to access a protected object for users and groups.

You can see these ACL entries by navigating with the Web Portal Manager interface to **ACL** → **List ACL**. Click an ACL to view it. Figure 14-24 on page 467 shows the ACL entries for the Worker role of the SWIPE application. We will create those ACL entries later.

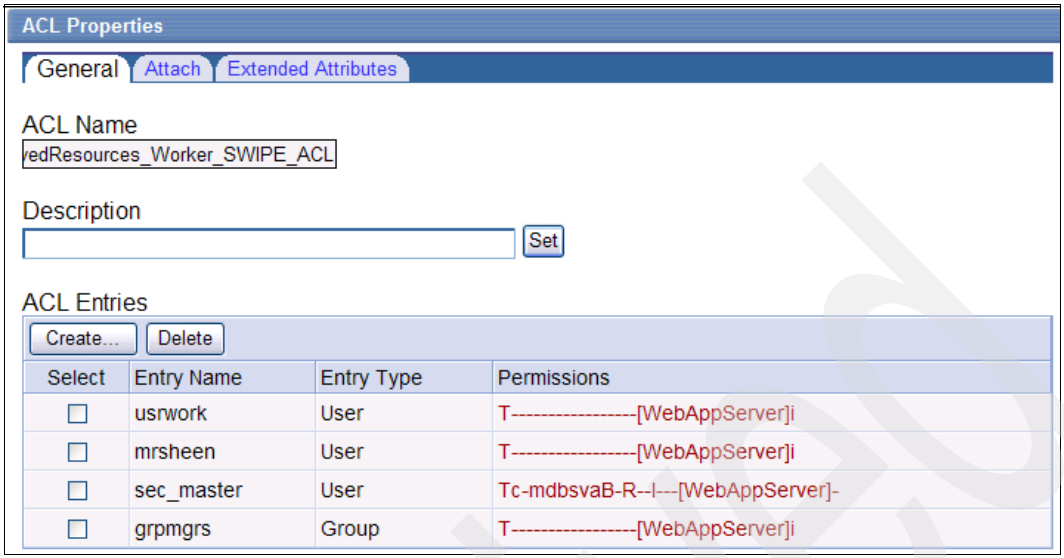


Figure 14-24 Tivoli Access Manager WPM interface ACL Properties window

Granting user or group access to specific role consists of one step: Within the ACL attached to the protected object representing the role, define an ACL entry that gives the proper permissions to the user or the group.

For more information about how to set up permissions and action groups, refer to *IBM Tivoli Access Manager Base Administration Guide, V5.1*, SC32-1360-00.

To grant user and group access to roles for SWIPE using the **pdadmin** utility, complete the following steps:

1. Create a file called grantrolesaccess.txt containing what the information shown in Example 14-10.

Example 14-10 The grantrolesaccess.txt file

```
acl modify _WebAppServer_deployedResources_Employee_SWIPE_ACL set user usremp
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_Employee_SWIPE_ACL set user usrwork
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_Employee_SWIPE_ACL set group grpmgrs
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_Employee_SWIPE_ACL set user mrsheen
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_Worker_SWIPE_ACL set user usrwork
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_Worker_SWIPE_ACL set user mrsheen
T[WebAppServer]i
```

```
acl modify _WebAppServer_deployedResources_Worker_SWIPE_ACL set group grpmgrs
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_Manager_SWIPE_ACL set group grpmgrs
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_CEO_SWIPE_ACL set user usrceo
T[WebAppServer]i
acl modify _WebAppServer_deployedResources_CEO_SWIPE_ACL set user roleceo
T[WebAppServer]i
```

2. Run a command similar to the following command to run the grantrolesaccess.txt script:

   ```
   pdadmin -a sec_master grantrolesaccess.txt
   ```

   This command will request the sec_master password and then execute the contents of the grantrolesaccess.txt script.

   All the commands included in this script appear at execution time. There is no message saying that the execution is successful. If any errors occur, an additional error message is displayed.

3. Check that ACL entries have been created with commands similar to the following commands:

   ```
   acl show _WebAppServer_deployedResources_Worker_SWIPE_ACL
   ```

14.3.4 Deploy an application: SWIPE

This section describes how to deploy an application when WebSphere Application server is integrated with Tivoli Access Manager for authorization.

WebSphere Application Server V5.1 provides a utility to migrate roles from an .ear file to the Tivoli Access Manager registry. This utility is named **migrateEAR5**. It is designed to migrate security policy information from deployment descriptors (from enterprise archive files) to Tivoli Access Manager for WebSphere Application Server V5.1. It creates protected objects representing roles, creates ACLs to protect them, and also creates ACL entries if any user or group mapping is present. With this utility, you do not need to know roles in advance. It will find roles, create protected objects, and create ACLs for you.

Migrating security roles from the .ear file to Tivoli Access Manager: migrateEAR5

This section describes how to use the **migrateEAR5** utility.

If protected objects representing roles for your application and ACLs have already been defined in Tivoli Access Manager, you do not need to run the *migrateEAR5* utility. Therefore, if you ran the createusersgroups.txt,

createsecureroles.txt, and grantrolesaccess.txt scripts from the preceding sections for SWIPE, you do not need to run the **migrateEAR5** utility on SWIPE.

Note: The **migrateEAR5** utility has limitations that are well described in Chapter 4, “Migrating security roles,” of *IBM Tivoli Access Manager for e-business IBM WebSphere Application Server Integration Guide, V5*, SC32-1368.

The **migrateEAR5** utility reads the application.xml file from the .ear file. It also reads the ibm-application-bnd.xmi and ibm-application-ext.xmi files if there are any.

If you are using the Network Deployment configuration, replace “appserver” with “DeplMgr” in the following steps. Complete the following steps:

1. If your .ear file contains an ibm-application-bnd.xmi file containing mappings between roles and specific users or groups, you first need to define those users and groups within Tivoli Access Manager. This file, if present, is located in the same directory as the application.xml file.

   For example, the SWIPE application contains mappings to the users MrSheen and CURCEO and the groups GRPMGR, GRPEMP, GRPFORM, GRPWORKR, and GRPSSL. Pay attention to the distinguished name (DN). It should have the following format:

   ```
   cn=<username or groupname>,<suffix>
   ```

   For example:

   ```
   cn=CURCEO,o=itso
   ```

2. Transfer the .ear file to the host system in a directory of your choice. It is /u/valence in our environment.
3. Create a UNIX script file called migrateEAR5MyApp.sh. Example 14-11 shows the contents of this file.

Example 14-11 migrateEAR5MyApp.sh sample script

```
export WAS_HOME=/WebSphere/BS05/appserver
usage() {
   echo
   echo " Usage: migrateEAR5MyApp.sh <tamPwd> <earFile>"
   echo "    where <tamPwd> is the TAM sec_master password"
   echo "          <earFile> is the ear file to migrate"
   echo
   exit 0;
}
if [ -z "$1" ]
then
  usage
```

```
fi
if [ -z "$2" ]
then
  usage
fi
echo "BEGIN migrateEAR5MyApp.sh"
. $WAS_HOME/bin/setupCmdLine.sh
export TAM_PASSWD=$1
export EAR_FILE=$2
export CLASSPATH=$WAS_HOME/java/jre/lib/ext/PD.jar:$WAS_CLASSPATH
export JDK_DIR=$JAVA_HOME
export PDWAS_HOME=$WAS_HOME
echo "Migrating "$2"..."
command="$WAS_HOME/bin/migrateEAR5
     -j $EAR_FILE
     -a sec_master
     -p $TAM_PASSWD
     -w wsadmin
     -d o=itso
     -c file:$WAS_HOME/java/jre/PdPerm.properties"
$command
echo "END migrateEAR5MyApp.sh"
```

Customize the WAS_HOME first line, <ldap_suffix> (o=itso in our example), to fit with your environment.

4. Change the UNIX permission bit to be able to execute this script with the following command:

```
chmod +x migrateEAR5MyApp.sh
```

5. Run the script with the following command:

```
./migrateEAR5MyApp.sh tamsecret SWIPE-C.ear
```

You should now see the following output:

```
BEGIN migrateEAR5MyApp.sh
Migrating SWIPE-C.ear...
AWXWS0021I  Logging all activity to the file .//pdwas_migrate.log.

AWXWS0025W   The pdwas-admin group already exists, and its members are
[wsadmin]
.
AWXWS0051E  The migrate tool has successfully completed.
END migrateEAR5MyApp.sh
```

You can check the pdwas_migrate.log file to make sure that no errors occurred.

Deploying the application with the administrative console

The deployment process stays the same when Tivoli Access Manager is integrated with WebSphere Application Server.

Important: Keep in mind that users' or groups' access to roles is now secured by Tivoli Access Manager ACLs. This means that the "Map security roles to users/groups" panel is irrelevant when deploying the application. All the other panels stay relevant.

The deployment of the SWIPE application is described in 7.5, "Deploying SWIPE" on page 147.

After deployment, manage roles-to-user mappings for this application. For this purpose, refer to the preceding sections: 14.3.1, "Create users or groups with Tivoli Access Manager" on page 460, 14.3.2, "Create and secure roles with Tivoli Access Manager" on page 463, and 14.3.3, "Grant user access to J2EE roles with Tivoli Access Manager" on page 466.

15

WebSphere administration and administrative security

This chapter describes subjects and tasks related to administrative security. We discuss the following topics:

- Securing the administrative tasks
- Securing Java Management Extensions (JMX) beans
- Authentication
- Role-based authorization for administration
- Fencing cells, nodes and servers
- Enabling global security
- Securing CosNaming services

15.1 WebSphere administration approaches

There are different ways to achieve administrative tasks in WebSphere Application Server V5:

- Command line or console command tools for performing simple functions like starting a server.
- Administrative console provides a browser-based “thin” GUI client interface for performing interactive administration functions.
- The *wsadmin* scripting tool is a command line alternative to the administrative console to configure and manage the WebSphere Application Server.
- JMX management API in WebSphere provides a programming interface for customized resource management.

In a base server environment, administrative security is configured at the server level. In a Network Deployment environment, administrative security is configured at the cell level, but some security features can be overridden at the server level.

15.2 Securing the administrative tasks

WebSphere Application Server V5 administrative tasks include a range of functions that an administrator can perform from z/OS and at the administrative console or using the wsadmin tool. This section focuses on securing some of the administrative tasks performed on z/OS.

15.2.1 Securing deployment manager and node agent

Deployment manager is a specialized application server that hosts the administrative console application and provides cell-level administration function in a network deployment configuration.

Node agent is an administrative process that provides node-level administration in a network deployment configuration. It routes administrative requests issued from the deployment manager to a particular application server. The node agent has access to the configuration repository of each server in the node.

Both deployment manager and node agent are started tasks for z/OS. Because it is a distributed management model, we need to ensure secure communications between the two.

The default protocol for communication between deployment manager and node agent is SOAP/HTTP. SOAP/HTTPS (SSL) should be used for transport layer

security. As mentioned previously, there are two types of SSL repertoires from which you must choose for WebSphere Application Server for z/OS: the System SSL (SSSL) repertoire, which refers to a RACF keyring and is used for the Web container, and the JSSE repertoire, which refers to a RACF keyring or an HFS file and is used for the JMX SOAP/HTTP connector. During the SSL setup, we need to ensure that the keystores and truststores of the deployment manager and node agents are set up to trust each other. Because the two SSL facilities need to handshake during the synchronization process in a Network Deployment environment, both the System SSL repertoire and the JSSE SSL repertoire must contain the trust basis, generally the public certificate of the signing CA, of the other.

15.2.2 Securing configuration files

Because both configuration data and applications are stored in HFS, correct permission bits need to be set for the directories and files to protect this data from unwanted access from other z/OS UNIX users or groups.

Base Application Server configuration

In a base Application Server configuration, the administrative console application resides in the same application server as other J2EE applications.

The permission bits of the WebSphere configuration files are set by default by WebSphere Application Server to *read/write* access for the owner and group identities of the WebSphere environment.

Given that an application deployed in the server runs with the servant region's identity, which is a member of the WebSphere group, it could be possible for an application to be deployed in the base Application Server with access to change the WebSphere HFS configuration files of the environment where it is deployed.

Tip: If you use the base Application Server configuration in your environment, we recommend that you use a unique user ID and group ID for each base Application Server installation. This will prevent a user from installing an application in a server and gaining access to modify a different server's HFS configuration files.

Network Deployment configuration

In a Network Deployment configuration, the administrative console application resides in the deployment manager server, isolated from additional J2EE applications deployed in the environment.

However, the identity of the deployment manager and the identity of the server controller and servant regions where the additional J2EE applications are deployed share the same WebSphere HFS group ownership.

There is a chance that an application deployed in a server in the WebSphere cell could have access to change WebSphere cell configuration files in the HFS if no additional security measures are taken.

There is also the possibility that an application deployed in a cell could have access to change HFS configuration files of different WebSphere cells if the different cells are sharing the administrator user ID and group ID.

We recommend that you take this into consideration when setting up cells in your environment. We recommend that you assign a unique user ID and group ID to each cell in the environment. If this becomes unmanageable in a large environment where there are many cells, we recommend that you at least assign a different ID and group identity to test cells to prevent users from deploying applications in test environments that could potentially change WebSphere Production and User Acceptance Test (UAT) configuration files. Ideally, a secure environment requires a unique administrator ID and group ID per cell.

ACLs can also be used to further restrict HFS file access. For more information about ACLs, refer to 10.12, “HFS ACLs” on page 324.

Note: WebSphere, by default, installs HFS files using read/write access for the WebSphere Application Server group identity (permission bits 770). There is documentation in the *WebSphere Application Server Information Center* about how to change the default umask for the servant and controller regions in the JCL procedures; however, this is not recommended by IBM. In our internal testing, changing the umask in the JCL procedures did not give us a successful nor expected result.

15.3 Securing JMX and MBeans

Administration in WebSphere Application Server Version 5 uses the JMX core framework. This section describes the securities of JMX and MBeans after an introduction to JMX.

15.3.1 JMX architecture in WebSphere

Java Management Extensions (JMX) is the framework for managing application servers and applications. It defines a framework that allows hardware and software resources to be wrapped in Java and exposed in a distributed

environment. All WebSphere Application Server V5 administration is done through JMX operations.

There are two levels in the JMX framework: the instrumentation level and the agent level. Layers of the JMX architecture are shown in Figure 15-1.

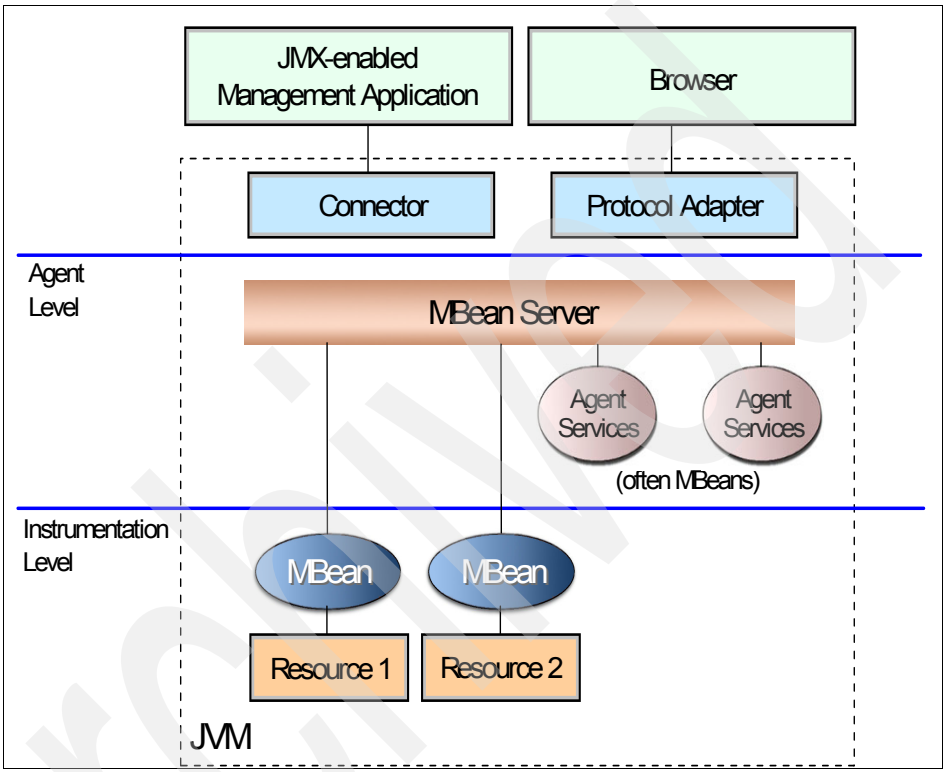


Figure 15-1 JMX architecture

The instrumentation level provides instrumentation implementation of JMX manageable resources. This is done by wrapping resources as Managed Beans (MBeans). A JMX manageable resource can be an application, an implementation of a service, a device, a user, and so forth. MBeans are Java objects that conform to certain design patterns and specific interfaces. MBeans can be standard or dynamic. The MBean has a simple activate/deactivate life cycle and it is activated/deactivated by calls through the MBean server.

The other level is the agent level, which provides management infrastructure. The agent level consists of MBean servers and agent services.The MBean server is a registry for MBeans. It is the component that provides the services for manipulating MBeans. Agent services are objects that can perform management operations on the MBeans registered in the MBean server. Agent services are

often MBeans as well, allowing them and their functionality to be controlled through the MBean server.

In addition to the two levels defined in the current JMX specification, JMX adaptors and connectors, although not mandatory in current implementations of the specification, are also implemented in WebSphere Application Server V5 to provide a communication channel between managed servers and enable client access to the WebSphere administrative service (AdminService). AdminService is a layer outside of the MBean server.

WebSphere provides many types of MBeans. An application server MBean might expose operations such as start and stop. An application MBean might expose operations such as install and uninstall. For WebSphere Application Server for z/OS Version 5 runtime, there are also control MBeans that reside in control region JVM, and servant MBeans that reside in servant region JVM.

15.3.2 JMX and MBean security

This section describes JMX and MBean security.

MBean authentication and authorization

MBeans live in every managed server. All MBeans are described by XML descriptors. These descriptors, along with the DTD files, are stored in a WebSphere JMX .jar file. This XML file is used to register these MBeans with WebSphere MBean Server. Security information, including authentication and authorization configuration, is stored in the deployment descriptor.

When security is enabled, invoking WebSphere-provided MBeans requires user ID and password or other types of credentials for authentication. User ID and password can be provided at wsadmin command line input or in a customized application. Security tokens, such as an LTPA token, are generated from logon for authentication when requests are cascaded to other servers in the domain.

There are four administrative roles defined for WebSphere Application Server: administrator, operator, configurator, and monitor. More detailed descriptions of these administrative roles are in 15.5, “Role-based administrative security” on page 480.

Each of the MBean functions, attributes, operation, and notification, of a WebSphere-supplied MBean is mapped to one of the administrative roles. When security is enabled, the role map for each MBean is set up at MBean activation time by calling the security component code. Every MBean request coming into the system is evaluated to check whether the authenticated user identity is mapped to an administrative role required in order to perform the operation

requested. If the authenticated user does not have the required or higher authority administrative role, the request is rejected.

Similar to EJB role mapping, the security roles for MBean methods are defined in the deployment descriptor XML file. The required role map is set up at MBean activation time and checked when requests arrive.

JMX connector security

Secure connections are needed between clients and servers, and deployment manager to node agents and application servers. On the z/OS platform, the connectors supported between the MBean server and clients are SOAP/HTTP and RMI/IIOP connectors. Authentication on the RMI/IIOP JMX connector is similar to authentication on an EJB IIOP invocation. The CSIv2 protocol and SAF registry are used. The SOAP/HTTPS JMX connector supports existing authentication methods provided by WebSphere Application Server, which includes basic login method, form-based login method, and client certificate mutual authentication.

To select the JMX connector protocol and security settings from administrative console, complete the following steps:

1. Select the server task and application server.
2. Click the server that needs to be configured.
3. Under Additional Properties, select **Administration Services**.
4. Under Preferred Connector, select either **SOAPConnector** or **RMIConnector**.
5. Under Additional Properties, select **JMX connectors**.
6. Click the connector type selected in step 4.
7. Under Additional Properties, select **Custom properties**.

If SOAPConnector is selected, configure the “request timeout” and sslConfigs values. sslConfig values can be DefaultSSLSettings, RACFJSSESettings, or DefaultSOAPSSLSettings.

Java 2 security

When Java 2 security is enabled for WebSphere Application Server, in addition to providing user ID and password or other credentials for authentication, proper Java 2 security permission has to be set up for MBean code for custom administrative applications or applications with the intention to access WebSphere Application Server administrative and JMX classes and methods. Security exceptions are thrown when you attempt to invoke methods of these tightly protected classes without granted permission. If your MBeans are part of

your application, you can set the permission in the was.policy file that you supply as part of your application metadata. If the MBeans are part of shared libraries among several applications, depending on the installation, you can set permission in the library.policy file in node configuration, or server.policy in the properties directory, for example, `permission com.tivoli.jmx.MBeanServerPermission "MBeanServer.*"`

15.4 WebSphere administration authentication

When global security is enabled, authentication happens whenever an administrative client connects to a WebSphere process, whether the administrative tasks are performed through the administrative console, the wsadmin scripting tool, or a customized program. The user ID and password or some type of credential information is required.

The authentication approach is the same as authenticating a WebSphere application, with SWAM, LTPA, or ICSF as the authentication mechanism and a local SAF registry, LDAP, or a custom registry as the active user registry. The protocol authentication on the JMX connector that a client uses is described in 15.3.2, “JMX and MBean security” on page 478.

15.5 Role-based administrative security

IBM WebSphere Application Server for z/OS V5 allows much finer granularity of access control than previous versions.

15.5.1 Four administrative roles

When global security is enabled, the user authenticated to perform administrative actions is defined as one of the four administrative roles. These four roles are:

- Monitor: Least privileged; basically allows a user to view the WebSphere Application Server configuration and current state.
- Configurator: Monitor privilege plus the ability to change the WebSphere Application Server configuration.
- Operator: Monitor privilege plus the ability to change runtime states, such as starting or stopping services.
- Administrator: Operator plus configuration privilege and the permission required to access sensitive data, including server password, LTPA password and keys, and so on.

The relationships between the four administrative roles are shown in Figure 15-2.

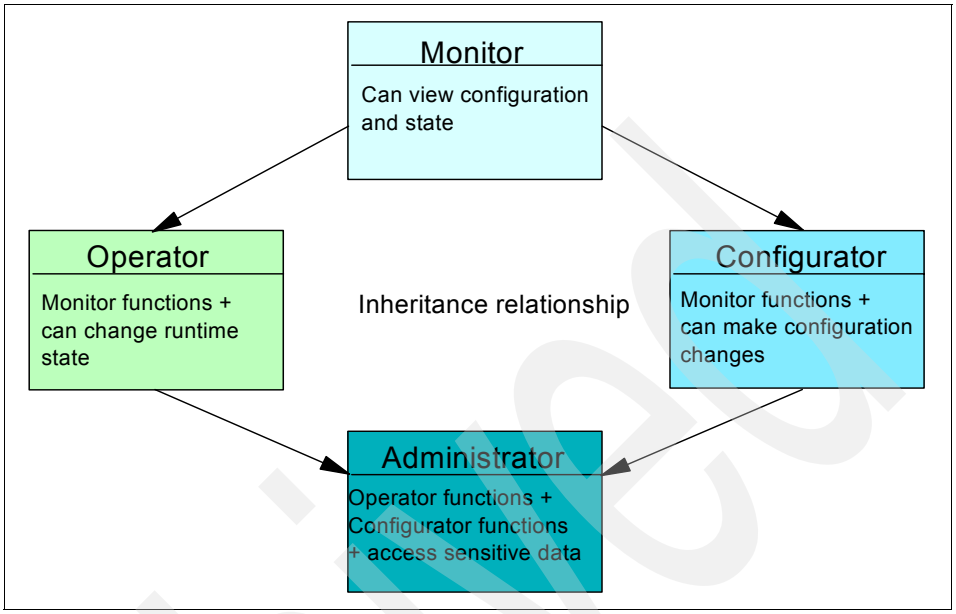


Figure 15-2 Administrative security roles inheritance relationships

When WebSphere Application Server global security is enabled, the administrative subsystem role-based access control is enforced.The administrative subsystem includes security server, user registry, and all the Java Management Extensions (JMX) MBeans. When security is enabled, both the Web-based administrative console and the administrative scripting tool require users to provide the required authentication data, which is then used for authorization role checking. If the user does not have appropriate privileges, the request is rejected.

Moreover, the administrative console is designed such that the control functions that are displayed on the GUI pages are adjusted according to the security roles a user has. For example, a user who has only the monitor role can only see non-sensitive configuration data. A user with the operator role would have options available on GUI pages to change the system state. Notice the difference in the administrative consoles in Figure 15-3 on page 482 and Figure 15-4 on page 483. The first is the console of an administrator, the second is the console of a monitor.

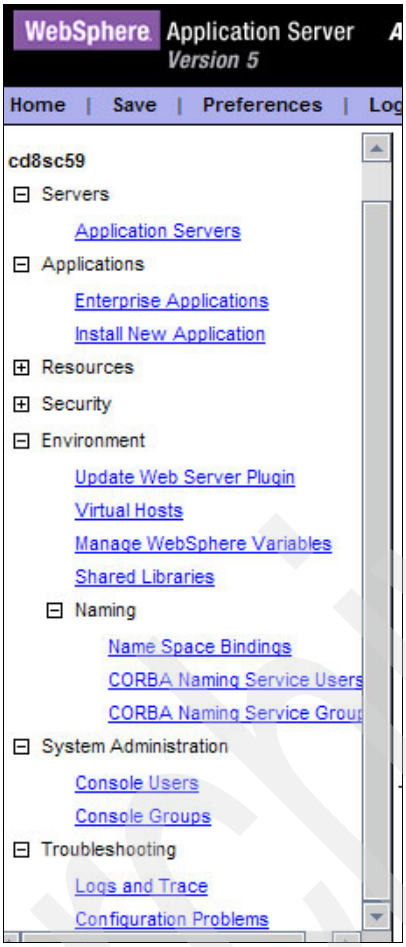


Figure 15-3 Console of administrator

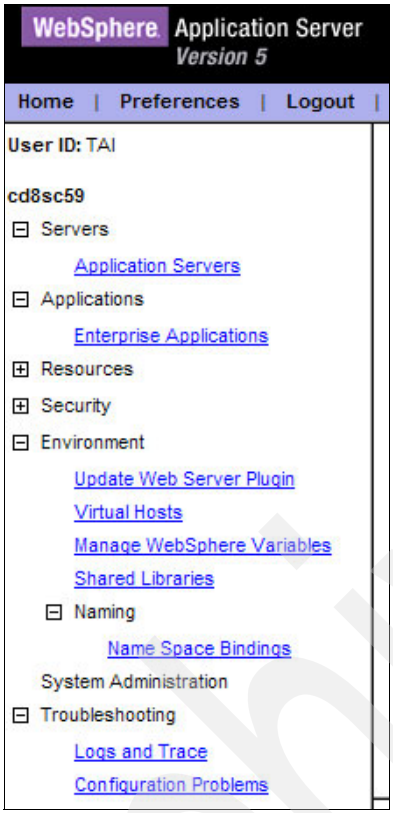


Figure 15-4 Console of monitor

15.5.2 Map a user to an administrative role

In order for a user or a group of users defined in the user registry to perform administrative actions with global security enabled, the user or the group needs to be mapped to one of the four administrative roles.

- The authorization engine in WebSphere Application Server for z/OS Version 5 is pluggable. There are two approaches for authorization role mapping. One uses XML metadata, which is the same as WebSphere Application Server Version 5 on other platforms. The other approach uses SAF authorization, which checks the RACF EJBROLE class profile. The choice between the two applies to all authorizations in the WebSphere management domain. The choice is made on the administrative console. To reset the choice from the administrative console, complete the following steps:

  a. Select **Security** → **User Registries** → **Local OS**.

b. Select **Custom properties** and set the property com.ibm.use.SAF.Authorization to `true` or `false` for using SAF authorization or non-SAF authorization.

Map a user to an administrative role when using SAF authorization

There are four profiles defined in the RACF EJBROLE class for administrative authorization. They are administrator, configurator, monitor, and operator. When using RACF for role mapping, you need to define RACF user ID or group READ access for one of these profiles, depending on the administrative authority you give to the user.

Map a user to an administrative role when using non-SAF authorization

Note: Role-based authorization for administrative tasks with a remote registry (CUR or LDAP) configured is supported only when you are on WebSphere level 5.02 or higher. If you use an earlier version, you need to synchronize the WebSphere administrator user IDs between the remote registry and the local registry, because authorization always used the SAF EJBROLE implementation.

To map a user to an administrative role, complete the following steps:

1. Select **System Administration** from administrative console.
2. Select **Console User** to see the current console users and their administrative roles.
3. Click **Add**, type in a user ID that exists in the active user registry, and select one of the four roles for the user.
4. Click **Apply**, **Save**, and so on.
5. If security is not enabled before the user-role mapping, the server or deployment manager needs to be restarted with global security enabled.

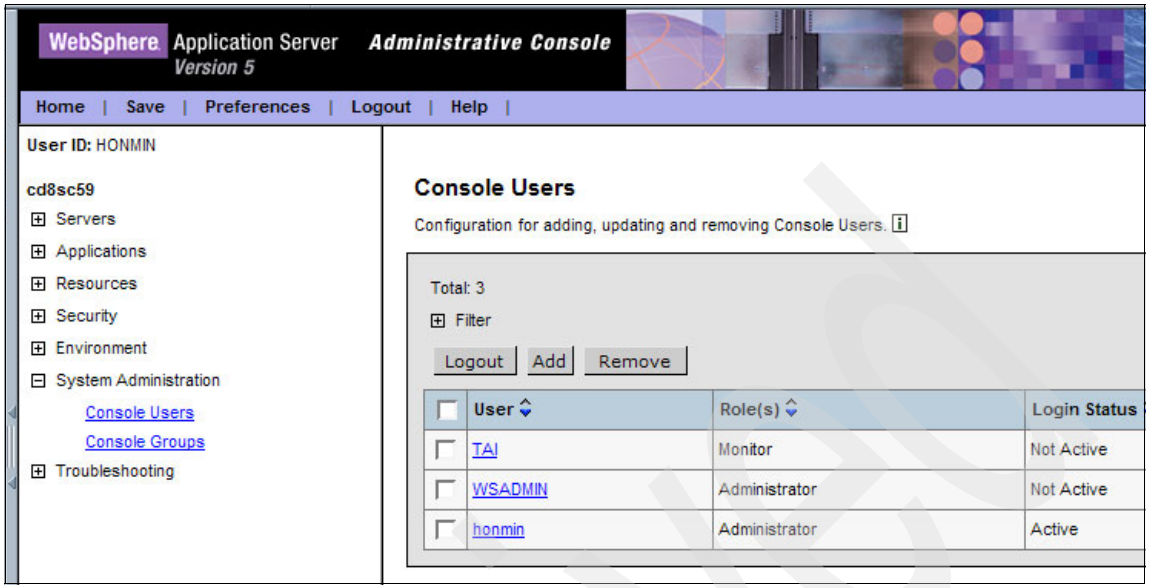


Figure 15-5 Map a user to an administrative role

To map a group to an administrative role, complete the following steps:

1. Select **System Administration** from administrative console.
2. Select **Console Group** to see the current console groups and their administrative roles.
3. Click **Add**, type in the group name that exists in the configured active user registry, or select the special subject **ALL_AUTHENTICATED** or **EVERYONE**. A special subject is a generalized group of a particular class of users. Select one of the four roles for the group.
4. Click **Apply**, **Save**, and so on.

To associate or remove a user or group from the administrative roles, click **Remove** instead of **Add** in step 3.

The changes made for the administrative roles are reflected in admin_authz.xml files under the cell configuration directory. In our example, for mount point /WebSphere/BS08, a base server configuration for server B08 with long cell name cd8sc59, the XML file is /WebSphere/BS08/appserver/config/cells/cd8sc59/admin-authz.xml. In the Network Deployment configuration example, for mount point /WebSphere/TEST, a cell TEST with long cellname *ctop*, the XML file is /WebSphere/TEST/DeplMgr/config/cells/ctop/ admin-authz.xml.

The sample admin_authz.xml file for Figure 15-5 on page 485 is shown in Example 15-1.

Example 15-1 Sample admin_authz.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:rolebasedauthz="http://www.ibm.com/websphere/appserver/schemas/5.0/roleba
sedauthz.xmi">
  <xmi:Documentation>
    <contact>{Your Contact Info}</contact>
  </xmi:Documentation>
  <rolebasedauthz:AuthorizationTableExt xmi:id="AuthorizationTableExt_1"
context="domain">
    <authorizations xmi:id="RoleAssignmentExt_1" role="SecurityRoleExt_1">
      <users xmi:id="UserExt_1" name="WSADMIN"/>
      <users xmi:id="UserExt_3" name="honmin"/>
      <groups xmi:id="GroupExt_1" name="WASADM"/>
      <specialSubjects xmi:type="rolebasedauthz:ServerExt"
xmi:id="ServerExt_1"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignmentExt_2" role="SecurityRoleExt_2"/>
    <authorizations xmi:id="RoleAssignmentExt_3" role="SecurityRoleExt_3"/>
    <authorizations xmi:id="RoleAssignmentExt_4" role="SecurityRoleExt_4">
      <users xmi:id="UserExt_2" name="TAI"/>
    </authorizations>
    <roles xmi:id="SecurityRoleExt_1" roleName="administrator"/>
    <roles xmi:id="SecurityRoleExt_2" roleName="operator"/>
    <roles xmi:id="SecurityRoleExt_3" roleName="configurator"/>
    <roles xmi:id="SecurityRoleExt_4" roleName="monitor"/>
  </rolebasedauthz:AuthorizationTableExt>
</xmi:XMI>
```

The administrative console application .ear file also has the authorization roles stored in its Web application deployment descriptor.

When global security is enabled, a WebSphere Application Server server runs under the server identity that is defined under the active user registry configuration. This is why a server special subject is mapped to the administrator role, as shown in Example 15-1:

```
<specialSubjects xmi:type="rolebasedauthz:ServerExt" xmi:id="ServerExt_1"/>
```

The server special subject is not shown in the administrative console or any other administrative tools.

In WebSphere Application Server for z/OS Version 5 using Local OS, the server identity is the Java representation of the started task ID of each process (address

space) established in the RACF STARTED class profiles. These processes include deployment manager, node agent, controller, servant, and so on. In addition to relying on special subject of server, all WebSphere started task IDs are connected to a configuration group defined in SAF, and this group is added to various xml authorization files during the customization process. In Example 15-1 on page 486, group WASDMIN, which is assigned to the administrator role, is the RACF group that all WebSphere started task IDs connect to.

15.6 Fencing servers, nodes, and cells in a sysplex

Sometimes, a user or group with a certain administrative authority to one cell or server might not have access or enough administrative authority to other cells or servers in their environment. For example, a user with administrator authority role on a test server might not have any administrative access to the production server. When several cells or servers coexist in a Parallel Sysplex or even the same LPAR, there are a couple of ways to set up administrative fencing.

15.6.1 Fencing base servers

In a WebSphere base server environment, one way to fence servers is to use different authentication user registries for different servers. For example, use LDAP for one server and a custom registry for another server.

Because administrative access control is through role-based authorization, the other approach is to assign different administrative roles to different server administrative users. For example:

- Use XML metadata admin_authz.xml to map authorization roles for all servers; different servers have different XML files.
- Use SAF authorization for one server, that is, use RACF EJBROLE class profiles, and set up another server with XML metadata for administrative role mapping.
- Use a security domain prefix to define RACF EJBROLE class profiles (administrator, configurator, monitor, and operator). SAF security domains make fencing with SAF authorization possible. The concept was introduced to the code with WebSphere Application Server Version 5.02. See “Security domains” on page 354 for more information about security domains.

15.6.2 Fencing cells

Authentication and authorization setup is at the cell level. Fencing cells can use the suggestions described in 15.6.1, “Fencing base servers” on page 487.

- Use different authentication user registries for different cells.
- Use XML metadata for administrative role mapping for all cells; different cells have different XML metadata.
- Use SAF authorization for one cell and XML metadata authorization for the other cells.
- Through the use of a RACF EJBROLE class profile name prefix, a shared RACF can be used for multiple cells as the active user registry and authorization mechanism. Fencing can be set up by using different profile names for different cells.

15.6.3 Fencing nodes and servers in a cell

Authentication and XML metadata-based authorization is configured at the cell level. Fencing nodes and servers within a cell is, therefore, not possible. If fencing nodes and servers is required, the approach that can be used is to separate nodes and servers into different cells. Group nodes and servers requiring the same administrative user or group access into one cell. Use the cell fencing suggestions described earlier for protection.

15.6.4 Security concerns of the federation process

When adding a base Application Server node to a cell, if the cell has security enabled, the added server inherits the user registry and authentication mechanism from the cell. If the server has security enabled with a different user registry and authentication mechanism than the cell, some application changes might be needed.

Another issue is to ensure that addNode can communicate as an SSL client to the deployment manager. This requires that the addNode truststore (configured in sas.client.props) contains the signer certificate of the deployment manager personal certificate as found in the keystore (specified in the administrative console).

For a more detailed discussion about these concerns, refer to the *WebSphere Application Server for z/OS Information Center*.

15.7 Enabling global security

In this section, we discuss global security and how to enable it.

15.7.1 Global security

The term global security implies a security configuration that is effective for the entire security domain. It can be thought of as a “big switch” that activates a wide variety of WebSphere security settings. Values for these settings can be specified, but they will not take effect until global security is activated. The settings include the authentication of users, the use of SSL, the choice of user registry and Java 2 security. In particular, application security, including authentication and role-based authorization, is not enforced unless global security is active.

You can disable security on individual application servers while global security is enabled. However, you cannot enable security on an individual application server while global security is disabled.

Global security is disabled by default, in order to simplify the installation of the server. But after you have built a server and installed the administrative console, you will find that any user can log on to the administrative console and that no password is required. Global security is necessary to secure the administrative console. But proper planning is required, because incorrectly enabling global security can lock you out of the administrative console, or cause the server to abend.

15.7.2 Why turn on global security?

Turning on global security activates the settings that protect your server from unauthorized users. There might be some environments where no security is needed, for example, a development system. On these systems you can elect not to enable global security. But in most environments, you will want to keep unauthorized users from accessing the administrative console as well as your business applications. Global security must be enabled in order to do this.

15.7.3 What global security protects

The settings that are activated when global security is enabled include:

- Authentication of HTTP clients
- Authentication of IIOP clients
- Administrative console security

- The use of SSL transports
- Role-based authorization checks of servlets, EJBs, and MBeans
- The propagation of identities (RunAs)
- CBIND checks

If cell security is enabled, but security for individual servers is disabled, J2EE applications are not authenticated or authorized. However, naming and administrative security is still enforced. Consequently, because Naming Services can be called from user applications, you will need to grant “Everyone” access to the Naming functions required so they will accept unauthenticated requests. User code does not directly access administrative security except through the supported scripting tools.

15.7.4 Enabling global security on a base Application Server node

Global security activates a number of WebSphere security settings. You might not understand all of these settings or know what value they should be set to. Fortunately, most of the settings receive their default value from the installation scripts, run during server installation. The following is a checklist for enabling global security on a base Application Server node:

1. Ensure that you are running at service level W500101 or later.
2. Ensure that the installation tasks were run, including the security tasks. See 10.10.2, “ISPF customization dialogs for RACF command generation” on page 313 for details.
   a. In the ISPF dialog, define variables for security customization.
   b. In the ISPF dialog, run “Generate customization jobs”.
   c. Run the BBOCBRAJ job that generates the corresponding RACF commands.
   d. Run the BBOCBRAK job that executes the RACF commands created by the installation scripts.

   This will build the keyrings and certificates and a bunch of other RACF stuff.
3. Start the server if it is not already up.
4. Go to the administrative console (use Microsoft Internet Explorer, it seems to work better). Sign in using any user ID, no password needed.
5. Click **Security** → **Authentication Mechanisms** → **LTPA**. Fill in a password and confirm it by entering it again. Click **Apply**, **Save**, and so on.

6. Click **Security** → **User Registries** → **Local OS**. On the Local OS User Registry page, click **Custom Properties**. If you want WebSphere to check RACF EJBROLE profiles for determining whether a user has a role, select com.ibm.security.SAF.authorization and com.ibm.security.SAF.delegation and set them to true. Otherwise, leave them set to false. If you change them, click **Apply**, **Save**, and so on.
7. If you chose to use EJBROLE profiles, now is a good time to use RACF to PERMIT your administrative user IDs to the EJBROLE class profile "administrator". If you chose not to use EJBROLE profiles, you should click **System Administration** → **Console Users**, and add your user IDs as administrators. Click **Apply**, **Save**, and so on.
8. Click **Security** → **User Registries** → **Local OS** → **OK**. This will take you to the Global Security page.
9. On the Global Security page, scroll to the bottom and click **Custom Properties**. On the Custom Properties page, click **EnableTrustedApplications** and set its value to true. Click **Apply**, **Save**, and so on.
10. Click **Security** → **Global Security**. Select the option that says **Enabled**, and then clear the box that says Enforce Java 2 Security. The Active Protocol should be CSI and SAS. The Active Authentication Mechanism should be LTPA. The Active User Registry should be Local OS. Click **Apply**, **Save**, and so on.
11. Now you can recycle the server and connect to the administrative console using your browser. The server should successfully redirect you to the SSL port, where you will get the usual certificate warnings. Then you should see the login page where you can enter the user ID and password of a valid administrator.

15.7.5 Disabling global security

If global security is working, you can log on to the administrative console and select **Security** → **Global Security**. Then clear the check box labelled Enabled. Restart the server and global security will be off.

If global security isn't working properly, it can cause the server to not come up, or it will come up but you won't be able to log on. There are two ways to disable global security.

One way is to use the wasdmin scripting tool:

1. From the bin directory of the Application Server, issue the following command:

```
wsadmin -conntype NONE
```

2. At the command-line prompt for wsadmin, issue the **wsadmin** command:

   ```
   securityoff
   ```

3. Restart the server. Global security is now disabled.

The wsadmin scripting tool is described in more detail in the next section.

15.7.6 Using wsadmin scripting to enable global security

WebSphere Application Server for z/OS Version 5 provides a new administration scripting engine that can be used to manage and deploy applications. Scripts are executed under the new tool wsadmin.

Overview

The wsadmin client uses the BeanScripting Framework (BSF) and is based on JMX.

The JACL (Java Command Language) that is used in scripts allows you to create procedures. Procedures can be grouped into profile files that can be passed to wsadmin in a command line to create a custom environment for wsadmin execution. These procedures can then be used as normal JACL commands. For more information about how to create profiles, refer to the *WebSphere Application Server Information Center* Web site.

Setting global security

When you run wsadmin in an environment where global security is enabled, you need to supply authentication information in order to communicate with the server. The user's name and password can be specified either in the command line arguments for wsadmin or in the sas.client.props file. Changes introduced into the properties file will depend on whether the RMI or SOAP connector is used to communicate with the server. Remember that if you specify the user's name and password in the command line, it will override this information in the properties file.

Invoking wsadmin through command line

To invoke wsadmin, issue the following command from the <WAS_Home>/bin directory:

```
./wsadmin.sh -user <username> -password <password>
```

Example 15-2 Example output from successful invocation of the wsadmin tool

```
/WebSphere/V5ROMO/AppServer/bin>./wsadmin.sh -user wsadmin -password wsadmin
WASX7209I: Connected to process "wd3nd3cd3sc59" on node nd3cd3sc59 using SOAP
connector; The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"
wsadmin>
```

Editing the property file

The property file updates required for running in secure mode will depend on whether a Remote Method invocation (RMI) or Simple Object Access Protocol (SOAP) connector is being used to connect. If the RMI connector is being used, set the following properties in the sas.client.props file with the appropriate values:

com.ibm.CORBA.loginUserid=

com.ibm.CORBA.loginPassword=

com.ibm.CORBA.loginSource=properties

If a SOAP connector is being used, set the following properties in the soap.client.props file with the appropriate values:

com.ibm.SOAP.securityEnabled=true

com.ibm.SOAP.loginUserid=

com.ibm.SOAP.loginPassword=

The wsadmin tool makes security configuration easier. There are two security profiles that set up procedures that can enable and disable security. There are four procedures:

- securityon: This turns global security on using LocalOS security.
- securityoff: This turns global security off.
- LTPA_LDAPSecurityOn: This turns LTPA/LDAP global security on using the LDAP registry.
- LTPA_LDAPSecurityOff: This turns LTPA/LDAP global security off.

For the correct syntax for the securityon and LTPA_LDAPSecurityOn procedures issue the wsadmin help command:

```
wsadmin > securityon help
Syntax: securityon user password
```

JACL language and scripts

This section gives a few examples of scripts that can be used to configure security easier. These sample scripts are based on and tested with base Application Server, not Network Deployment.

Setting global security

The script shown in Example 15-3can enable or disable global security based on the input parameter that is given.

Example 15-3 Global security JACL file

```
#Enabling Global Security
#set the security object
set security_item [$AdminConfig list Security ]
#Here we are setting the parameter from the input parameter that was passed in
set argv0 [lindex $argv 0]
#creating and initializing the value parameter
set value null
#Set the value based on the input parameter
if {[regexp $argv0 enable]} { set value true }
if {[regexp $argv0 disable]} { set value false }
if {[regexp $value null]} {
 puts "Wrong parameter, use enable|disable"
 return
}
#modifying the attribute in the configuration
$AdminConfig modify $security_item [list [list enabled $value]]
#Save the changes from the script.
$AdminConfig save
```

To invoke the script in Example 15-3, issue the following command:

```
<WAS_Home>/bin>./wsadmin.sh -user <username> -password <password> -f
GlobalSecurity.jacl enable
```

Changing authentication mechanisms

The following script changes the authentication mechanism to LTPA.

Example 15-4 Authentication mechanism authentication.jacl file

```
#change authentication mechanisms
#for the security object
set user_authName "LTPA"
#get the security object
set security_item [$AdminConfig list Security ]
#list all the authentication mechanisms defined
set auth_mechs [$AdminConfig list AuthMechanism ]
#find the one that starts with the name we set at the beginning of the script
foreach auth_mech $auth_mechs {if {[regexp $user_authName $auth_mech ] } {
 set new_auth_mechs $auth_mech;break }}
#modify the Authentication mechanism attribute for the security object
$AdminConfig modify $security_item [list [list activeAuthMechanism
$new_auth_mechs ]]
#saving the configuration
$AdminConfig save
```

To invoke the script in Example 15-4 on page 494, issue the following command:

```
<WAS_Home>/bin>./wsadmin.sh -user <username> -password <password> -f
authentication.jacl
```

Changing the user registry

The following script changes the user registry based on the input parameter that is given.

Example 15-5 Registry change to registry.jacl file

```
#change the user registry
#for the security object
#get the security object
set security_item [$AdminConfig list Security ]
#Here we are setting the parameter from the input parameter that was passed in
set argv0 [lindex $argv 0]
#creating and initializing the value parameter
set value null
#Set the value based on the input parameter
if {[regexp $argv0 CUR]} { set value CUR }
if {[regexp $argv0 LDAP]} { set value LDAP }
if {[regexp $argv0 Local]} {set value Local }
if {[regexp $value null]} {
 puts "Wrong parameter, use CUR|LDAP|Local"
 return
}
#list all the user registries defined
set user_regs [$AdminConfig list UserRegistry ]
#find the one that starts with the name we set at the beginning of the script
foreach user_reg $user_regs {if {[regexp $value $user_reg ] } {
 set new_user_reg $user_reg;break }}
#modify the user registry attribute for the security object
$AdminConfig modify $security_item [list [list activeUserRegistry $new_user_reg
]]
#saving the configuration
$AdminConfig save
```

To invoke the script in Example 15-5, issue the following command:

```
<WAS_Home>/bin>./wsadmin.sh -user <username> -password <password> -f
registry.jacl CUR
```

15.7.7 Disable server-level application security in a cell

In a Network Deployment environment, when cell security is enabled, user security for individual servers can still be disabled. When this happens, J2EE

applications are not authenticated or authorization checked. However, naming and administrative security are still enforced.

To disable server security in a cell when global security is enabled, complete the following steps:

1. Go to **Servers**, select **Application Servers**, and then select <**server name**>.
2. Under Additional Properties, select **Server security**.
3. Go to the Server Level Security panel, clear the **Enabled** flag and click **OK** or **Apply**.

Some security attributes set at global security level can also be overridden at the server level. The Server Level Security panel lists attributes that are on the Global Security panel and can be overridden at the Server Level. These attributes include Java 2 Security Manager, CSIv2, SAS, and so on. You can also reset the attribute to the same setting of global security by clicking selections such as **Use Cell Security**, **Use Cell CSI**, and **Use Cell SAS**.

15.8 Securing the CosNaming service

CosNaming services provide access to the content of the WebSphere Application Server name space. CosNaming in WebSphere Application Server for z/OS Version 5 uses a distributed model. There are multiple CosNaming servers. Deployment manager, daemons, application servers all provide CosNaming services. Objects are bound into contexts within the same process. Typically, lookups start in local processes and end in processes where the target object is. There are generally two ways in which client programs result in CosNaming calls: through the JNDI interfaces, and with CORBA clients invoking CosNaming methods directly.

15.8.1 Associate users and groups to CosNaming roles

WebSphere Application Server for z/OS Version 5 provides finer granularity access control on CosNaming services. There are four predefined security roles with authority levels from low to high:

- CosNamingRead: Users can query the WebSphere Application Server name space, using, for example, the JNDI lookup method. The special subject Everyone is the default policy for this role.
- CosNamingWrite: Users can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. The special subject AllAuthenticated is the default policy for this role.

- CosNamingCreate: Users can create new objects in the name space through such operations as JNDI createSubcontext and CosNamingWrite. The special subject AllAuthenticated is the default policy for this role.
- CosNamingDelete: Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special subject AllAuthenticated is the default policy for this role.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment result in an org.omg.CORBA.NO_PERMISSION exception from the CosNaming Server.

CosNaming security roles can be configured through the administrative console or the wsadmin scripting tool. To assign a user to a cosNaming role, complete the following steps:

1. Select **Environment task** → **Naming**.
2. Select **CORBA Naming Services Users** or **CORBA Naming Services Users** to see the list of users or groups currently assigned with CosNaming roles.
3. Click **Add**.
4. Type in the user ID or Group Name currently defined in the user registry, and select one or more CosNaming roles.
5. Click **Apply**, **Save**, and so on.
6. Restart the server for the changes to take effect.

The changes made for the CosNaming roles are reflected in the naming_authz.xml files under the cell configuration directory. In our example, for a base server configuration for server B08 with the long cell name cd8sc59, the XML file is /WebSphere/BS08/appserver/config/cells/cd8sc59/naming-authz.xml. In the Network Deployment configuration example, for a cell TEST with the long cell name ctop, the XML file is /WebSphere/TEST/DeplMgr/config/cells/ctop/naming-authz.xml.

Example 15-6 Example naming_authz.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:rolebasedauthz="http://www.ibm.com/websphere/appserver/schemas/5.0/roleba
sedauthz.xmi">
  <xmi:Documentation>
    <contact>WebSphere Security</contact>
  </xmi:Documentation>
  <rolebasedauthz:AuthorizationTableExt xmi:id="AuthorizationTableExt_1"
context="domain">
```

```
    <authorizations xmi:id="RoleAssignmentExt_1" role="SecurityRoleExt_1">
      <users xmi:id="UserExt_1" name="JOE"/>
      <specialSubjects xmi:type="rolebasedauthz:EveryoneExt"
xmi:id="EveryoneExt_1"/>
      <specialSubjects xmi:type="rolebasedauthz:ServerExt"
xmi:id="ServerExt_5"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignmentExt_2" role="SecurityRoleExt_2">
      <users xmi:id="UserExt_2" name="JOE"/>
      <specialSubjects xmi:type="rolebasedauthz:AllAuthenticatedUsersExt"
xmi:id="AllAuthenticatedUsersExt_2"/>
      <specialSubjects xmi:type="rolebasedauthz:ServerExt"
xmi:id="ServerExt_6"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignmentExt_3" role="SecurityRoleExt_3">
      <users xmi:id="UserExt_3" name="JOE"/>
      <specialSubjects xmi:type="rolebasedauthz:AllAuthenticatedUsersExt"
xmi:id="AllAuthenticatedUsersExt_3"/>
      <specialSubjects xmi:type="rolebasedauthz:ServerExt"
xmi:id="ServerExt_7"/>
    </authorizations>
    <authorizations xmi:id="RoleAssignmentExt_4" role="SecurityRoleExt_4">
      <users xmi:id="UserExt_4" name="JOE"/>
      <specialSubjects xmi:type="rolebasedauthz:AllAuthenticatedUsersExt"
xmi:id="AllAuthenticatedUsersExt_4"/>
      <specialSubjects xmi:type="rolebasedauthz:ServerExt"
xmi:id="ServerExt_8"/>
    </authorizations>
    <roles xmi:id="SecurityRoleExt_1" roleName="CosNamingRead"/>
    <roles xmi:id="SecurityRoleExt_2" roleName="CosNamingWrite"/>
    <roles xmi:id="SecurityRoleExt_3" roleName="CosNamingCreate"/>
    <roles xmi:id="SecurityRoleExt_4" roleName="CosNamingDelete"/>
  </rolebasedauthz:AuthorizationTableExt>
</xmi:XMI>
```

If a user is assigned a particular naming role and that user is a member of a group that has been assigned a different naming role, the user will be granted the most permissive access between the role he was assigned and the role his group was assigned.

15.8.2 SAF EJBROLE authorization for naming

SAF EJBROLE authorization is also supported for naming service access control. Four profiles are defined for the RACF EJBROLE class: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. RACF user IDs and groups need to be granted READ access to these profiles to gain correct naming service authorization.

Note: At the time of writing, CosNaming Authorization only honors special subjects AllAuthenticated and Everyone Access when the configured active use registry is *not* a LocalOS registry. This issue has been addressed in APAR II13596 and a fix is expected at a later time.

If SAF security domains are implemented, the RACF EJBROLE class profiles for the naming service authorization should be preceded by the security domain prefix.

As illustrated in Figure 12-6 on page 355, if a cell qualified domain is implemented with domain name “CELL1”, the following four profiles should be defined for the RACF EJBROLE class:

- CELL1.CosNamingRead
- CELL1.CosNamingWrite
- CELL1.CosNamingCreate
- CELL1.CosNamingDelete

See “Security domains” on page 354 for more information about security domains.

16

Web container security

In the J2EE application architecture, the Web module of the enterprise application comprises one or more related servlets, Java Server Pages (JSP files), XML and HTML files that can be managed as one integrated unit. The files in the Web module provide the presentation and control logic to perform one or more business functions.

The Web modules of the enterprise application run in the Web container of the application server. The Web container, as a runtime environment for the Web application, is responsible for handling requests for servlets, JSP files, and other components running on the server side. The Web container creates servlet instances, loads and unloads servlets, creates and manages requests and response objects, and performs other servlet management tasks.

This section describes the process and tools of WebSphere Application Server to configure security for the Web module of enterprise application.

16.1 Introduction

When a security policy is specified for a Web resource and IBM WebSphere Application Server global security is enabled, the Web container performs access control when the resource is requested by a Web client. The Web container asks the Web client for authentication data, if none is present, according to the authentication method specified in the deployment descriptor. The Web container ensures that the data constraints are met and determines whether the authenticated user has access to the security role associated with the Web component being accessed.

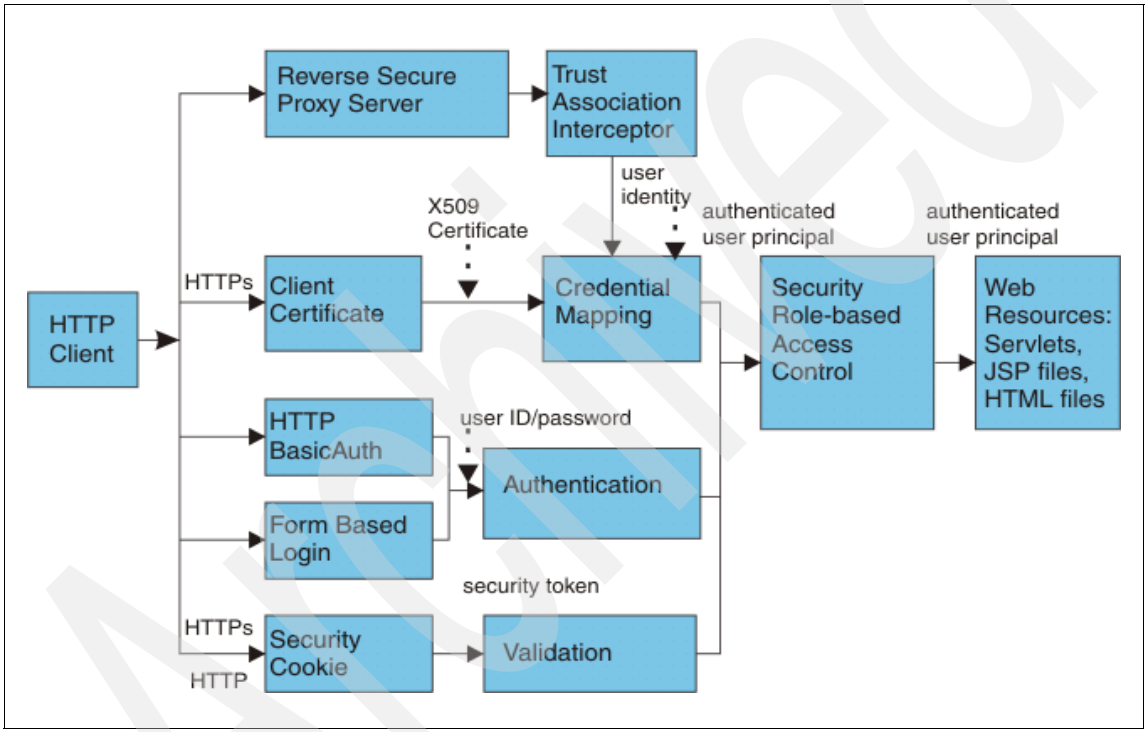


Figure 16-1 Web container security overview

When either the LTPA or ICSF authentication mechanism is configured, and single sign-on (SSO) is enabled, a security cookie is issued to an authenticated client, which can represent the user within the specified security domain. You can use Secure Sockets Layer (SSL) to protect the security cookie from being intercepted and from being replayed. When a trust association is configured, WebSphere can map an already-authenticated user identity to security credentials based on the trust relationship established with the authentication server, which is usually a secure reverse proxy server.

The Web security collaborator component of WebSphere Application Server enforces role-based access control by using an authorization module, which makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested servlet or JSP file if it has access to the security role associated with those Web components in the authorization constraint (within the security constraint within the deployment descriptor).

Within applications, servlets and JSP files can use the HttpServletRequest methods, getUserPrincipal, and isUserInRole to find the current security principal and check that the principal has access to a particular role.

When a servlet file accesses EJB methods, either the caller identity or a run-as identity is propagated to the EJB container, depending on the RunAs configuration in the deployment descriptor.

This security implementation is illustrated in summary in Figure 16-2.

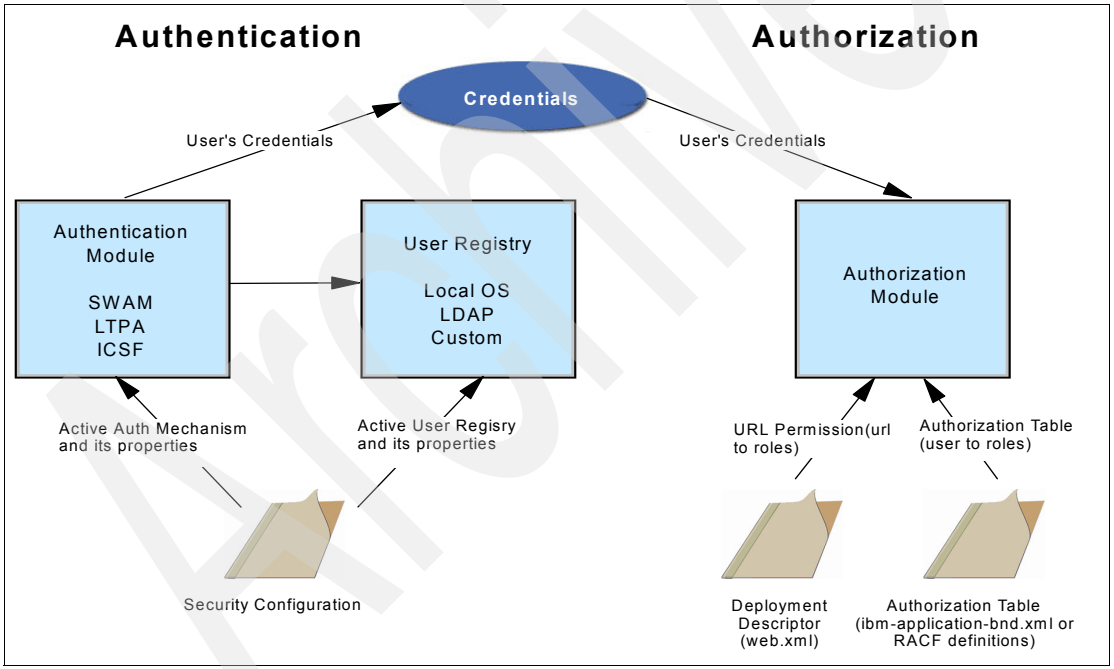


Figure 16-2 Web container WebSphere Application Server security overview

Note: The term “authentication mechanism” can be found in both Java Servlet Specification Version 2.3 and WebSphere literature, but they do not refer to the same concept.

“Authentication mechanism”, as referred to by the Java Servlet Specification Version 2.3, relates to how the user is challenged and how a credential is forwarded to the application server. It will be referred to as *authentication methods* or *login facilities*.

Authentication mechanism, as it appears in WebSphere manuals and product menus, relates to the WebSphere internal modules that handle a user authentication after WebSphere has received it from the network. It is referred to as *WebSphere authentication mechanism*.

16.2 Web container authentication

The authentication method defines how the Web container authenticates the user. Before any authorization constraint is applied, the user must pass the authentication process using a configured mechanism. The possible options are:

- Basic authentication

  The Web server sends a request to the client, containing the realm name in which the user is authenticated. The browser prompts the user for a user ID and password, which are encoded and included in every HTTP request.

- Form-based authentication

  The application programmer provides a customized HTML form in which the user enters authentication data. The values that the end user supplies in the form are transmitted in clear text as parameter values in the first HTTP request. After the server has authenticated the user, only an encrypted token is flowing between the server and the browser. This encrypted token does not contain the user password.

- Digest authentication

  The password is not sent over the communications channel during the authentication exchange. Authentication relies on checksums matching. WebSphere Application Server does not support digest authentication (on any platform).

- Client certificate authentication

  The client is challenged for a valid and approved x.509 certificate in order to be able to start a communication with the server. After the server accepts the certificate, an encrypted channel is created between the client and the server. Moreover, the Web server can extract the credentials from the certificate and use them to perform a mapping with the users in the current registry.

16.2.1 Configure authentication for Web components

Web application authentication is selected in the web.xml file.

The login mechanism is defined adding a login-config XML tag that contains the following:

- Authentication method
- Realm name
- Name of HTML pages related to the form-based login (optional)

Note: In order to be challenged by the login mechanism, you must have your resource protected by a security constraint as described in 16.3.1, “Web resource protection” on page 542. If the Web application resource is not protected, WebSphere ignores the login configuration.

XML file

The web.xml file located in the .war file’s WEB-INF directory contains the login configuration. The tools used on the IDE environment generate the web.xml file and we show it only in the case where you would like to browse it directly under the WebSphere HFS structure and look for the login configuration tags.

The web-app xml tags for the login configuration descriptor are shown in Example 16-1.

Example 16-1 XML Web application login configuration elements

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>WAS Forms Auth Example</realm-name>
    <form-login-config>
      <form-login-page>/WASFormLogin.html</form-login-page>
      <form-error-page>/WASFormError.html</form-error-page>
    </form-login-config>
</login-config>
```

Login configuration with WebSphere Studio Application Developer

The usual way an application programmer defines the login configuration involves the J2EE perspective of WebSphere Studio Application Developer. After double-clicking the .war application, WebSphere Studio Application Developer opens the Web deployment descriptor windows. Select the Pages folder to obtain a screen like that shown in Figure 16-3.

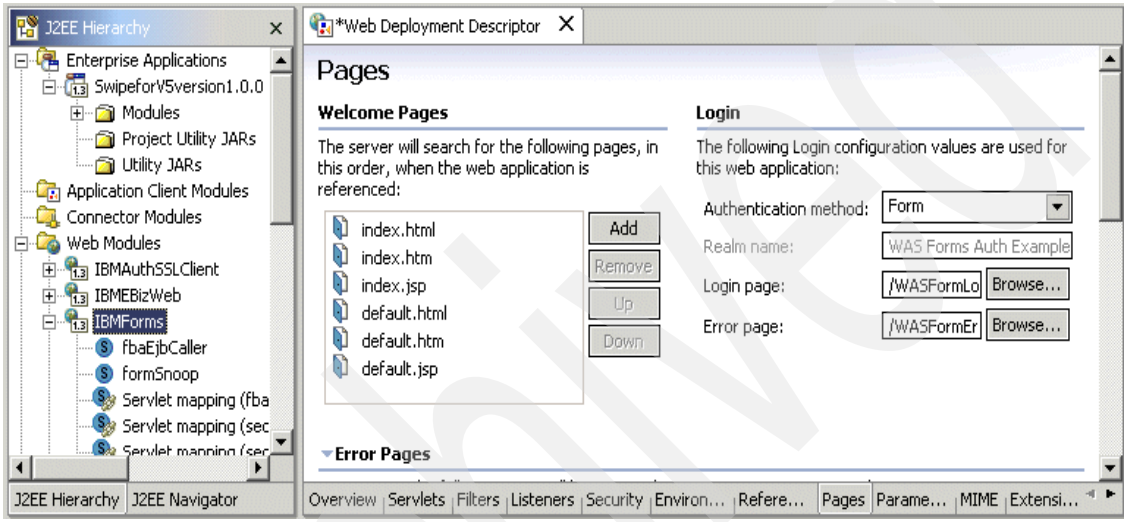


Figure 16-3 XML Web application login configuration with WebSphere Studio Application Developer

Login configuration with Application Assembly Tool (AAT)

Although most applications are delivered to an application deployer with the login configuration already set, the application deployer might need to add or change the login configuration just before installing the application in the application server. In this case, the AAT can be used in order to define a user login.

The deployer does the following:

1. Imports the application into the AAT.
2. Expands the application folder.
3. Expands the Web Modules folder. After selecting the Web application, the right window allows you to set the login configuration in the Advanced tab, as shown in Figure 16-4 on page 507.

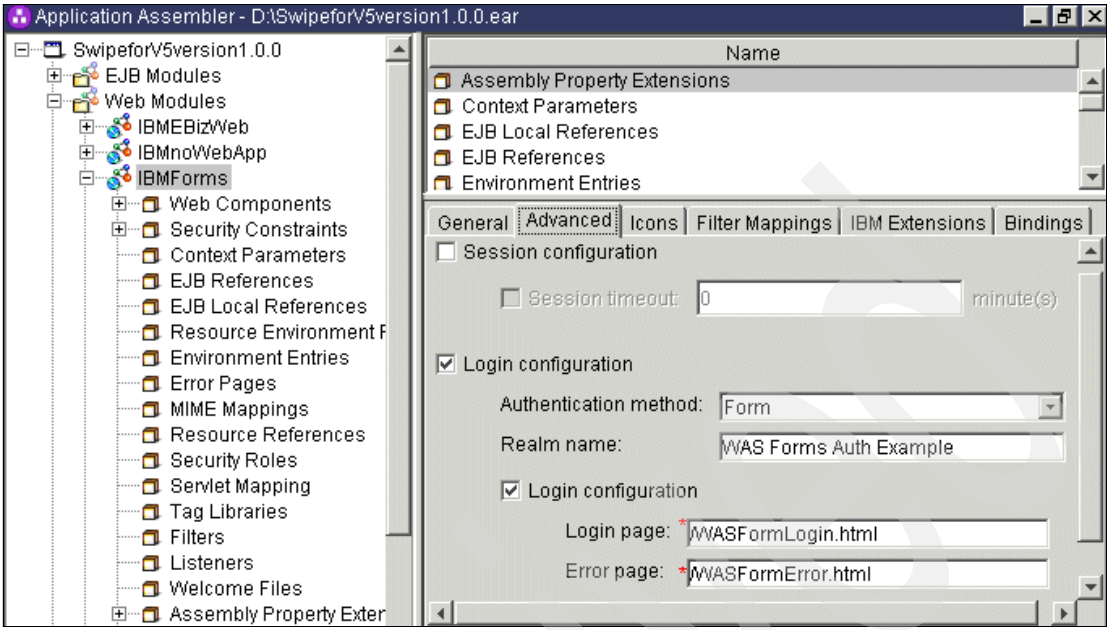


Figure 16-4 XML Web application login configuration with AAT

The authentication methods describe the way a user is challenged for a security token and how WebSphere should be configured to support this security token.

16.2.2 Basic authentication

This section describes basic authentication.

Overview

For the HTTP protocol, basic authentication has been defined in “HTTP Authentication: Basic and Digest Access Authentication,” RFC 2617. It is a simple challenge-response authentication scheme where the client is challenged for a user ID and a password. The Internet is divided into realms. A realm is supposed to have one user repository, so a combination of user ID and password is unique within a realm.

The user challenge contains the name of the realm so users with different user IDs and passwords on different systems know which one to apply.

For HTTP, the user challenge has the following format:

```
WWW-Authenticate: Basic realm="realm_name"
```

The user agent (for instance, a Web browser) returns the following HTTP header field:

```
Authorization: Basic userid:password
```

The string containing user ID and password is base64 encoded. For a description of base64 encoding, see “MIME Part One: Format of Internet Message Bodies,” RFC 2045. The purpose of base64 encoding is to avoid sending possibly unprintable or control characters over an interface that expects text characters. It does not provide any security because the clear text can readily be restored. Therefore, sending a user ID and password with basic authentication is equivalent to sending them in the clear.

Security can be improved by establishing an SSL or TLS session before the basic authentication challenge is issued. In this case, the entire communication between the two partners is encrypted. If an appropriate cipher suite is selected for the SSL session, there should be no risk of the password being exposed.

With basic authentication, error handling is rudimentary. The user is presented with the message `Authentication failed` and is offered an option to retry the authentication process. Specifically, should the user’s password have expired, there is nothing in the protocol that allows the user to be notified about this condition or to choose a new password. This is another reason why basic authentication is not suitable for use with a WebSphere Application Server for z/OS V5 system that is using RACF as the user registry.

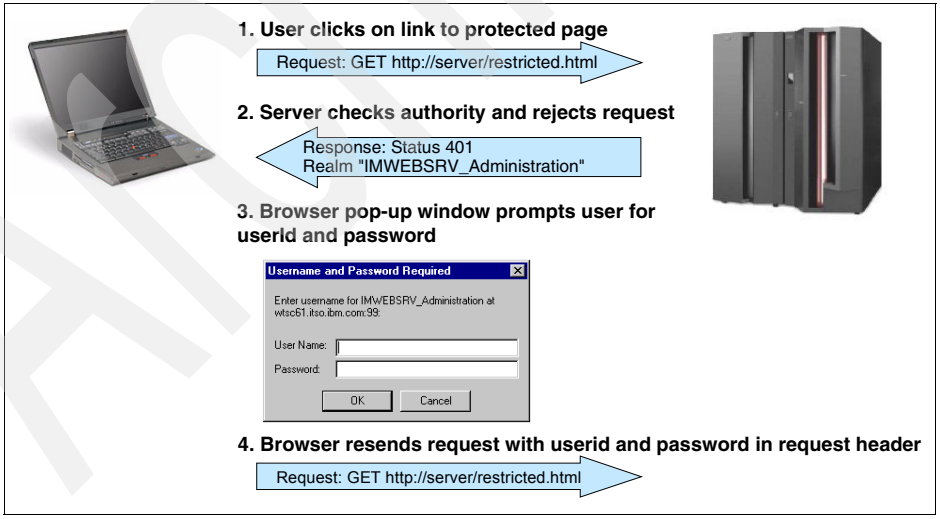


Figure 16-5 HTTP basic authentication flow

The flow as shown in Figure 16-5 on page 508 is as follows:

1. Using a Web browser, a user enters a URL or clicks a link. The browser translates this into a GET request, which is sent to the server whose address corresponds to the domain name or IP address in the URL.
2. The server checks the appropriate configuration file to determine if the target of this URL is protected and the user needs to be authenticated. Assuming that this is the user's first request to this server, no authentication has occurred so far, and therefore the server rejects the request and returns a HTTP return code of 401 (unauthenticated) to the browser. With the status code, it also returns the name of the server realm to the browser.
3. The Web browser displays a pop-up window that displays the server's realm and instructs the user that user ID and password are required.
4. After the user enters the required information and clicks **OK**, the browser resends the same GET request, this time with an Authorization header containing the user ID and password.

The server now repeats the check in step 2, using a call to the security manager (RACF, LDAP or CUR) to authenticate the user ID and password combination. If the authentication is unsuccessful, an HTTP return code of 401 is returned to the browser. Otherwise, request processing continues.

Why should I enable this function?: This authentication method is the most simple and does not require any coding effort or configuration. It is an easy mechanism to do tests.

Why might I not enable this function?: The user ID and password flow across the network at each request with a very weak encoding, so basic authentication must be reserved for secured or private networks. The password is checked at every user request that has an impact on user registry performance. Single sign-on is not possible with this mechanism. There is no possibility to set up a trust association interceptor.

Enablement

To enable basic authentication, modify the web.xml file as in Example 16-2 on page 510, using the appropriate tools.

Example 16-2 XML descriptors for basic authentication

```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>WASWebContainer</realm-name>
</login-config>
```

Or, select **Basic** in the Pages tab of the WebSphere Studio Application Developer Web Deployment Descriptor configuration menu shown in Figure 16-6.

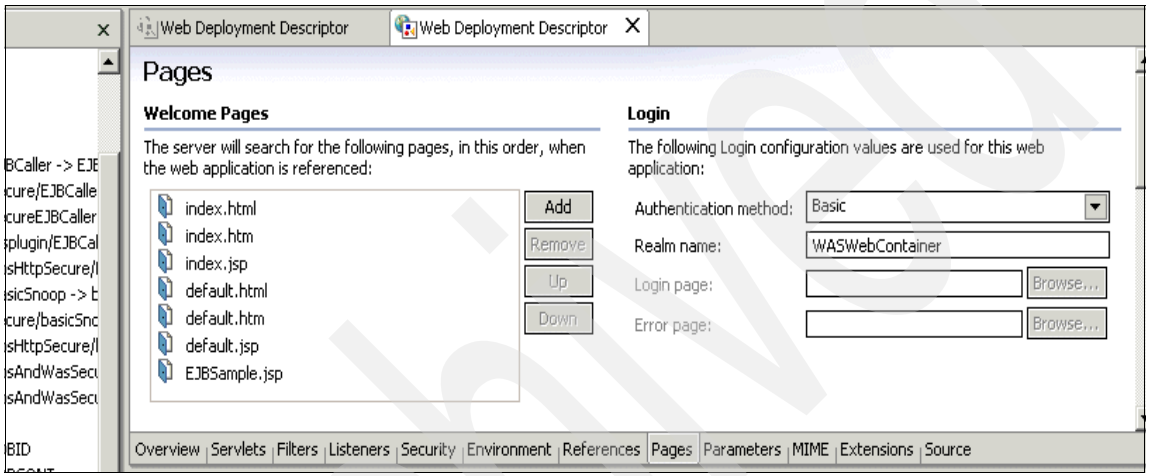


Figure 16-6 Selecting basic authentication with WebSphere Studio Application Developer

Or, select **Basic** from the authentication method list of the AAT Advanced tab, as shown in Figure 16-7.

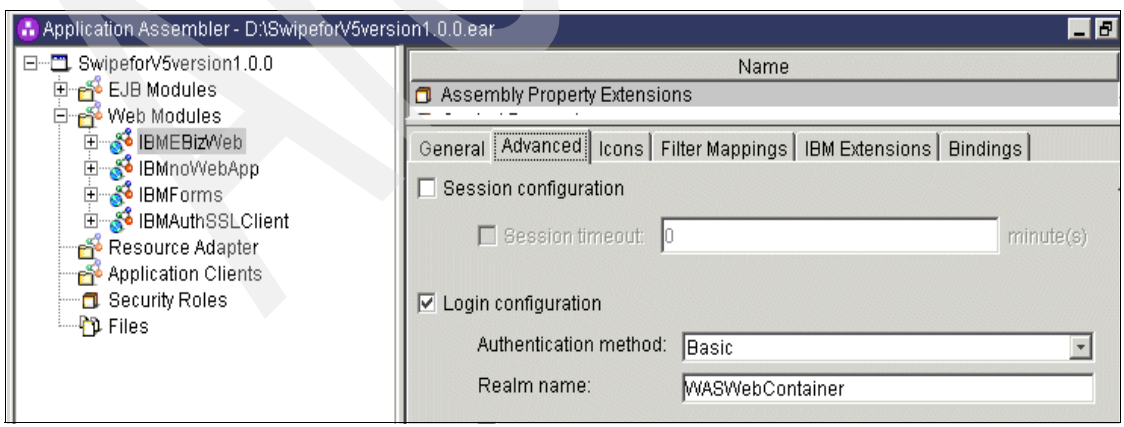


Figure 16-7 Selecting basic authentication with the AAT

Validation

To test basic authentication with SWIPE, we accessed the EJBCaller servlet in the IBMEBizWeb Web application using the `/IBMEBizWeb/secure/EJBCaller` URL.

Test case BA1: Basic authentication through transport handler

We tested basic authentication in a base Application Server with direct requests to the HTTP transport handler.

The LDAP registry was an IBM Directory Server 5.1 hosted on a Windows 2000 Intel-based machine.

The custom user registry was the FileRegistrySample as delivered with WebSphere.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet principal an the first page of the application. For all of the tests, we looked at the *Servlet Security Info* table to check that the principal was the user ID entered at the realm prompt. Figure 16-8 shows the Servlet Security Info table in SWIPE.

Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 16-8 Servlet Security Info table from SWIPE

Table 16-1 Basic authentication login test case 1 (BA1)

BA1 config: WebSphere base, transport handler							
Registry	**SSL**	**SSO**	**Input**			**Output**	
			User ID	**RunAs**	**Map-ping**	**Principal**	**Comments**
RACF	No	SWAM	stfan	Caller	none	stfan	
	No	LTPA	stfan	Caller	none	stfan	
	No	ICSF	stfan	Caller	none	stfan	

BA1 config: WebSphere base, transport handler							
Registry	**SSL**	**SSO**	**Input**			**Output**	
			User ID	**RunAs**	**Map-ping**	**Principal**	**Comments**
LDAP	No	SWAM	cn=stfan, ou=ITSO ,o=IBM,c =us	Caller	none	stfan	
	No	SWAM	stfan	Caller	none	stfan	
	No	LTPA	stfan	Caller	none	stfan	
	No	ICSF	stfan	Caller	none	stfan	
CUR	No	SWAM	stfan	Caller	none	stfan	
	No	LTPA	stfan	Caller	none	stfan	
	No	ICSF	stfan	Caller	none	stfan	

16.2.3 Form-based authentication

This section describes form-based authentication.

Overview

Basic authentication provides very little flexibility to the application designer because the user interface is the sole responsibility of the user agent, typically a Web browser. The application designer cannot change the way the challenge is presented to the user, nor is it possible to add error messages or other items to assist or instruct the user if an error of some kind occurred. In addition, the user ID and password flow in clear with each request.

To improve security and to allow for more flexibility, form-based authentication is provided. It allows the application designer to determine the look and feel for the user ID and password challenge the user receives. More importantly, form-based authentication allows handling of the error messages. Because a servlet or JavaServer Page can be used for the error handling, it can go far above just displaying an error message to the end user. For instance, it would be possible to prompt the user for a new password and change the password in case it has expired.

Form-based authentication can take place over an unencrypted or encrypted session. In either case, it offers several clear advantages over basic authentication. Form-based authentication only applies to the Web container.

When a user attempts to access a protected resource but has not been authenticated, the following occurs:

1. The associated login form is sent to the client and the URL path that required the authentication is stored by the container.
2. The user fills in the form, specifically the user ID and password fields, and posts the form back to the server.
3. The container attempts to authenticate the user using the user ID and password information from the form.
4. If the authentication fails, the error page is returned to the client and the status code of the HTTP return code is set to 401 (unauthenticated).
5. If the authentication succeeds, the Web container checks that the authenticated user ID is authorized to access the Web resource.
6. If the user is authorized, the client is redirected to the URL path that was stored by the container (see item 1).

It should be noted that the application designer, while being able to determine the look and feel of the login form and error page, cannot directly determine the point in time that the user is presented with the authentication challenge. The container sends the login form on the first request that attempts to access a protected resource. Thus, if you want the user to be prompted for authentication at a particular point in the application flow, you must make sure that the client request that takes place accesses a resource that requires authorization and (indirectly) forces the user to be authenticated at this point.

Form-based authentication does not fully address the lack of security in basic authentication. As with basic authentication, the password is transmitted in the clear and the server is not authenticated. If more security is required, the login form and error page should only be sent over an SSL or TLS session.

Under the form-based authentication protocol, the application provides two files. One is called the login screen (which could be a static HTML panel or a JSP) and is used to solicit a user ID and password from the end user if none is passed in the HTTP header. The other is an error screen that is returned to the user if the user ID and password do not meet the requirements of the security system. These screens provide a much richer opportunity for the application developer to tailor what the end user sees during the authentication process.

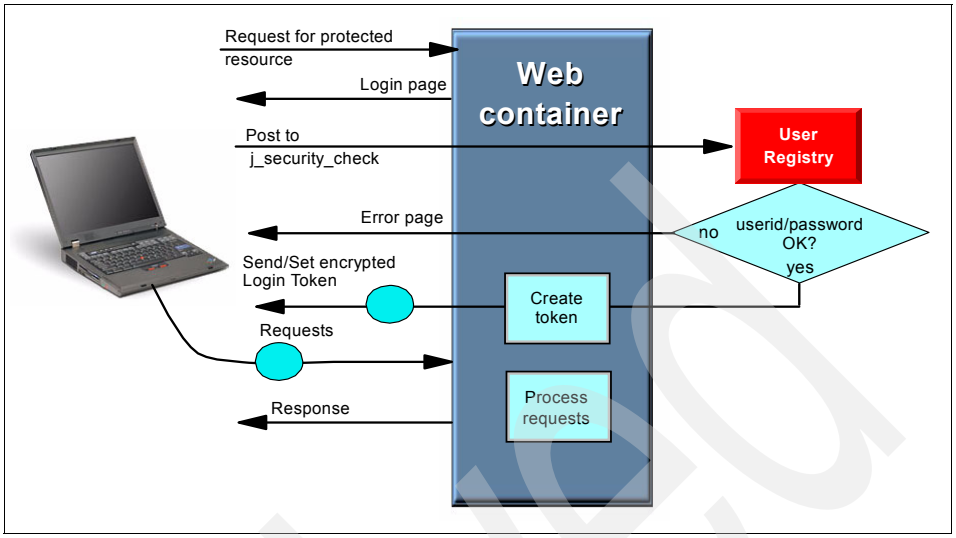


Figure 16-9 Form-based login flow

Figure 16-9 shows how, when the Web container receives a request for a protected resource, it returns the login screen defined by the application developer. The login screen must contain an HTTP form with the action being j_security_check. The form must solicit a user ID and password from the end user.

When the user clicks the action button, the j_security_check method (which is provided by the Web container vendor, not the application developer) is invoked to authenticate the user ID and password. If the authentication is unsuccessful, the error screen is returned. Otherwise, the original request is processed provided that the authenticated user is authorized to the Web application.

Furthermore, there is no need for the user ID and password to flow with each request. When the container authenticates the user ID and password, it creates a login session containing the principal and credentials. It can then return a token that is meaningful only in the container's operating environment. The browser can return this token instead of a user ID and password on subsequent requests. The token is valid until it is expired by the container or until the user uses the logout function.

Form-based authentication login flow in detail

If you need to understand the interactions between the browser client and WebSphere during a form-based authentication request, they are now described

in more detail here. This section explains the role of the Security Collaborator component and describes the different HTTP cookies involved.

1. On the first request, the user issues an HTTP GET for a protected resource within a Web application.
2. The Web container Security Collaborator of the target WebSphere server identifies the request as protected.
3. The Collaborator checks for a configured trust association interceptor (TAI).
4. If no TAI is configured, it searches the request headers for a security token.
5. Because this is the first time that the user has issued a request, there is no such a token. It then searches the deployment descriptor for the type of authentication specified for this resource that is found to be “form”. The Collaborator searches the configuration for the name of the form to be sent back to the client. It is at this point that the “WASReqURL cookie named is created. This cookie keeps track of the client’s original URL request.
6. A redirection to the configured login form is sent to the client, together with the WASReqURL cookie.
7. The client receives the redirection and issues an HTTP*GET* for the form.
8. The client receives the form, completes it, and posts to the specified “j_security_check” action, passing the WASReqURL cookie.
9. The Security Collaborator obtains the predefined fields from the form and checks with RACF to make sure they are legitimate.
10. If successful, it creates a security token using the Server’s configured ICSF key, creates a cookie called LTPA, and puts the token inside it.
11. The Security Collaborator also reads the WASReqURL cookie to know where the client wanted to go before the authentication happened and sends a redirect to the client specifying the URL found in the WASReqURL cookie. After WASReqURL is read, it is discarded.
12. If session affinity is configured, another cookie will also be created, called JSESSIONID. This cookie has no security information. Its purpose is to provide session routing information to the various WebSphere components in order to enable the correct routing of subsequent requests during the same client session to the same server region that last fulfilled the request.
13. The client gets the redirect, together with the LTPA cookie (and possibly JSESSIONID).
14. The client requests access to the protected resource.

15. The Security Collaborator sees that it is a protected resource and checks for the existence of the LTPA cookie. If it is there, it extracts the token and tries to decrypt it using the server's configured SSO (ICSF or LTPA) key. If successful, it gives access to the wanted resource.

On subsequent requests, as long as the security credentials are valid, only steps 14-15 reoccur.

Note: It should be noted that in IBM WebSphere Application Server for z/OS and WebSphere Application Server V5 for distributed platforms, the LTPA cookie mentioned above has nothing but the name in common with the Lightweight Third Party Authentication approach. In previous versions of distributed WebSphere Application Server, the form-based login required the LTPA mechanism to be enabled, so the form-based login relied on the LTPA WebSphere Application Server mechanism to build the form-based login LTPA cookie. This is no longer true, but we now have different objects sharing the same name.

HTTP session implications

Another possible interaction occurs when there are HTTP sessions involved and if the client posts a form with action name defined as "ibm_security_logout". This triggers some code that discards the JSESSIONID cookie in the specific WebSphere Application Server's Web container and invalidates the LTPA cookie hosted in the browser. Consequently, the next time the client tries to obtain access to the given protected resource, the Security Collaborator will see that it does not have a record of the authentication (cookie) and forces the client to authenticate again. Deleting of the JSESSIONID cookie at logout has the effect of "unbinding" the user to a particular servant region (that is, the session affinity is deleted) and this means that the control region can now choose any servant to handle the next request from that user.

Note: During our tests we found that ibm_security_logout was not honored by the ICSF authentication mechanism. We tested an early fix that will be included in APAR PQ77065 PTF.

This automatic cookie creation and manipulation is only possible because the Web container code contains functions that have been specifically written to handle those scenarios. That is why the login forms need to make use of specific parameter names ("j_username" and "j_password") and use specific ACTIONs of "j_security_check" for login and "ibm_security_logout" for logout.

The WebSphere Application Server development team could not possibly provide a solution for all the different possible types of authentication topologies.

Therefore, the main objective is to deliver a solution that addresses the simpler scenarios and provides a framework within which more complex solutions can be built. This will become more apparent when we look into the inadequacies of the current form-based authentication specification compared to typical customer requirements.

If the login session is shared in a security domain, sometimes called a *realm*, it allows a single sign-on capability within that domain. The J2EE specification requires vendors to support single sign-on, but does not define the scope of a security domain.

Configuration considerations

Because the encryption key used is unique per server instance or single sign-on domain but not per client, cookies are potentially subject to “cookie stealing” if the session is not protected with SSL. A stolen cookie that has not expired might be used on the same server instance (or single sign-on domain) by the attacker. To reduce the exposure involved with “cookie stealing”, the expiration time period should usually be set to be in the order of minutes. The value is set in the LTPA (see Figure 16-20 on page 534) and ICSF (Figure 16-23 on page 536) configuration panels.

Within insecure networks where cookie stealing could be a problem, only SSL sessions should be used with form-based authentication. Using SSL can be enforced by setting the Requires SSL check box in the LTPA (see Figure 16-20 on page 534) and ICSF (Figure 16-23 on page 536) configuration panels.

Why should I enable this function? Compared to basic authentication, form-based authentication lets you customize the HTML login prompt. For z/OS customers with an SAF security manager such as RACF, form-based authentication is a good solution for authentication when accessing intranet applications.

After user authentication, the password does not flow across the network on every request. This reduces the risk of the user ID and password being stolen, but as with basic authentication, you must use SSL to provide confidentiality of the login process.

The fact that the user ID and password do not flow on each request also reduces the load on the security manager, so form-based authentication should perform better.

Why might I not enable this function? Someone has to create the HTML login and error pages before you can use form-based authentication.

The password flows through the network unencrypted at least once (but you can enable SSL V2 on a server to avoid this).

The J2EE specification for form-based authentication does not include support for important things such as password expiration and change.

Enablement

To enable form-based login, you need an HTML login page that delegates authentication to the j_security_check internal servlet.

Then, to enable form-based authentication, either add or modify the web.xml file as in Example 16-3.

Example 16-3 XML descriptor for form-based authentication

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>WAS Forms Auth Example</realm-name>
    <form-login-config>
      <form-login-page>/WASFormLogin.html</form-login-page>
      <form-error-page>/WASFormError.html</form-error-page>
    </form-login-config>
</login-config>
```

Or, select **Form** in the Pages tab of WebSphere Studio Application Developer Web deployment descriptors configuration menu, as in Figure 16-10 on page 519, and fill in the Login and Error page fields.

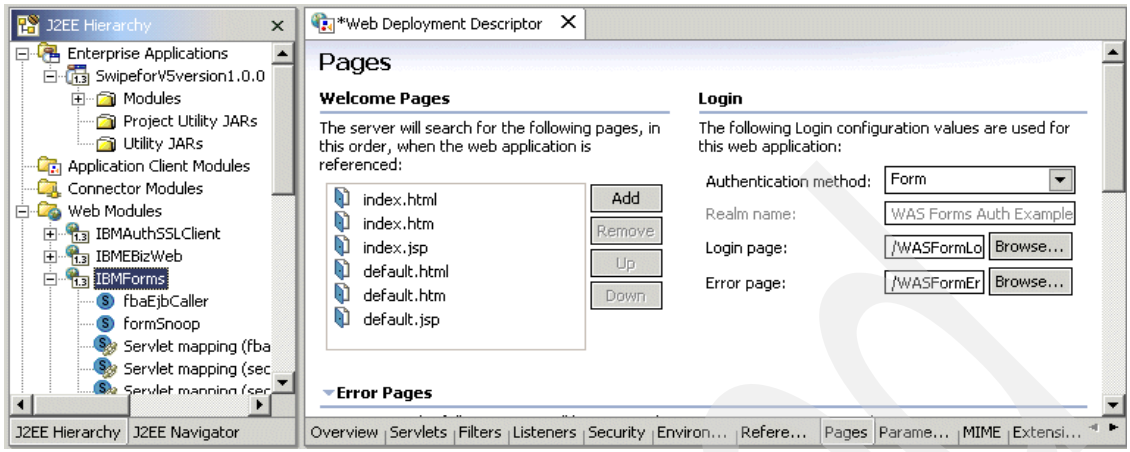


Figure 16-10 Selecting form-based authentication with WebSphere Studio Application Developer

Or, alternatively select **Form** from the authentication method list of the AAT advanced tab, as in Figure 16-11.

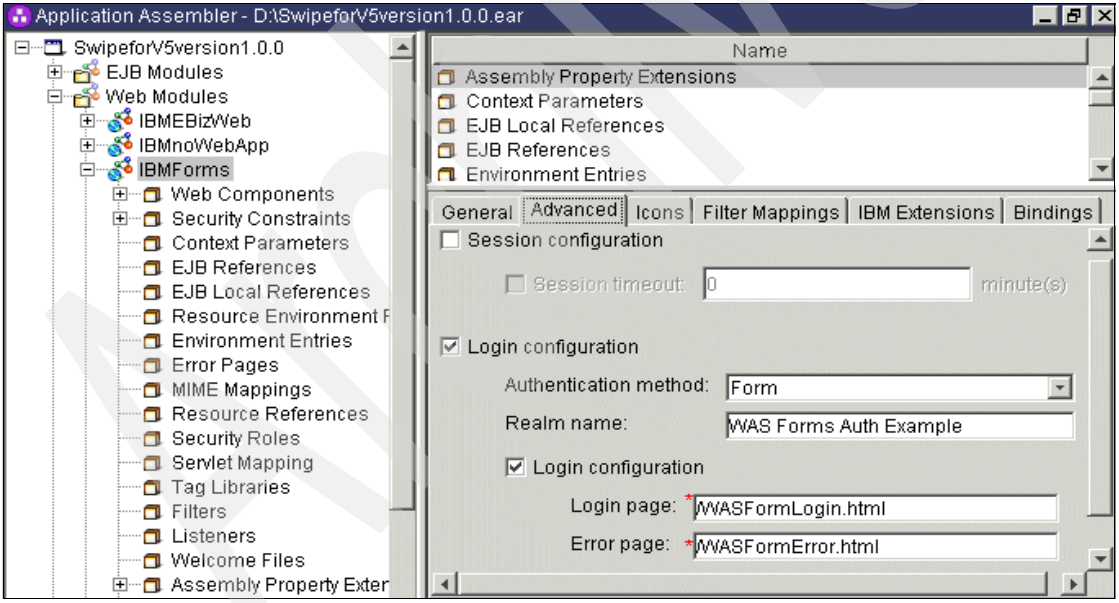


Figure 16-11 Selecting form-based authentication with the AAT

Validation

To test form-based authentication with SWIPE, we accessed the fbaEjbCaller servlet in the IBMForms Web application using the `/IBMForms/secure/fbaEjbCaller` URL.

Test case FBL1: Form-based login using transport handler

We tested form-based login in a base Application Server with direct requests to the HTTP transport handler.

The LDAP registry was an IBM Directory Server 5.1 hosted on a Windows 2000 Intel-based machine.

The custom user registry was the FileRegistrySample as delivered with WebSphere.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet current principal on the first page of the application. For all of the tests, we looked at the Servlet Security Info table to make sure the principal was the user ID entered in the form login page. Figure 16-12 shows the Servlet Security Info table in SWIPE.

Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 16-12 Servlet Security Info table from SWIPE

Table 16-2 Form-based login test case 1 (FBL1)

FBL1 config: WebSphere base, transport handler						
Registry	**SSL**	**Authentication mechanism**	**Input**		**Output**	
			User ID	**RunAs**	**Principal**	**Comments**
RACF	No	SWAM	stfan	Caller	stfan	
	No	LTPA	stfan	Caller	stfan	
	No	ICSF	stfan	Caller	stfan	
LDAP	No	SWAM	stfan	Caller	stfan	
	No	LTPA	stfan	Caller	stfan	
	No	ICSF	stfan	Caller	stfan	

FBL1 config: WebSphere base, transport handler						
Registry	**SSL**	**Authentication mechanism**	**Input**		**Output**	
			User ID	**RunAs**	**Principal**	**Comments**
CUR	No	SWAM	stfan	Caller	stfan	
	No	LTPA	stfan	Caller	stfan	
	No	ICSF	stfan	Caller	stfan	

16.2.4 Enhanced form-based login

Basic and form-based authentication are described in the Java 1.3 Servlet specifications and are supported by IBM WebSphere Application Server for z/OS V5. However, the design of both basic and form-based authentication does not provide the ability to handle expired passwords. In response to customer requests, programmers from WebSphere Application Server development and the Washington Systems Center created an authentication method that addresses the limitations in basic and form-based authentication. It is the Enhanced Form-Based Authentication (EFBA), and it handles expired passwords. If a user wants to change the password before it expires, EFBA handles that as well. It also provides more information to the user in cases where the authentication failed.

Why use EFBA?

If your applications require that users authenticate using RACF user IDs and passwords, and you have a RACF password interval, you should consider implementing EFBA. It's the only authentication method that supports the changing of expired passwords as well as user-initiated password changes.

An overview of EFBA

Enhanced Form-Based Authentication (EFBA) builds on the form-based authentication function in WebSphere, and requires that it be enabled. It is recommended that you review the section on form-based authentication method before you attempt to understand EFBA.

The business application is protected by form-based authentication, with a modified login form. The inability of form-based authentication to handle expired passwords is due to limitations of j_security_check, the method that handles authentication. EFBA replaces the j_security_check method with a separate authentication servlet. The ACTION statement of the form-based authentication login form points to the URL of the EFBA authentication servlet instead of j_security_check.

The EFBA servlet uses Java for z/OS SAF interfaces (also known as racf.jar) to validate the client's user ID and password, handle expired passwords, handle user-initiated password changes and provide information to the user in the event of an authentication failure. For successful authentications, the EFBA servlet creates a login token and stores it on the client browser, just as j_security_check does. The EFBA servlet then redirects the client to the business application, where the login token is used to authenticate the client using form-based authentication.

The Java for z/OS SAF interface used by the EFBA servlet calls the z/OS UNIX __passwd callable service. In order to use __passwd, the calling program must be loaded from a controlled library. This means that in the process of enabling EFBA, you must mark several WebSphere MVS libraries and HFS programs as program-controlled. By marking these program files and libraries as program-controlled, you must authorize any application in the server to use the Java for z/OS SAF interfaces. If you want to limit authorization to use the Java for z/OS SAF interfaces to just the EFBA servlet, you should investigate securing the interfaces using Java 2 security or run the EFBA servlet in a separate server.

As with form-based authentication, the use of SSL is strongly recommended when EFBA is used. User IDs, passwords, the login token, and possibly sensitive application data flow between the client and the server. SSL (with a suitable encryption suite) is needed to protect this information.

Authentication flow using EFBA

The example below describes the authentication flow steps for a business application using EFBA.

1. Using a Web browser, type in the URL of the business application.
2. The Web container processes the deployment descriptor for the business application. The deployment descriptor specifies that form-based authentication is the authentication mechanism for this application. Because this is the first time you have issued a request, the Web container presents the application's login page to you. The Web container also writes a cookie (WASReqURL).
3. You receive the form, complete it, and click the login button on the form. This posts the form data and the WASReqURL cookie to the URL of the EFBA servlet, as specified by the ACTION statement in the login form.
4. The Web container determines that the EFBA servlet has no authentication requirements. The EFBA servlet takes the user ID and password values from the login form and calls the Java for z/OS SAF interfaces to validate the user ID and password on the login form. If you specified a new password (twice) along with the user ID and password, the EFBA servlet passes those to the Java for z/OS SAF interfaces. The EFBA servlet receives SAF return code

and reason code information from the Java for z/OS SAF interfaces. In the event that the return code and reason code are not zero, the EFBA authentication servlet presents a status page to you explaining what the problem is, and the process stops here.

5. If the SAF return code and reason code received by the EFBA servlet are both zero, the EFBA servlet calls WebSphere to create a security token using the server's ICSF or LTPA key. A cookie called LTPA is created and the security token is placed in it.
6. The EFBA servlet reads the WASReqURL cookie to find out where you wanted to go before the authentication happened, and sends a redirect to you specifying the URL found in the WASReqURL cookie. After WASReqURL is read, it is discarded.
7. If session affinity is configured, another cookie is also created, named JSESSIONID. This cookie has no security information. Its purpose is to provide session routing information to the various WebSphere components in order to enable the correct routing of subsequent requests during the same client session to the same server region that last fulfilled the request.
8. The client gets the redirect, together with the LTPA cookie (and possibly JSESSIONID).
9. The client requests access to the business application.
10. The Security Collaborator sees that it is a protected resource and checks for the existence of the LTPA cookie. If it is there, it extracts the token and tries to decrypt it using the key specified by the authentication mechanism in use by the server (either ICSF or LTPA). If successful, it gives access to the wanted resource.

On subsequent requests, as long as the security credentials are valid, only steps 9-10 reoccur.

Obtaining and implementing EFBA

The sample EFBA authentication servlet, along with instructions to implement it, are available at:

`http://www.ibm.com/support/techdocs`

Or, at the following address:

`http://www.ibm.com/software/webservers/appserv/zos_os390/`

Select **Support** and search for `TD101255`. The Techdoc name is *Implementing Enhanced Form Based Authentication with Servlet Filters in J2EE Applications under WebSphere Application Server v5.0.x and v5.1 for z/OS*.

16.2.5 Certificate-based authentication

This section describes certificated-based authentication.

Overview

Clients can authenticate to a server by presenting an X.509 digital certificate and proving ownership of the private key that is associated with the certificate. Client authentication with a certificate is part of the Secure Sockets Layer (SSL) V.3 or Transport Layer Security (TLS) 1.0 protocol. This means that an SSL or TLS session is required. The transfer of the client certificate and the certificate verify message that proves ownership of the private key is part of the SSL handshake.

Because System SSL repertoires are the only repertoires supported for HTTP client authentication in WebSphere Application Server for z/OS V5, RACF is called to validate the X.509 digital certificate even when the WebSphere user registry is LDAP or a CUR. In these cases, RACF provides three different ways to associate a RACF user ID with a client certificate:

- A certificate can be stored in the RACF database with a user ID associated with it. This can either be done using the RACF RACDCERT command or using a certificate self-registration application that allows Web browser clients to register their own certificates in RACF (a sample application is provided in SYS1.SAMPLIB). This method provides a one-to-one relationship between certificates and user IDs.
- Certificate name filter rules can be defined in RACF using the RACDCERT command. If the portion of the certificate owner's distinguished name or the certificate issuer's distinguished name in the filter rule match the certificate, the user ID assigned to the filter rule is associated with the certificate. This method is typically not used to provide a one-to-one mapping between certificates and user IDs because this would require a separate filter rule for each certificate. Certificate name filter rules are used in cases where a number of certificates map to a common user ID, typically a surrogate user ID, which represents a set of users with the same security requirements.
- A certificate can contain an extension named *hostIdMappings*. This extension contains a list of host names and associated user IDs. If the certification authority that signed the client certificate is trusted for this extension and an entry in the list corresponds with the host name of the system, RACF uses the user ID from the list entry.

The container behaves similarly with HTTPS basic authentication. HTTPS uses Secure Sockets Layer (SSL) to establish a secure, encrypted session between the browser and the HTTP server. If someone on the Internet intercepts the session, the user ID and password are not easily discovered.

The J2EE specification requires that containers support the use of client certificates for authentication. The J2EE specification refers to this as SSL Mutual Authentication.

For a detailed view of the SSL handshake, during which the client certificate is sent to the server, refer to Figure 9-1 on page 244 and the associated discussion. When client authentication (for example, a com.ibm.ssl.protocol value of SSLv3) is specified in the WebSphere SSL repertoire, the SSL handshake will include the optional server “certificate request” message. If the client cannot provide a certificate, it responds with the “no certificate” alert, which might cause the WebSphere server to close the connection.

Note: The *MutualAuthCBindCheck* custom property on HTTPS transport indicates how a client certificate should be resolved to an SAF principal. Default is false. If this property is set to true:

- All SSL connections from a browser must have a client certificate.
- The client certificate should map to a valid RACF user ID.
- The user ID associated with that client certificate must have RACF CONTROL authority for CB.BIND.*servername*, where *servername* is the value of the *Cluster transition name*.

If these conditions are not met, the connection is closed.

Important: Before the WebSphere APAR PQ76644 level, WebSphere tried to map SSL client certificates with a RACF user ID wether or not the user registry was RACF and wether or not *MutualAuthCBindCheck* on HTTPS was set. If the mapping failed, the request was rejected.

Why should I enable this function?: All the Web flow is encrypted. There is no password flowing between the user and the server.

Why might I not enable this function?: SSL V3 encryption and decryption have a CPU cost. The more users you have the more certificates you manage, and you might not want to implement a full Public Key Infrastructure (PKI).

Enablement

To enable certificate-based authentication, either add or modify the web.xml file as in Example 16-4 on page 526.

Example 16-4 XML descriptor for certificate-based authentication

```
<login-config>
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>WAS SSL Client Example</realm-name>
</login-config>Client-
```

Or select **Client-Cert** in the Pages tab of the WebSphere Studio Application Developer Web deployment descriptors configuration menu as in Figure 16-13 and fill in the Login and Error page fields.

Figure 16-13 Selecting certificate-based authentication with WebSphere Studio Application Developer

Or alternatively select **Client cert** from the authentication method list of the AAT advanced tab as in Figure 16-14.

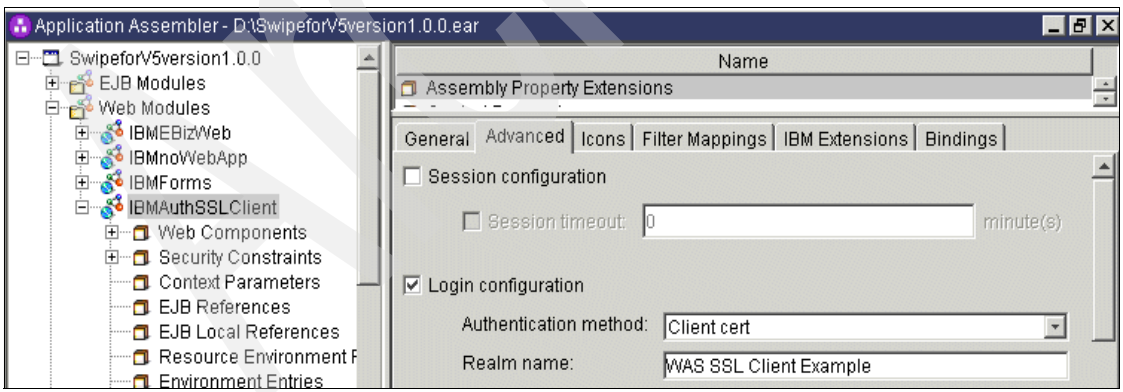


Figure 16-14 Selecting certificate-based authentication with the AAT

Enablement in WebSphere

To enable SSL certificate authentication in WebSphere, you need to insure that the **Client Authentication** check box on the current HTTPS transport is selected as shown in Figure 16-15. By default, it is not selected.

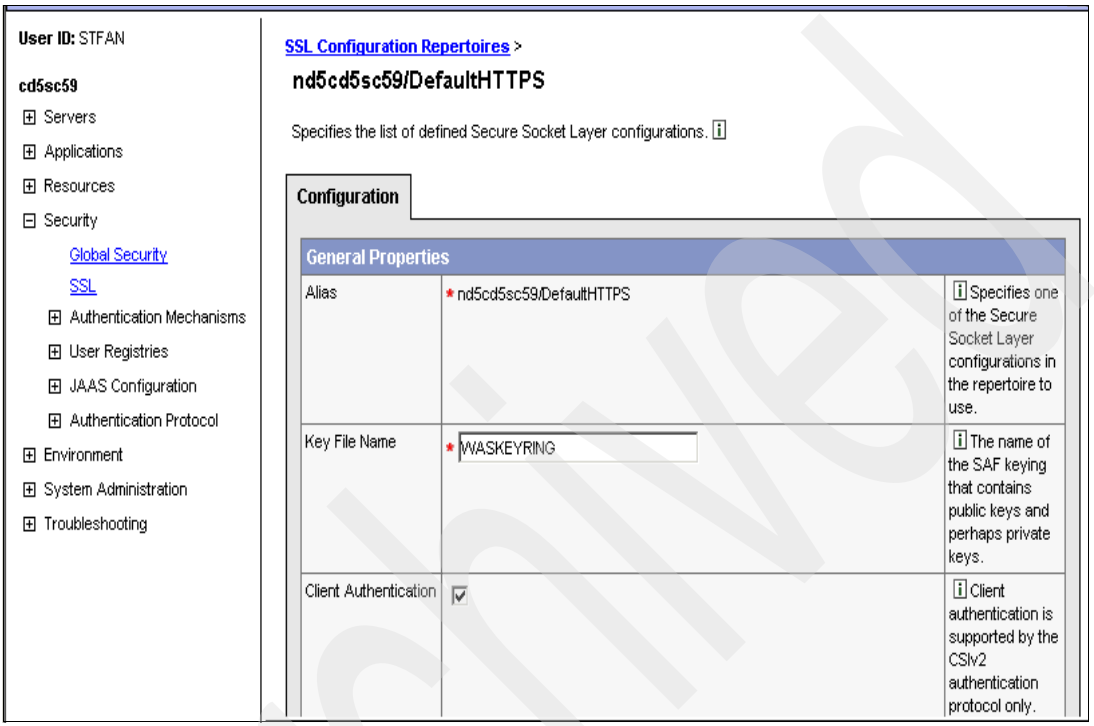


Figure 16-15 Enabling certificate authentication

Validation

To test certificate authentication with SWIPE, we accessed the EJBCaller servlet in the IBMClientSSL Web application using the `/IBMClientSSL/secure/EJBCaller` URL.

Test case SSL1: SSL certificate login through transport handler

We tested certificate login in a base Application Server with direct requests to the HTTPS transport handler with MutualAuthCBindCheck set to false.

The LDAP registry was an IBM Directory Server 5.1 hosted on a Windows 2000 Intel-based machine.

The custom user registry was the FileRegistrySample as delivered with WebSphere.

We created the user certificate with the following RACF commands:

```
RACDCERT ADDRING(WASKEYRING) ID(STFAN)
RACDCERT ID (STFAN) GENCERT +
  SUBJECTSDN(CN('stfan') OU('ITSO') O('IBM') c('us')) +
  WITHLABEL('STFANLDAP') SIGNWITH(CERTAUTH LABEL('WS for z/OS CA'))
RACDCERT ID(STFAN) CONNECT (LABEL('STFAN') RING(WASKEYRING) DEFAULT)
```

Then, we copied them to HFS:

```
RACDCERT ID(STFAN) EXPORT(LABEL('STFANLDAP')) +
   DSN(CERTLDAP.P12BIN) FORMAT(PKCS12DER) PASSWORD('XXXXXXXX')
OPUT CERTLDAP.P12BIN '/tmp/stfanldap.p12' binary convert(no)
```

To install the certificate into Windows, we downloaded with FTP the p12 type file to a local Windows 2000 disk and double-clicked the file. This opened the Windows Certificate Import Wizard. We just followed the instructions.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet principal an the first page of the application. For all of the tests, we looked at the Servlet Security Info table to check that the principal was the user ID mapped by RACF with the certificate. Figure 16-16 shows the Servlet Security Info table in SWIPE.

Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 16-16 Servlet Security Info table from SWIPE

Table 16-3 SSL certificate-based login test case 1 (SSL1)

SSL1 config: WebSphere base, transport handler, RACF issued certificate							
Registry	**SSL**	**Authc. mech.**	**Input**			**Output**	
			User ID	**RunAs**	**Map-ping**	**Principal**	**Comments**
RACF	Yes	SWAM	cert	Caller	none	STFAN	access
	Yes	LTPA	cert	Caller	none	STFAN	access
	Yes	ICSF	cert	Caller	none	STFAN	access

SSL1 config: WebSphere base, transport handler, RACF issued certificate							
Registry	**SSL**	**Authc. mech.**	**Input**			**Output**	
			User ID	**RunAs**	**Map-ping**	**Principal**	**Comments**
LDAP	Yes	SWAM	cert	Caller	none		failed [a]
	Yes	LTPA	cert	Caller	none		failed [a]
	Yes	ICSF	cert	Caller	none		failed [a]
CUR	Yes	SWAM	cert	Caller	none	CN=stfan, OU=ITSO, O=IBM, C=us	access [b]
	Yes	LTPA	cert	Caller	none	stfan	access
	Yes	ICSF	cert	Caller	none	stfan	access

a. Problem fixed with level W501000 (PTF UQ80305).
b. Need to create a user ID in CUR registry as the complete certificate domain name.

16.2.6 Password management

A SAF-based user registry performs password expiration. LDAP does not perform password expiration. Tivoli Access Manager performs password expiration. LDAP on z/OS with native authentication performs password expiration. Depending on where you store the user registry passwords and depending on your security system, password expiration occurs or not.

In any secure environment with a serious password policy, password expiration occurs. This raises the following question: How can end users change their password by themselves when it expires or even before it expires?

Not all authentication methods support expired passwords nicely; RFC or Java standard implementations fail to deliver expired password support. We describe solutions available depending on where the password is kept and depending on your configuration.

Passwords stored in the SAF user registry

Consider the following solutions when the passwords are stored in the SAF user registry:

- WebSphere with LocalOS registry and form-based login authentication: With this configuration, the solution is to use the enhanced form-based login described in 16.2.4, “Enhanced form-based login” on page 521. This solution

enables you to handle expired password, invalid user ID or revoked user ID situations.

- WebSphere with LocalOS registry and basic authentication: With this configuration, there is no included feature to let end users change their password. Meanwhile, users can change their password by starting a TSO session or a UNIX System Services Telnet session if they are allowed to do so.
  If you use the IBM HTTP Server for z/OS, you might consider using the "pwapi" function. It is a GWAPI sample that enables end users to change their expired z/OS password using a browser. It is available at the following address:

  ```
  http://www.ibm.com/software/webservers/dgw/apisamp.htm
  ```

- WebSphere with LDAP registry and native authentication: With this configuration, there is no included feature to let end users change their password. The password is not kept inside LDAP but inside RACF. Therefore, it expires. Nevertheless, users can change their password by starting a TSO session or a UNIX System Services Telnet session if they are allowed to do so.
  If you use the IBM HTTP Server for z/OS, you might consider using the "pwapi" function. It is a GWAPI sample that enables end users to change their expired z/OS password using a browser. It is available at the following address:

  ```
  http://www.ibm.com/software/webservers/dgw/apisamp.htm
  ```

- WebSphere integrated with Tivoli Access Manager and native authentication: With this configuration, the password can expire because of the RACF password policy or because of the Tivoli Access Manager password policy. End users benefit from the Tivoli Access Manager capabilities. If their password expires, they will be redirected to a change password Web page after the first login attempt. In addition, end users can change their password anytime at the following address:

  ```
  https://<web_portal_manager_hostname>:9443/delegate
  ```

  End users log on and select **Change My Password** from the Task List.

- WebSphere with LocalOS registry front-ended by IBM HTTP Server for z/OS doing basic authentication, accessing the same SAF-based registry: With this configuration, IBM HTTP Server for z/OS enables end users to change their expired password, specifying the following values in the Basic Authentication pop-up password field:

  ```
  <old_password>/<new_password>/<new_password>
  ```

  This is documented in *z/OS HTTP Server Planning, Installing, and Using Guide*, SC34-4826.

Passwords stored in the LDAP user registry

Consider the following solutions when the passwords are stored in the LDAP user registry:

- WebSphere with LDAP registry and no native authentication: With this configuration, neither WebSphere Application Server nor LDAP perform password expiration. Neither WebSphere Application Server nor LDAP provide a good end-user interface to change end-users password. Tools such as the **ldapmodify** command line utility or a LDAP browser should let you change passwords. You can find a free LDAP browser at the following address:

  `http://www.iit.edu/~gawojar/ldap/`

- WebSphere integrated with Tivoli Access Manager and no native authentication: With this configuration, end users benefit from the Tivoli Access Manager capabilities. If their password expires, they will be redirected to a change password Web page after the first login attempt. In addition, end users can change their password anytime at the following address:

  `https://<web_portal_manager_hostname>:9443/delegate`

  End users log on and select **Change My Password** from the Task List.

Passwords stored in a custom user registry

In this case, password management entirely depends on your custom user registry implementation. Refer to your custom user registry specifications to discover how to manage passwords.

16.2.7 WebSphere authentication mechanisms

An authentication mechanism defines rules about security information, including whether a credential is forwardable to another Java process, and the format in which security information is stored in both credentials and tokens.

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine, or an application.

An authentication mechanism in WebSphere typically collaborates closely with a user registry. The user registry is the repository of user and group accounts that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a credential that is a WebSphere internal representation of a successfully-authenticated client user. Not all credentials are created equal. The abilities of the credential are determined by the configured authentication mechanism.

Although WebSphere provides several authentication mechanisms, only a single “active” authentication mechanism can be configured within a cell. WebSphere provides three authentication mechanisms and you must select one of these when enabling global security. The choice is between SWAM, LTPA, and ICSF, and these are now described.

Simple WebSphere Authentication Mechanism (SWAM)

SWAM stands for Simple WebSphere Authentication Mechanism because this authentication mechanism does not provide any additional functionality to the login facilities previously discussed.

Important: SWAM is only supported in base server configurations.

SWAM is intended for simple, non-distributed, single application server type runtime environments. The single application server restriction is due to the fact that SWAM does not support forwardable credentials.

Using SWAM does not require additional configuration for IBM WebSphere Application Server for z/OS V5; you can simply select **SWAM** as the authentication mechanism on the Global Security page, as shown in Figure 16-17.

Figure 16-17 Enabling SWAM for the cell

Why should I enable this function? You are running only base Application Servers and are not interested in single sign-on capabilities nor trust association interceptor support.

Why might I not enable this function? You plan to run in a Network Deployment-controlled application server. You need single sign-on capabilities. You plan to use trust association interceptor code.

Lightweight Third Party Authentication (LTPA)

Lightweight Third Party Authentication (LTPA) was intended for distributed, multiple application server and machine environments. It supports forwardable credentials, and therefore supports single sign-on. LTPA can support security in a distributed environment through the use of cryptography.

Even if LTPA is typically configured using a central shared user registry (such as LDAP, a Windows domain type registry, or a custom user registry) you can use it with an SAF-based registry.

LTPA does require configuring an LTPA key that will be used to encrypt the cookie passed back to the browser after a successful authentication. This cookie, also known as the LTPA token, contains the user ID and validity duration of the authenticated user. The LTPA key can also be imported from another server participating in the same single sign-on domain, or you can export the key you generated in order to provide it to other WebSphere domain administrators.

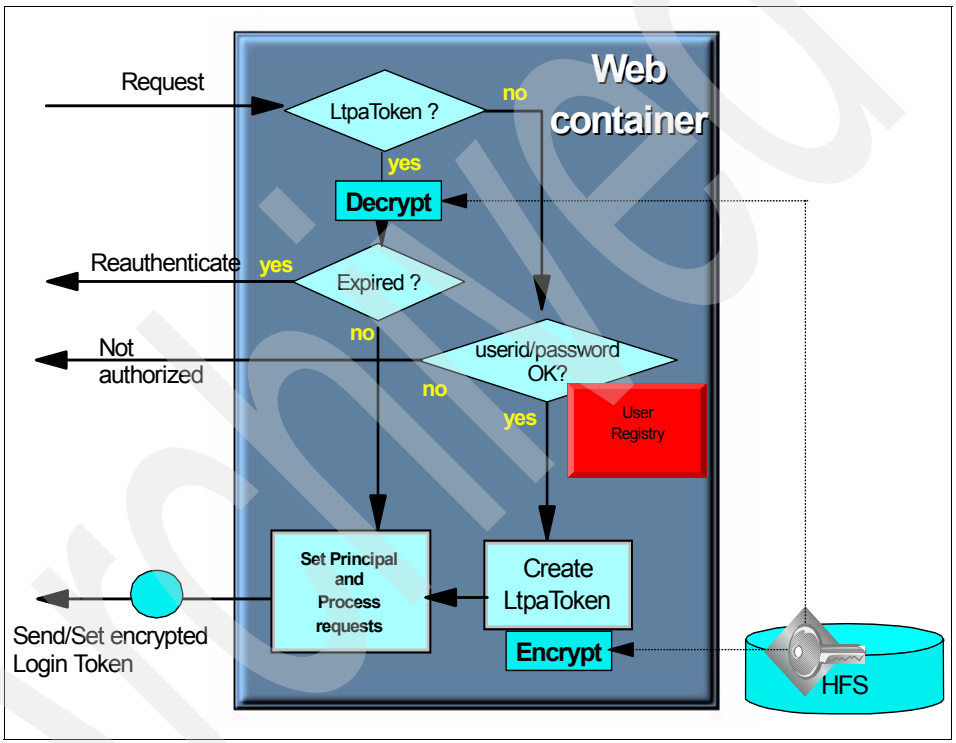


Figure 16-18 LTPA authentication mechanism

You must provide the time period in minutes at which an LTPA token will expire.

Note: Because the LTPA mechanism relies on the LTPA token cookie, it might happen during tests that a browser already has an LTPA token from a previous session. As a result, before and between LTPA tests, clean your cookies from your browser. Otherwise, you might face some `SECJ4034I Token Login failed` error because of an old cookie.

The LTPA configuration panel opens, as shown in Figure 16-19.

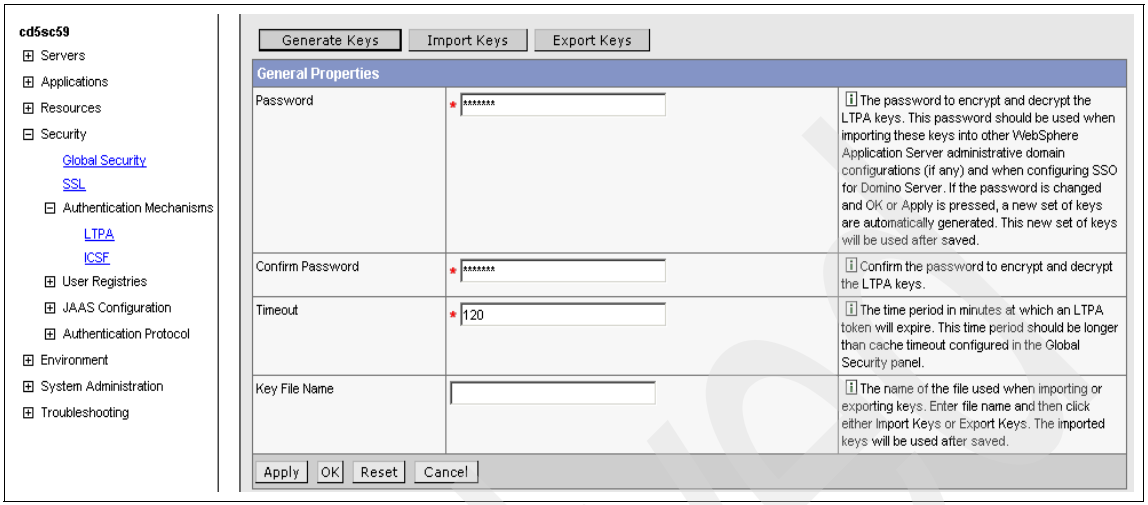


Figure 16-19 LTPA configuration page

To activate LTPA, you must use the WebSphere administrative console and select **LTPA** as the authentication mechanism on the Global Security page, as shown in Figure 16-20.

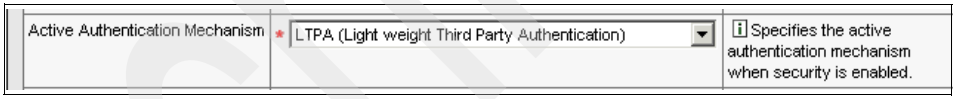


Figure 16-20 Enabling LTPA for the cell

Why should I enable this function?: You plan to run in a Network Deployment-controlled application server. You need single sign-on capabilities with applications running on distributed platforms. You plan to use trust association interceptor code.

Why might I not enable this function?: You need single sign-on capabilities with WebSphere Application Server for z/OS and OS/390 V4.0.1.

You do not want to suffer an interrupt while the LTPA key is updated on all systems in the LTPA realm.

Integrated Cryptographic Service Facility (ICSF)

WebSphere's ICSF security mechanism runs in the same realm as LTPA except that the encryption keys are stored in the Cryptographic Key Data Set (CKDS),

encrypted with the Triple DES master key of the z/OS system and addressed by their ICSF key label. ICSF does not provide functions to retrieve clear key values of keys stored in the CKDS. If server instances in different z/OS systems belong to the same realm, these systems need to share the CKDS in order to be able to share the encryption keys.

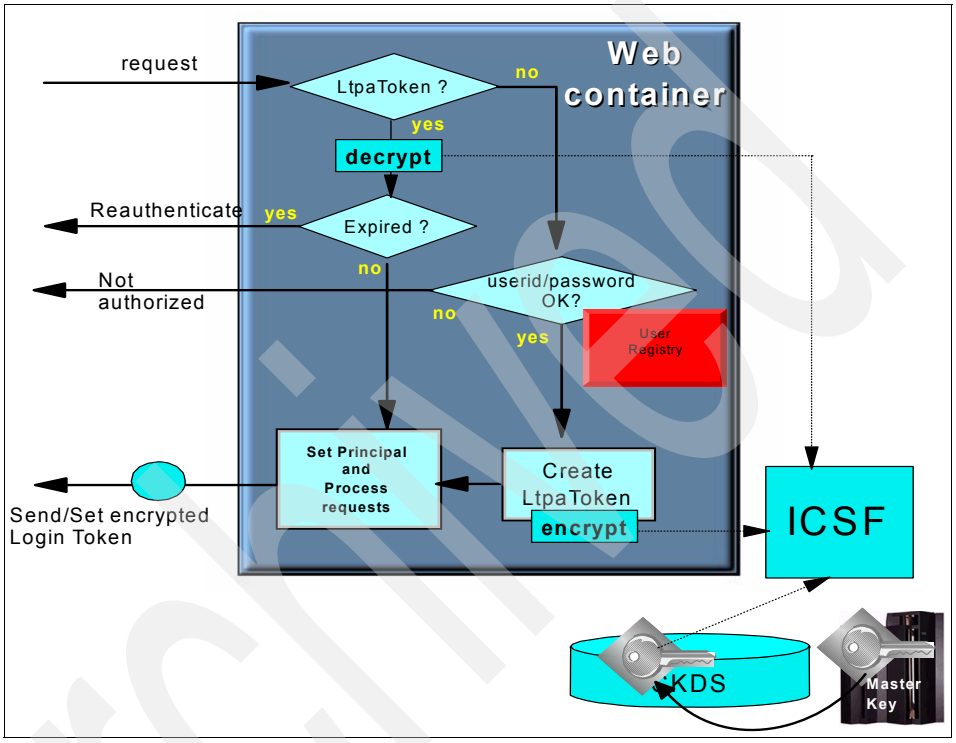


Figure 16-21 ICSF authentication mechanism

This mechanism is exclusive to z/OS so it can provide single sign-on abilities only for a set of WebSphere Application Server for z/OS and OS/390 V4.0.1 domains and WebSphere Application Server for z/OS Version 5 cells.

Note: Because ICSF mechanism relies on the ISCF token cookie (LTPA), it might happen during tests that a browser already has a token from a previous session. As a result, before and between ICSF tests, clean your cookies from your browser. Otherwise, you might have a `SECJ4034I Token Login failed` error because of an old cookie.

As shown on Figure 16-22, the ICSF authentication mechanism configuration requires that you give the name of the ICSF key that the ICSF administrator has generated and the time period in minutes at which an ICSF token will expire.

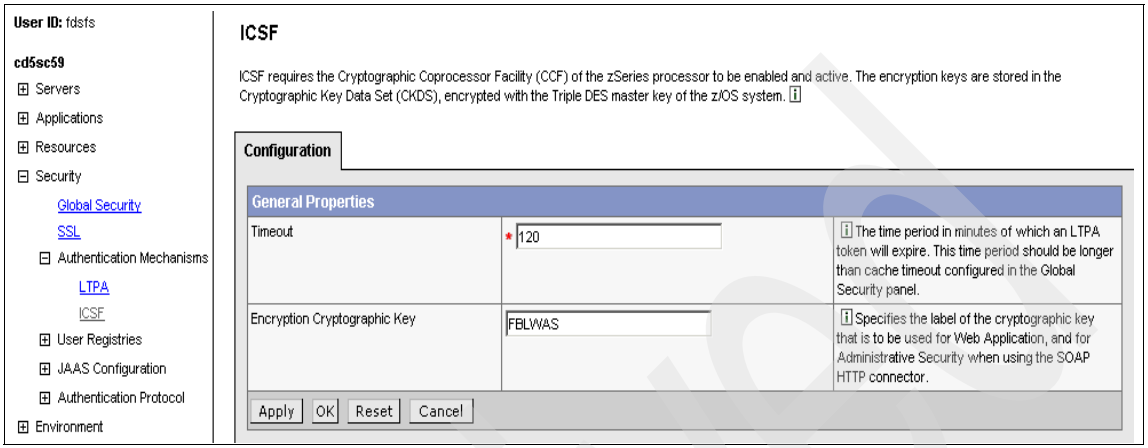


Figure 16-22 ICSF configuration page

To activate ICSF, you must select **ICSF** as the authentication mechanism on the Global Security page, as shown in Figure 16-23.

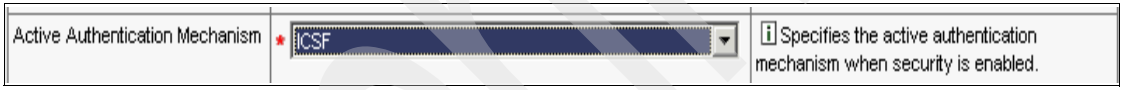


Figure 16-23 Enabling ICSF for the cell

Why should I enable this function?: You plan to run in a Network Deployment controlled application server. You need single sign-on capabilities with WebSphere Application Server for z/OS and OS/390 V4.0.1 but not with applications on distributed systems. You do not want to suffer an interrupt while the LTPA key is updated on all systems in the LTPA realm. You plan to use trust association interceptor code.

Why might I not enable this function?: You need single sign-on capabilities with distributed platform. ICSF is not enabled on the machine.

16.2.8 HTTP-based single sign-on

This section describes HTTP-based single sign-on.

Description

For single sign-on users, provide their credentials (user identity, password, or token) once within a session. These credentials are available to all enterprise applications for which single sign-on was enabled without prompting the user to reenter user ID and password.

The goal is for an enterprise to be able to have one network identity per user, allowing the centralized management of the various roles the user might have in different applications, so correct rules can be applied without duplication of either user data and without requiring multiple identities for the user.

In practice, network identity management is not yet mature enough within most enterprises to achieve a single user registry, particularly when established applications are exposed to the Web. Different application servers have typically implemented their own security or used the operating environment security of the platform on which they have been deployed. The next task is to authenticate a user and provide, within the current session, a credential that can be passed through to and understood by each application.

IBM has previously developed the Lightweight Third Party Authentication, (LTPA) mechanism for enabling single sign-on between various application servers on non-z/OS platforms. In WebSphere Application Server for z/OS and OS/390 V4.0.1 there is an equivalent mechanism that uses cookies encrypted using keys managed by the Integrated Cryptographic Service Facility (ICSF).

Remember: Because SWAM does not provide any forwardable token mechanism, it does not support single sign-on.

For both the LTPA and ICSF mechanisms, a token (the transient cookie) is generated by the authenticating server. The cookie is encrypted using a key which must be shared among all single sign-on participating servers, and contains user authentication information, the network domain in which it is valid for single sign-on, and an expiry time.

The token is issued to the Web user in a cookie called a *transient* cookie. This means that the cookie resides in the browser memory, is not stored on the user's computer system, and expires when the user closes the browser. This cookie is easily recognized by its name: *LtpaToken* for both LTPA and ICSF.

Tivoli Access Manager, with its reverse proxy security server, WebSEAL, provides a mechanism for single sign-on which can be used in conjunction with LTPA and trust association interceptor (TAI) provided as the trust association mechanism (TAM). Tivoli Access Manager can integrate most back-end application servers by using its global sign-on mechanism that can work with many third-party user registries. In addition, it is possible to extend the TAM schema to include established user identities and passwords.

The requirements for enabling single sign-on are as follows:

- All single sign-on participating servers have to use the same user registry.
- All single sign-on participating servers must be in the same DNS domain (cookies are issued with a domain name and do not work in a domain other than the one for which it was issued).
- All URL requests must use domain names. No IP addresses or host names are allowed because this causes the cookie to function incorrectly.
- The browser must be configured to accept cookies.
- Server time and time zone must be correct. The single sign-on token expiration time is absolute.

All servers participating in the single sign-on scenario must be configured to share their LTPA or ICSF keys.

Enablement

- In order to support single sign-on between two application servers, you must have the same encryption keys on both servers.
- With the ICSF authentication mechanism, this is done by using the same key label on all of the servers and sharing the ICSF CKDS.
- With LTPA, you need to export the LTPA key from one server and import it onto the other SSO domain servers.

When all of the SSO domain application servers share the same key, you have to go to the SSO menu available from the bottom of LTPA and ICSF configuration pages. This prompts you to enter the fields shown in Figure 16-24 on page 539.

The “Enabled” option means that the application server generates LTPA tokens. If it is not selected, it will still accept tokens generated by other servers.

Note: When you disable SSO, you can no longer log on to the administrative console because the administrative console is protected by form-based authentication, and this requires SSO to be enabled when not using the SWAM WebSphere authentication mechanism. See 15.7.5, “Disabling global security” on page 491.

Figure 16-24 Single sign-on configuration

Validation

We were able to validate single sign-on with base Application Servers, for both basic authentication and form-based authentication.

Test case SSO1: Single sign-on

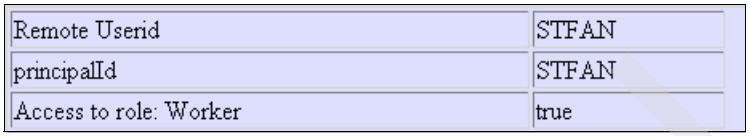
We tested versus in a base Application Server with direct requests to the HTTP transport handler.

The user registry was RACF.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet principal on the first page of the application. For all of the tests, we logged on in a

first server (Server 1 in the test case table) and looked at the Servlet Security Info table to check that the principal was the user ID entered at the security prompt. Figure 16-25 shows the Servlet Security Info table in SWIPE.



Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 16-25 Servlet Security Info table from SWIPE

Then, we logged on with the same URL onto a different server (Server 2 in the test case table) and checked whether we were prompted again to enter a user ID, and if not, we then checked that the principal in the Servlet Security Info table was the same as appeared on the first server. Single sign-on was effective when these last two conditions were met:

- To test SSO with basic authentication with SWIPE, we accessed on a server the EJBCaller servlet in the IBMEBizWeb Web application using the `/IBMEBizWeb/secure/EJBCaller` URL. Then we accessed the same URL on another server, ensuring that there was no additional realm prompt and the principal was the same as entered in the first server.
- To test with form-based authentication with SWIPE, we accessed on a server the formSnoop servlet in the IBMEBizWeb Web application using the `/IBMForms/secure/formSnoop` URL. Then we accessed the same URL on another server, ensuring that there was no additional login form prompt and the principal was the same as entered in the first server.

We expected that configuring different keys in the server should avoid SSO.

Table 16-4 Single sign-on test case 1 (SSO1)

SSO1 config: WebSphere base, transport handler, RACF, HTTP							
Configuration	**Authc. method**	**Authc. mech.**	**Input**			**Output**	
			User ID	**Server 1**	**Server 2**	**SSO effective**	**Comments**
Different keys: SSO should not apply	Basic	SWAM	Not supported				
	Basic	LTPA	stfan	Key1	Key2	No	
	Basic	ICSF	stfan	ICSFKey1	ICSFKey2	No	
	FBL	SWAM	Not supported				
	FBL	LTPA	stfan	Key1	Key2	No	
	FBL	ICSF	stfan	ICSFKey1	ICSFKey2	not tested	
Same key: SSO should apply	Basic	SWAM	Not supported				
	Basic	LTPA	stfan	Key1	Key1	Yes	
	Basic	ICSF	stfan	ICSFKey1	ICSFKey1	Yes	
	FBL	SWAM	Not supported				
	FBL	LTPA	stfan	Key1	Key1	Yes	
	FBL	ICSF	stfan	ICSFKey1	ICSFKey1	Yes	

16.3 Web container authorization

Authorization involves checking whether an authenticated user is granted access to a resource.

WebSphere implements the roles-based security from the J2EE Specification. A security role is a logical grouping of principals. Access to a specific part of the application is granted based on the role, which can be mapped during the development or deployment phase to a specific user registry entry or authorization table. It gives a certain level of transparency to the application development process. The developer needs not to bother about the different user privileges that can be defined for the application.

16.3.1 Web resource protection

Providing an authentication mechanism for global application security does not provide the mechanisms to control access to the Web resources. You have to define a security constraint with an authorization constraint and Web resource collection in the deployment descriptor.

Security constraints declare how the content of the application is protected. For a given security constraint, three things should be defined:

- One or more Web resources that define actual application components that are to be protected by the security constraint. Web resource is a set of URL patterns and HTTP methods in those resources. All requests matched with the pattern defined for a given Web resource are subject to a security constraint.
- An authorization constraint that defines roles which will be provided access to the Web resources existing within the security constraint. An authorization constraint is a set of roles that the user must be granted in order to have access to a Web resource collection existing within a security constraint. In order to have access to the Web resource, the user should be granted at least one of the roles that are defined within the Authorization constraint.
- User Data constraint indicates the transport layer security setting for client/server communication in order to satisfy given security constraints. This setting should guarantee either content integrity (preventing tampering in transit) or confidentiality (preventing reading data during transfer). User Data constraint can override standard security settings for the application. For example, access to some functions of the application might require just basic login using a user ID and password, and at the same time some functions might require a higher level of protection. User Data constraint allows an application deployer to introduce such protection.

The application assembler defines *security-constraint* elements to specify which roles are authorized to each method. The security-constraint starts with a *web-resource-collection* element that defines one or more *url-patterns* and *http-methods* to which the constraint applies.

The security-constraint also contains an *auth-constraint,* which lists one or more role names that are to be authorized to access the URLs and methods listed in the web-resource collections.

Finally, the security-constraint includes a *user-data-constraint* element to specify whether data flowing between the client and the container must be protected.

Note: *Transport guarantee* defines how the communication between the client and the server is to be protected. There are three options to choose from:

- None: No constraint indicates that the application does not require any transport guarantee.
- Integral: This ensures that data cannot be changed in transit. In practice, this means that a request must be transmitted over an SSL encrypted channel.
- Confidential: This ensures that data cannot be viewed in transit. In practice, this means that the request must be transmitted over an SSL encrypted channel.

Example 16-5 shows an example of a security-constraint as it appears in web.xml. files.

Example 16-5 Security-constraint element

```
<security-constraint>
<web-resource-collection>
    <web-resource-name>WAS Forms Example</web-resource-name>
    <description>Forms Based Authentication</description>
    <url-pattern>/secure/*</url-pattern>
    <url-pattern>/ZosAndWasSecure/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
<auth-constraint>
    <description />
    <role-name>WasFormUser</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Web resource protection definition with WebSphere Studio

To enable security constrains with WebSphere Studio Application Developer, you need to select the J2EE perspective and select the Web project you want to secure. This opens the Web deployment descriptor configuration menu. The Security Constraint tab in the Security tab is dedicated to security constraints definitions. Figure 16-26 on page 544 shows such a page.

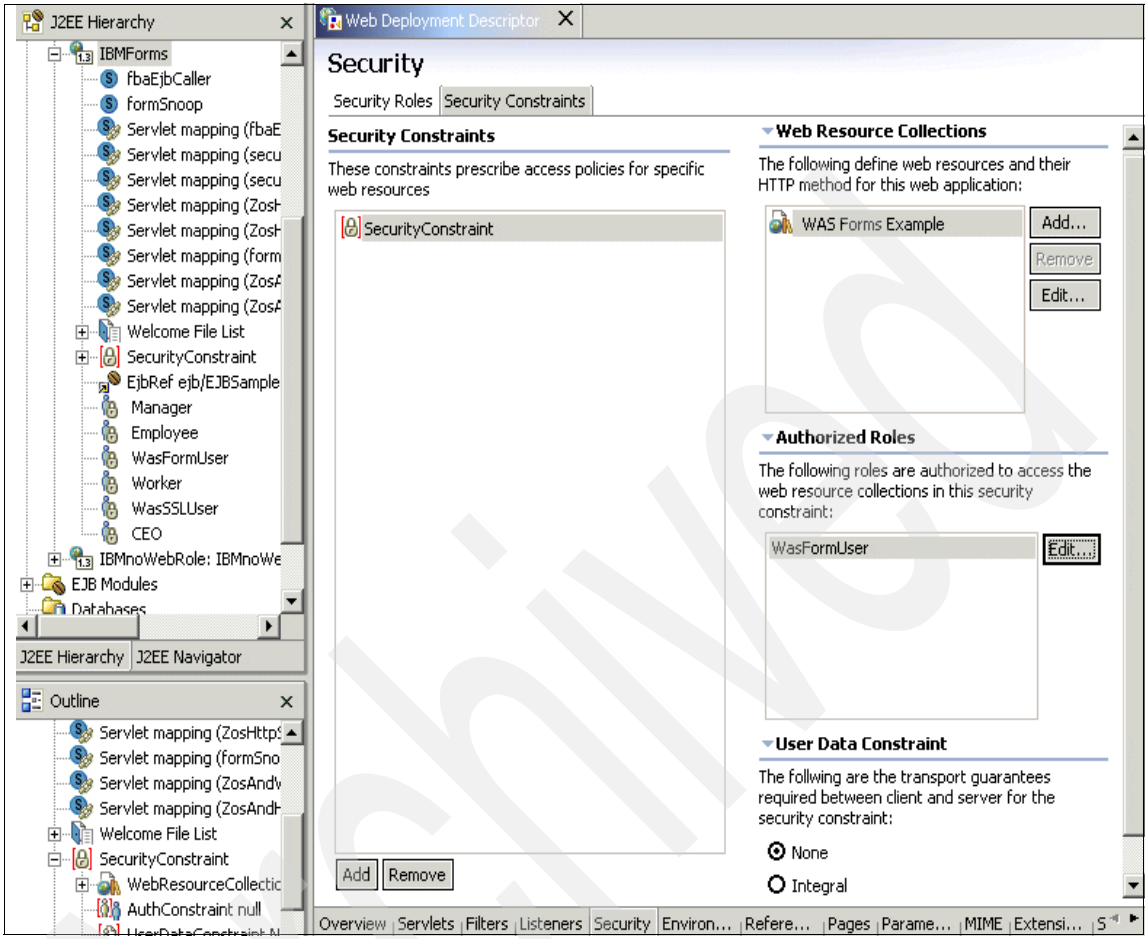


Figure 16-26 Configuring security constraints with WebSphere Studio Application Developer

Select **WAS Forms Example** in “Web Resource Selection,” and then click **Edit**.

Figure 16-27 WebSphere Studio Application Developer security constraint method

Web resource protection definition with AAT

To enable security constrains with the AAT, you need to expand the Web application on which you want your security constraints to apply. Then select the Security Constraints menu and right-click and select **New**. Then enter a name for the logical constraint, the roles associated with it, and the transport guarantee that applies. Such a page is printed in Figure 16-28 on page 546.

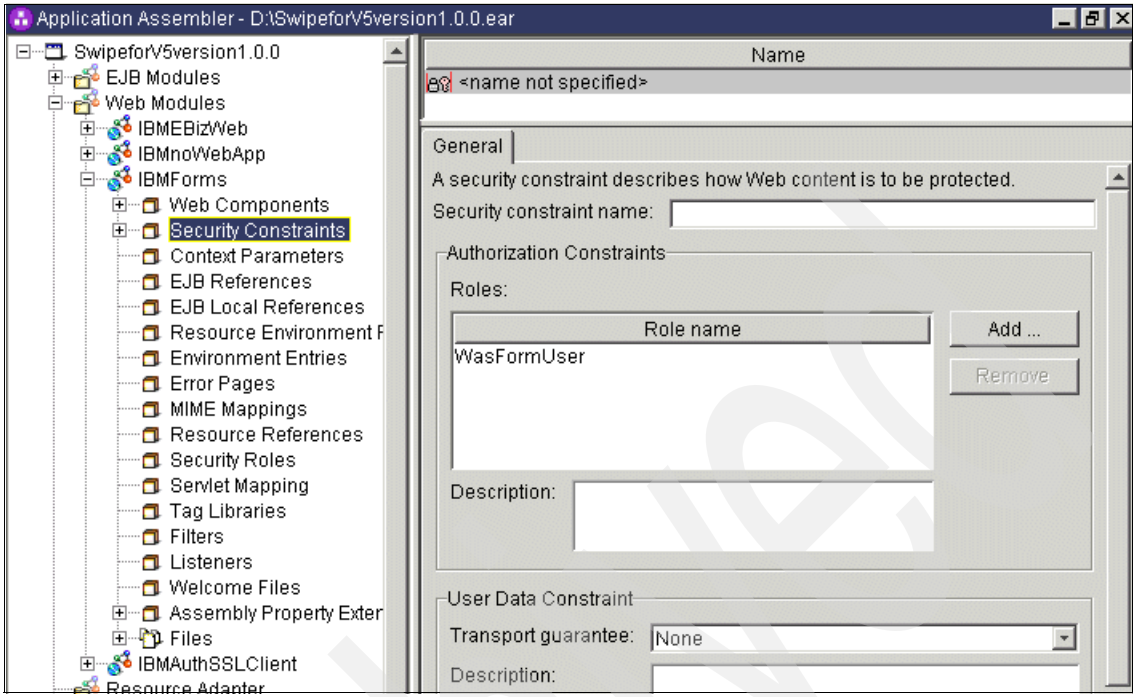


Figure 16-28 Security constraint definition with the AAT

After the security constraint first page has been completed, it appears in the list of the Security Constraints folder and you can expand it to enter the Web Resource Collection on which it applies. The sample of a Web Resource Collection page is in Figure 16-29 on page 547.

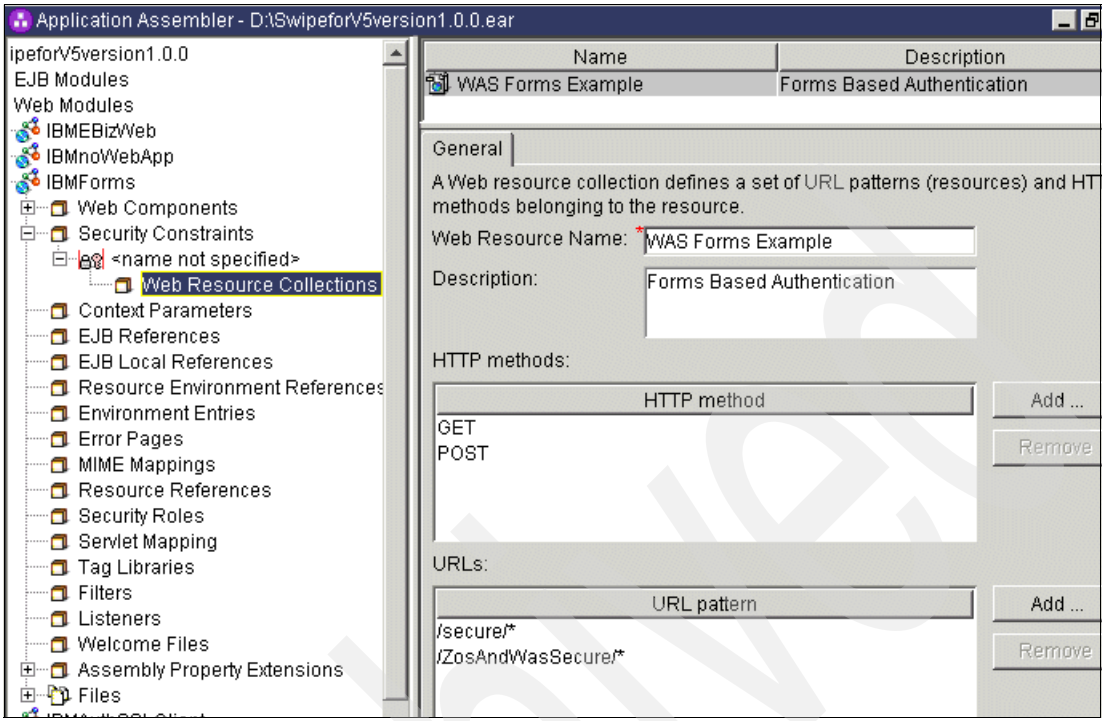


Figure 16-29 Web resource collections in the AAT

WebSphere Web container authorization implementation

A WebSphere Application Server for z/OS V5 can use three different places to determine if a user ID is authorized in a role:

- The local SAF registry when the local OS registry is selected and the local OS custom variable com.ibm.security.SAF.authorization is set to true. In this case, EJBROLE and GEJBROLE profiles need to be created.
- Binding, extensions, and deployment descriptors located in the application's .ear file and created either under WebSphere Studio Application Developer or the AAT. This applies when the user registry is not the OS registry or when the registry is the OS registry but the local OS custom variable com.ibm.security.SAF.authorization is set to false. This is controlled by the application configuration check box Use Metadata From Binaries, which needs to be selected.

- Binding, extensions, and deployment descriptors created during the application installation in the “Map security roles to users/groups” task. This applies when the user registry is not the OS registry or when the registry is the OS registry but the local OS custom variable com.ibm.security.SAF.authorization is set to false. This is controlled by the application configuration check box “Use Metadata From Binaries”, which needs *not* to be selected.

Validation

We have been able to validate the security constraint authorization mechanism with base Application Servers for the basic authentication method and ICSF authentication mechanism.

Test case WAT1: security constraint authorization

We tested the security constraint authorization in a base Application Server with direct requests to the HTTP transport handler.

The user registries were RACF and the FileRegistrySample.

How SWIPE validates the security context

SWIPE validates the security context for this test case by enforcing the Employee role to access the `/IBMEBizWeb/secure/EJBCaller` URL.

We created two users:

- USREMP was authorized in the Employee role.
- USRNOEMP was not authorized in the Employee role.

The selection of the authorization was controlled by the RACF user registry custom variable com.ibm.security.SAF.authorization.

When the USRNOEMP login was rejected, we verified the following message in the MVS log:

```
BBOO0220E SECJ0129E: Authorization failed for USRNOEMP while invoking GET on
default_host:/IBMEBizWeb/secure/EJBCaller, Authorization failed, Not granted
any of the required roles: Employee
```

When the roles were protected by RACF, we also checked for the ICH408I message to be issued:

```
ICH408I USER(USRNOEMP) GROUP(SYS1  ) NAME(SWIPE USER        )
  Employee CL(EJBROLE)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

Table 16-5 Web container authorization Test Case 1 (WAT1)

WAT1 config: WebSphere base, transport handler, HTTP, basic authentication, ICSF					
Configuration	**User registry**	**Authorization repository**	**Input**	**Output**	
			User ID	**Authorization**	**Comments**
User ID not in role	RACF	RACF	usrnoemp	no	
	RACF	Authorization table	usrnoemp	no	
User ID in role	RACF	RACF	usremp	yes	
	RACF	Authorization table	usremp	yes	

16.3.2 Web component protection with role references

The Servlet API provides interfaces to programmers in order to verify whether a principal is authorized to a specific role, but during the development phase of the application, the actual role names for security constraints might not be known to the groups of developers. However, the actual role names in a deployed runtime environment might not be known until the Web application is ready and assembled into the .ear file. Therefore, the role names used during development are considered to be “logical roles”. These logical roles are then mapped by the application deployer into the actual runtime roles during the application assembly and deployment phase.

Security role references provide a level of indirection to isolate roles used during development and actual runtime roles. They link the names of the roles used in the module to the corresponding name of the role in the encompassing application.

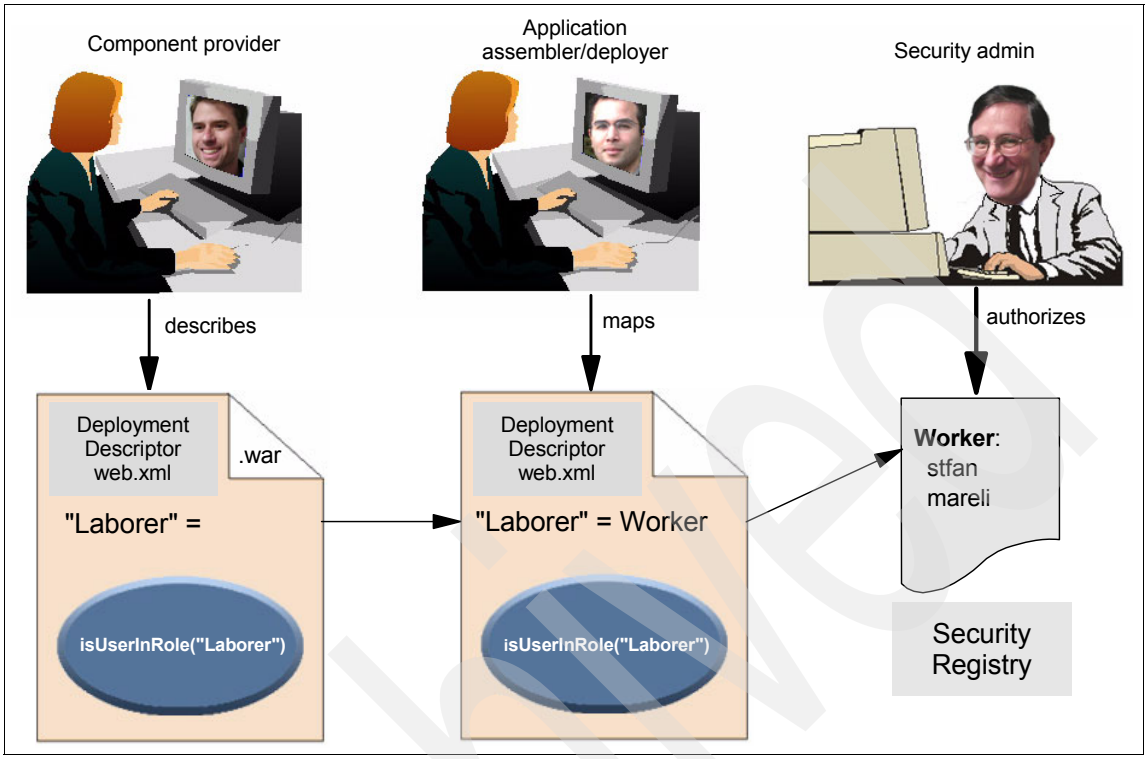


Figure 16-30 Security role references

The Web container supports two APIs for programmatic authorization. They belong to the javax.servlet.http.HttpServletRequest interface. An application programmer can use getUserPrincipal() to determine the identity of the authenticated user. The application logic can then make authorization decisions based on this identity. This method takes no arguments and returns a java.security.Principal object.

isUserInRole() takes a role name in the form of a java.lang.String as an argument and returns a boolean result. The result will be true if the current authenticated user has been authorized to the role name passed on the call. This is useful when a Web component method can be accessed by users in more than one role. It gives the application programmer a way to determine if the current user has been authorized to a specific role of the one or more that may be authorized to access the Web component method.

An application programmer who uses isUserInRole() might define the role names separately in the deployment descriptor file. This is done through the deployment descriptor's security-role-ref element. This element is contained within the *servlet*

element, so that the roles defined in one servlet can be independent of those defined in other servlets. Example 16-6 shows a security-role-ref element for the secureEJBCaller servlet, defining a role name of Laborer (the name used in the application for isUserInRole) and linking it to a role of Worker as used in the remainder of the application.

Example 16-6 Security-role-ref example

```
<servlet>
  <servlet-name>secureEJBCaller</servlet-name>
  <display-name>secureEJBCaller</display-name>
  <servlet-class>ejbTester.secureEJBCaller</servlet-class>
<security-role-ref>
    <role-name>Laborer</role-name>
    <role-link>Worker</role-link>
  </security-role-ref>
</servlet>
```

If the application deployment descriptor does not contain a security-role-ref declaration, the Web container associates the name used in the application for isUserInRole with a role-link of the same name.

Defining role reference with WebSphere Studio Application Developer

To create role links with WebSphere Studio Application Developer, you must first define a role with the same name as the role-name (the name used in the application for isUserInRole) in the Security tab of the Web application (where roles defined for security constraints purposes are defined).

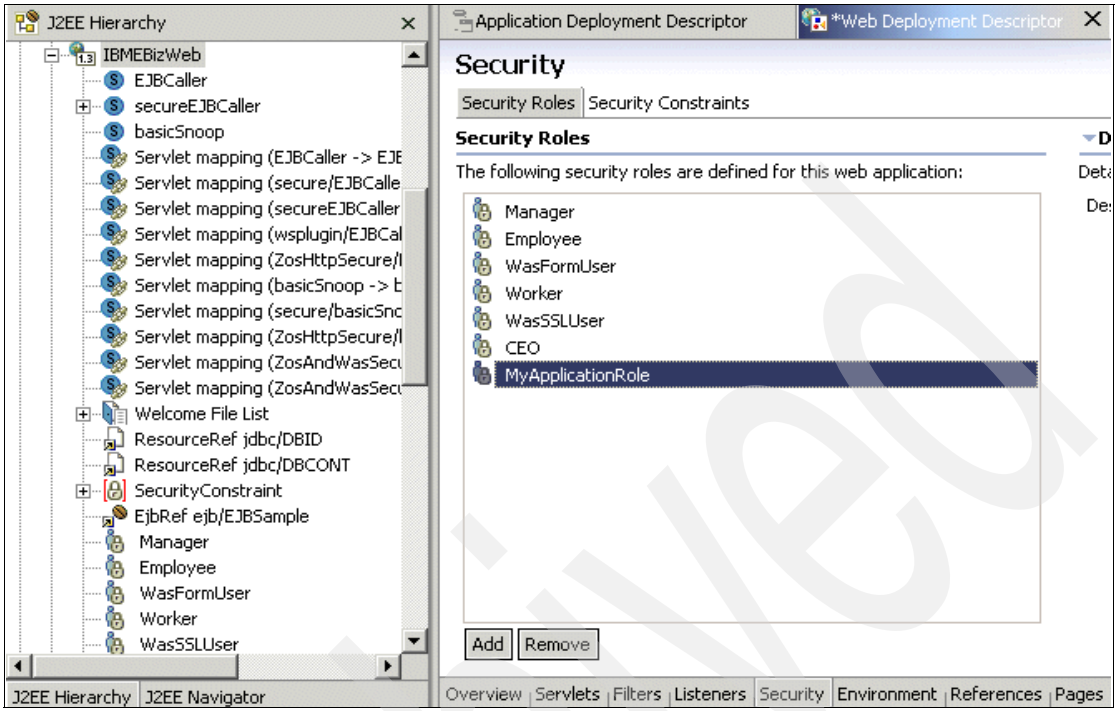


Figure 16-31 Security role definitions in WebSphere Studio Application Developer

Then, go to the servlet tab and add this role in the Authorized Roles box. This will create a <security-role-ref> block with <role-name> and <role-link> filled with the role-name. We did not find any way inside WebSphere Studio Application Developer to define a mapping with different <role-name> and <role-link>. As a result, if the deployer wants to map the <role-name> with a different <role-link> value, he will use the AAT (WebSphere installation does not provide the ability to remap them).

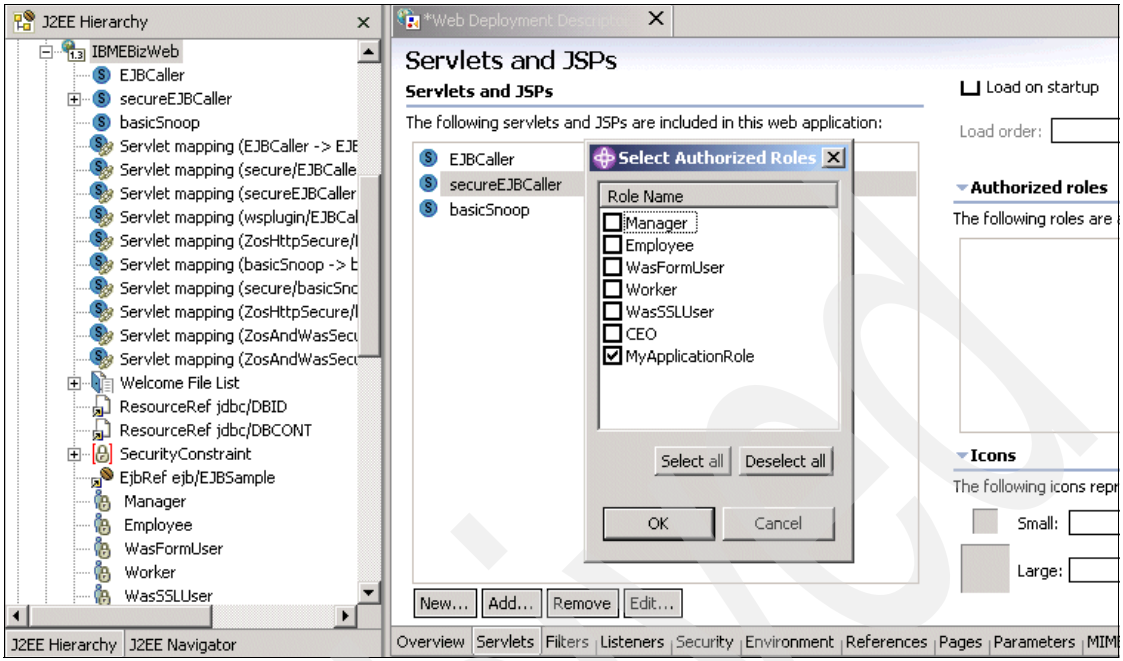


Figure 16-32 Authorization role selection in WebSphere Studio Application Developer

Note: The Authorized Roles box of WebSphere Studio Application Developer is subtitled with "The following roles are authorized to access this servlet." This is not exactly true. It should read "The access to the following roles can be checked by this servlet code."

Defining role reference with the AAT

To create role links with AAT, you need to first declare the role name that will be used by the OS registry of WebSphere (the role-link).To do so, open the Web module to change, select the **Security role** menu, right-click and enter the role-link as it appears in Figure 16-33 on page 554.

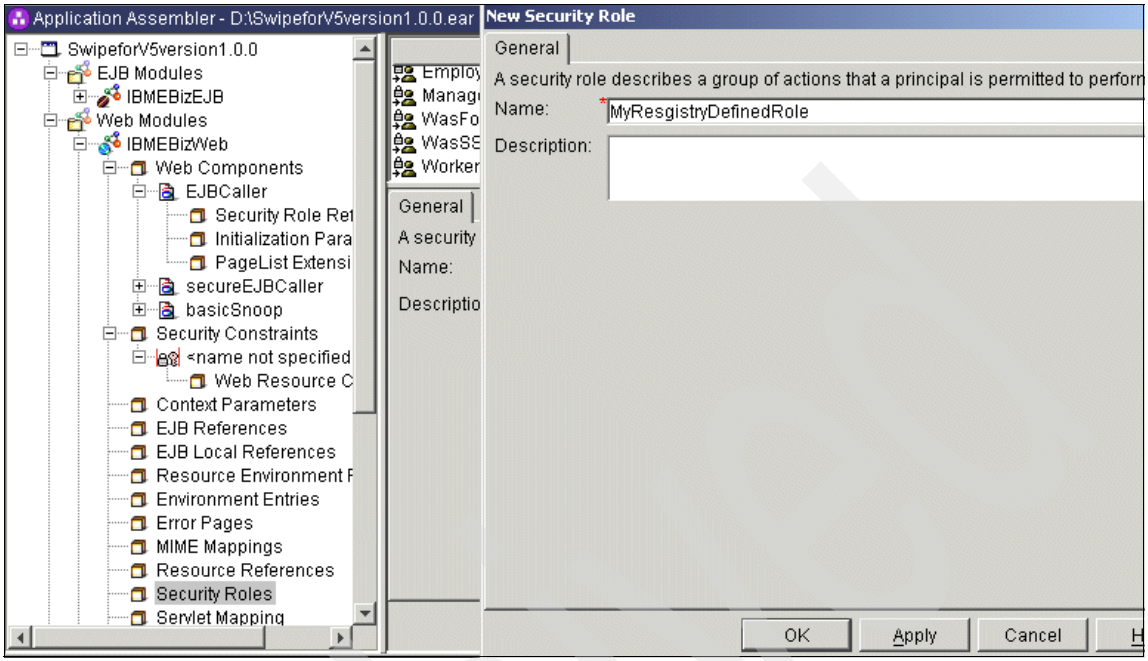


Figure 16-33 Security role definition in AAT

Then, you can expand the **Web Components** folder, expand the servlet that uses the role reference, and after right-clicking the **Security Role Reference** item, you can enter the role name value in the name field and select from the link list the ejb-link that you will implement in WebSphere. See Figure 16-34 on page 555.

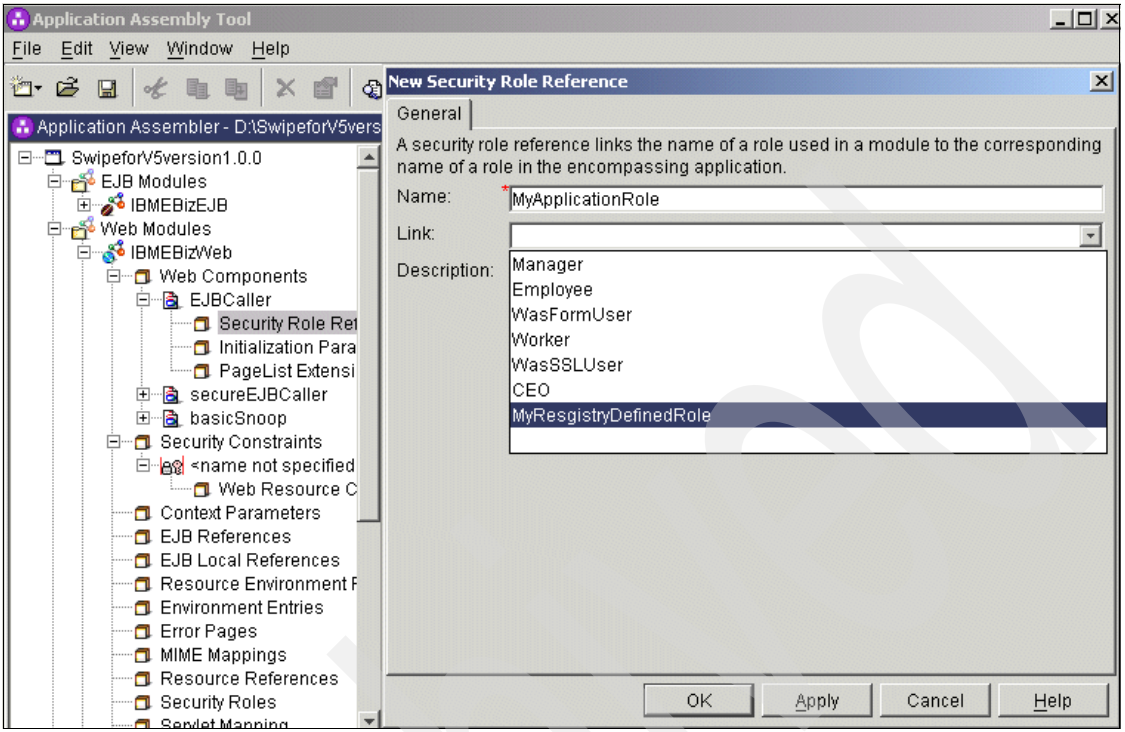


Figure 16-34 Defining role references in AAT

Test case RLK1: Role link

We tested the role links of a base Application Server with direct requests to the HTTP transport handler.

The LDAP registry was an IBM Directory Served V5.1 hosted on a Windows 2000 Intel-based machine.

The custom user registry was the FileRegistrySample as delivered with WebSphere.

Tip: In addition to the role mapping possibilities within the deployment descriptor, WebSphere Application Server for z/OS and OS/390 can make use of the role grouping capabilities of the SAF class GEJBROLE. This is described in “RACF grouping class GEJBROLE” on page 352.

Table 16-6 Role link authorization test case 1 (RLK1)

SSL1 config: WebSphere base, transport handler, basic authentication							
Registry	**(eg SSL)**	**Authv. mech.**	**Input**			**Output**	
			User ID	**RunAs**	**Map-ping**	**Flowed Principal**	**In role Worker**
RACF	No	SWAM	stfan	Caller	none	stfan	yes
	No	LTPA	stfan	Caller	none	stfan	yes
	No	ICSF	stfan	Caller	none	stfan	yes

16.4 Web container identity delegation and propagation

Note: The ability to associate a run-as role with a servlet was introduced with the Servlet 2.3 specification and is supported in WebSphere Application Server V5.

The propagation of security identity from one object to another is supported both in the Web container and the EJB container. The security identity actually propagated depends on the delegation mode or run-as setting, as described in 3.2.4, “RunAs versus run-as: Identity propagation” on page 52.

By default, the run-as is Caller, so the identity of the current thread will be used for any authorization checks that occur when the servlet tries to invoke a method on an EJB. In this way, all EJB methods in the calling chain would see the same principal if they were to call getCallerPrincipal(). Occasionally, however, it is desirable for one servlet to call an EJB with a previously defined identity, such as one that is a member of a specific role.

The run-as mapping is more restrictive in servlets than in EJBs. The only run-as that is permitted is the run-as role, although if you do not set a role the default run-as will be Caller. The servlet is declared to run as a “role” that the application deployer will associate with a registry user ID. After the user ID is authenticated and authorized, the request runs with the run-as role whether the user ID is authorized or not to the run-as role. The run-as role applies also to unauthenticated users.

In the Web container, after authentication has occurred, an authorization check will take place if a role has been associated with the security constraint governing security for the Web components. Associating a role with a security constraint

creates an authorization constraint in the web.xml file. The authenticated identity must be in that role before it can run the servlet.

Assuming that the authenticated identity can run the servlet, the servlet might then try to invoke an EJB (let us call it EJB1), or the servlet might contain code to get a connection from a J2CA connection factory and call a program or access data in an EIS. If there is no run-as role set for the servlet, the default is run-as (Caller). This means that the caller (that is, the authenticated identity) will be used if an authorization check must be performed before invoking the methods on EJB1.

In addition, a unique feature of WebSphere Application Server for z/OS V5 is the ability to propagate the identity of the current thread to the J2CA resource adapter. If the servlet uses a J2CA connector to an EIS system and a local-mode, container-managed connection, the run-as identity of servlet will be propagated to the connector. (This assumes that no container-managed JAAS authentication alias has been defined for the connection factory, because the JAAS alias will be used rather than the current thread identity if it is defined.)

If a servlet has a run-as role set and tries to invoke a method on EJB1 that has a Method Permission defined, the user ID associated with the servlet's role must be in the role that is authorized to the methods of EJB1.

Using the AAT, you set a run-as for a servlet on the Security tab, as shown in Figure 16-35.

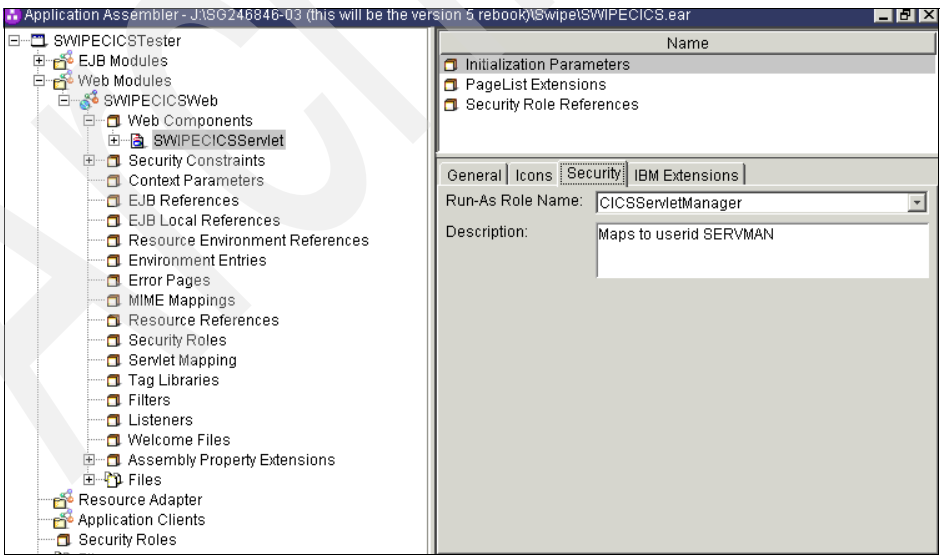


Figure 16-35 Setting Run-As Role for a servlet using the AAT

Important: A servlet bean configured to run as a member of a given security role actually executes using the identity of the caller. It is only when calling methods in EJBs that the run-as mode applies. These methods are called using the delegated identity.

16.4.1 WebSphere role to user implementation

A WebSphere application server can use three different places to determine the user association with a role:

- The local SAF registry when the local OS registry is selected and the local OS custom variable com.ibm.security.SAF.delegation is set to true. In this case, EJBROLE or GEJBROLE profiles need to be modified to contain a RACF user ID in the APPLDATA segment of the profile.
- Binding, extensions, and deployment descriptors located in the application's .ear file and created either under WebSphere Studio Application Developer or the AAT. This applies when the user registry is not the OS registry or when the registry is the OS registry but the local OS custom variable com.ibm.security.SAF.delegation is set to false. This is controlled by the application configuration check box Use Metadata From Binaries, which needs to be select.
- Binding, extensions, and deployment descriptors created during the application installation in the “Map security roles to users/groups” task. This applies when the user registry is not the OS registry or when the registry is the OS registry but the local OS custom variable com.ibm.security.SAF.delegation is set to false. This is controlled by the application configuration check box Use Metadata From Binaries, which needs *not* to be selected.

17

Security integration with the WebSphere HTTP plug-in

This chapter describes the concepts of *demilitarized zone* and *network security*. It lists the various flavors that you can implement. It focusses on the WebSphere Application Server Web plug-in and how to implement it together with WebSphere security.

17.1 Multi-tier topologies and DMZ

WebSphere Application Server security is only a part of the security for WebSphere applications. WebSphere servers are implemented inside physical networks and they have to fit with network architectures for security.

The goal of most network architectures for security is to isolate and filter the different logical component in different network zones. As a result, each zone can be separately controlled and user requests can be validated when passing from one zone to another.

This section describes the basis for secure network configurations, develops the concept of Demilitarized Zone (DMZ) and illustrates the more common implementations of WebSphere front-end network topologies.

17.1.1 Network security

While the authentication and authorization engine is a key component of the security infrastructure, it is not the only one. Other components are equally important:

- Firewalls

  A firewall is an important aspect of creating a secure environment that is connected to the Internet. There are a number of important functions associated with firewalls, too many to document here. Some enterprises have a standard firewall that they use everywhere (with different configurations). This makes management of the infrastructure easier. The danger of one type of firewall is that if a vulnerability is found in that firewall it can potentially be exploited on every firewall, thus affecting the entire enterprise. For this reason many enterprises use different firewalls.

- Network Intrusion Detection

  When people think of security, they often think only of authentication and authorization. There are other important aspects of security as it relates to computers. Network Intrusion Detection is important because it provides the security staff with alerts and audit trails for changes that are made. A complete security environment must include some form of logging, and preferably some alarm mechanism that responds to suspicious behavior.

- Encryption

  Data that is sensitive must be encrypted while it is in transit. This of course has some performance implications and should be done only at appropriate times. It is important to understand the data flow of the application and what is important and what is trivial. Some highly sensitive data might also have to be encrypted while at rest (that is, in a file or database).

- Accountability

  It is important to be able to tie every transaction to a person or at least a group of people responsible for that transaction. It is this accountability that ensures data integrity. When any system activity can be tied to a particular person or group, it becomes easier to pinpoint errors or breaches in the environment and ultimately easier to remove the threat of those breaches in the future.

This is by no means a complete list of things to keep in mind when setting up a security policy. This is a good starting point to help you to figure out what needs to be done. Traditionally, the items listed in the additional security considerations section are well covered in an enterprise. People who secure distributed platforms understand and can secure the network and base access to a platform. The weakness, from an enterprise point of view, comes into play when servers start communicating with servers to manipulate data, especially for customers on the Internet. It is this Internet component and the lack of understanding of the risks of the Web application environment that are a peril to enterprise security.

17.1.2 Putting the pieces together

The discussion so far has revolved around how to authenticate users to the enterprise and what that means to the back-end system. As has been stated before, this is only part of the picture. This section details how these pieces can work together. Because the reverse proxy model is becoming widely accepted in the marketplace, it will be this model that is used for demonstration purposes.

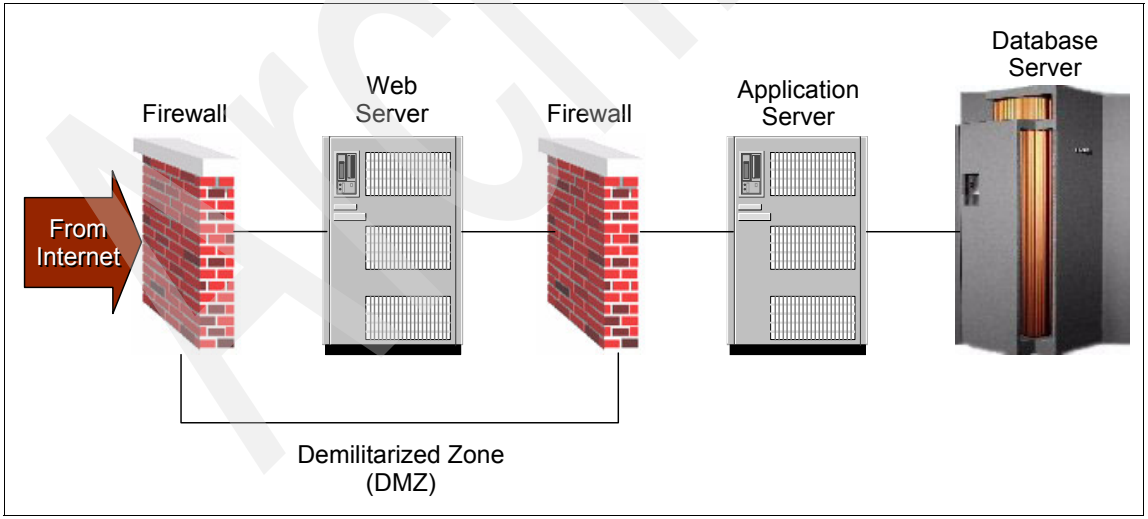


Figure 17-1 Web server in the DMZ

17.1.3 Basic network security setup

As has already been described, firewalls play a pivotal role in securing the enterprise. Figure 17-1 on page 561 shows a standardized setup for security.

In this picture, the Web server is in an isolated network called a Demilitarized Zone (DMZ). This network is isolated from the Internet by a firewall, which is responsible for protecting the Web server (or more likely Web servers) from attacks. This firewall protects the Web server from denial of service attacks as well as viruses and other attacks bent on keeping people from accessing the Web server. This firewall also does Network Address Translation (NAT). NAT hides the addresses in the DMZ from the addresses on the Internet. The Internet can see the Web server at address 202.123.64.22, while the DMZ network has the server at address 10.9.5.3. Because the Internet cannot access the server by its local address, the Web server can easily separate requests that come from the Internet from requests that come from the internal network.

The Web server in the DMZ is protected from the Internet by the firewall. This protection is marginal, however, because it is not meant to keep people out, it is meant to be protected from malicious attacks. The Web server beyond this firewall is protected from these attacks but is still vulnerable to more sophisticated attacks bent on taking over the machine. It is a sacrificial server and therefore is almost always replicated in the DMZ.

The second firewall is a bit more sophisticated. It is responsible for protecting the internal secure network (intranet) from attempts to gain access to critical resources. This firewall only lets verified users through with valid requests. It also allows traffic out from within the intranet using proxy or SOCKS services.

In the intranet, the user base is all known. Either the customers are executing in this environment using accepted identities (valid user IDs), or the internal users are accessing the data. This is not to say that everyone has the same level of authority, but the entrances to this network are more controlled, either by physical access or by network security. The application server runs in this network. It is by its nature a product that provides information and as such must live in a secure environment.

Database and transaction servers also run in this realm and generally must be more rigorously protected. Often server access management revolves around authorization of a particular user ID. While many different users can access the application server, access to the database and transaction servers is generally locked down to a smaller community.

It is within this context that the users must authenticate to determine who they are and what they can access. With most of the authentication and authorization models discussed above, there is generally some sort of compromise of the

second firewall (or inclusion of the application server in the DMZ) to accomplish the authentication. Often you will see the picture shown in Figure 17-2.

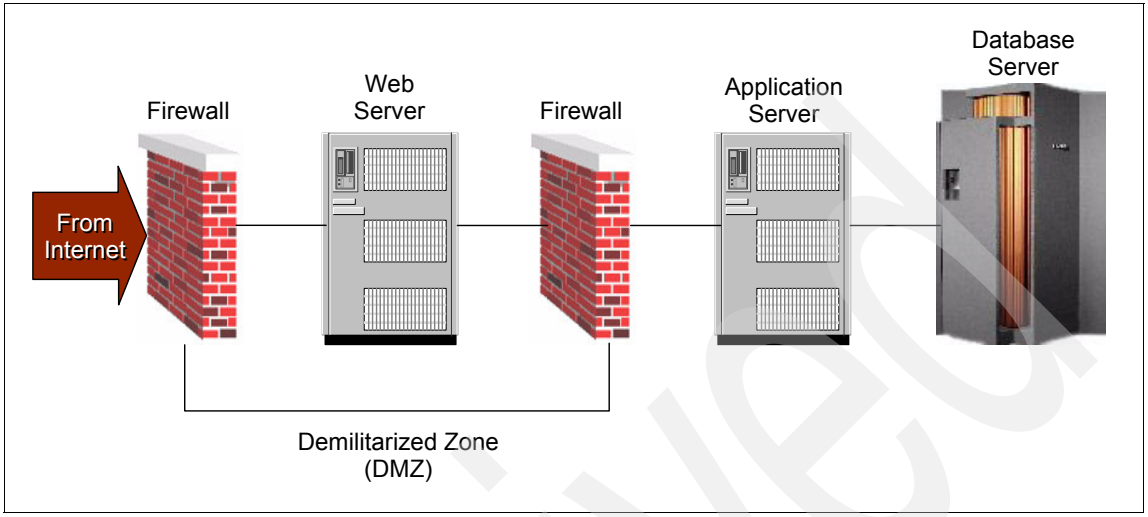


Figure 17-2 Web and application servers in the DMZ

Here the application server is in the DMZ and would do authentication and authorization. Requests for data or transactions would travel through the firewall to the database server in the secure environment.

This picture is deceiving, however. Often the Web server and the application server are peers in this environment. The Web server provides static content while the application server provides dynamic content.

17.1.4 Basic reverse proxy setup

Figure 17-3 on page 564 builds on the basic security setup to show how reverse proxy plays in this environment.

Here the Web server is responsible for the enterprise's static content. If a user wants to run a Web application, the request is redirected to the reverse proxy server, which initiates a secure session with the user (generally using SSL) and prompts the user for identification. Today this is usually in the form of a user ID and password or a digital certificate. The reverse proxy issues a request to the user registry over a network connection that is different from the connection being used by the user's request. This connection generally makes use of some secure protocol (like SSL).

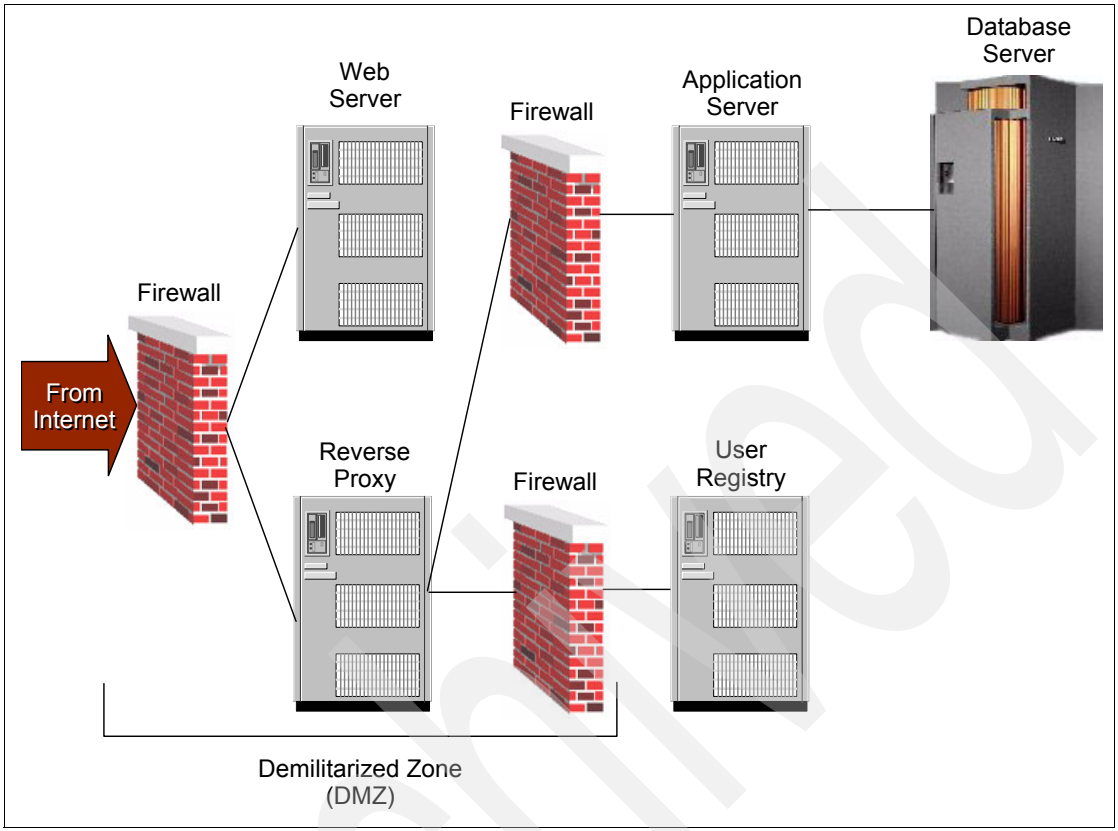


Figure 17-3 Showing a user registry such as LDAP

The user registry is usually an LDAP server that contains the user information, including a user ID and password or a digital certificate. It also contains attributes that an agent running on the application server platform can use. The reverse proxy server sends a request back to the application server with a credential that defines who is making the request. The application server can consult the agent to determine if the user has the authority to perform the attempted action.

In many cases the reverse proxy has the ability to perform some authorization as well. It is normally used to determine whether the user has the authorization to perform the request on the server. The earlier in the process a user is denied access, the more secure the system can be.

This picture is simplified and does not include other servers that will probably be involved in the management of the security policy. Remember, this environment must support auditing as well as servers responsible for maintaining and updating the security policy. In a well-managed security environment, users'

authorization to resources can be modified on the fly, which will be refreshed in the reverse proxy server as well as on servers running the agent (in our case, the application server).

This configuration provides the infrastructure for an enterprise to support digital certificates. While personal digital certificates have not yet become widespread on the Internet, it solves a number of problems that are associated with user ID and password schemes. The major concerns with user ID and passwords are:

- People tend to create trivial passwords. This means that passwords are either easily guessed or with minimal social engineering can be retrieved from the user.
- User ID and password access to most platforms is usually compromised by the fact that passwords must be updated regularly. Most software tools do not recognize that the password has expired with an element to update that password.
- A malicious hacker can spoof a site. Users thinking that they are on the real site will respond to the user ID/password query and unintentionally provide the hacker with the user ID and password.

Digital certificates solve these problems because:

- Having the password means nothing if the requestor does not have the certificate.
- Certificates are generally good for a year or more. This way a user does not have to keep modifying a password for access to a site.
- Certificates are password-protected, but the password is connected to that machine only and does not flow from place to place.
- Because half of the authentication process is based on something that never leaves the computer, a cracker can't spoof the site to steal a user's identity.

17.1.5 A business-to-business variation

The infrastructures that have been examined up to this point have been primarily business-to-customer infrastructures. These contain a single user attempting to access data from a Web site. While this is prevalent in a business-to- business environment where members of one business have a client relationship with another business, many enterprises have more complex relationships within the company, requiring a bit more complex setups.

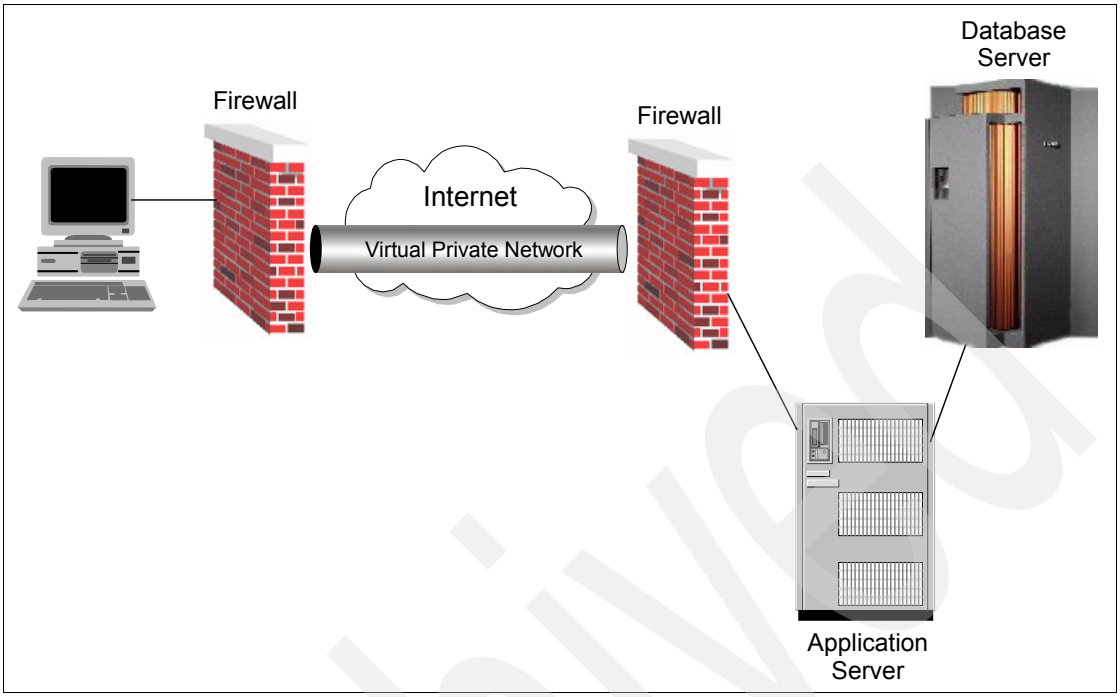


Figure 17-4 Firewall protection

Figure 17-4 shows a method many enterprises use if they have separate sites that need to communicate securely, or between businesses that share data over the Internet.

In this picture, the browser on the left connects directly with the application server over the Internet. In this scenario, the firewalls talk to each other over a Virtual Private Network (a VPN). The “client” firewall communicates with the “server” firewall and the two servers set up an encrypted connection for communication. Because each firewall has a certificate, each end can ensure that it is talking with a trusted partner. The link between the two is encrypted and multiple clients can make use of this VPN. Because the firewalls are responsible for creating and maintaining this link, the clients do not have to go through the effort of encrypting the requests.

This is valuable when an enterprise can trust the data to be in the clear in their internal networks but not over the Internet. This gives the enterprise a performance advantage because the initial setup of a secure session requires a significant amount of processing and communication between partners. In a VPN environment this is unnecessary.

Figure 17-4 on page 566 also shows the application server as responsible for authenticating the user. This does not mean that a reverse proxy model is not appropriate here. Generally the reverse proxy model can still work in a VPN environment, but the encryption requirement disappears. The user repository associated with the reverse proxy could be used in a VPN configuration as well as a standard DMZ environment. The power of the reverse proxy configuration lies in the ability to use the single registry in both environments. A more complex picture that makes use of this could look like Figure 17-5.

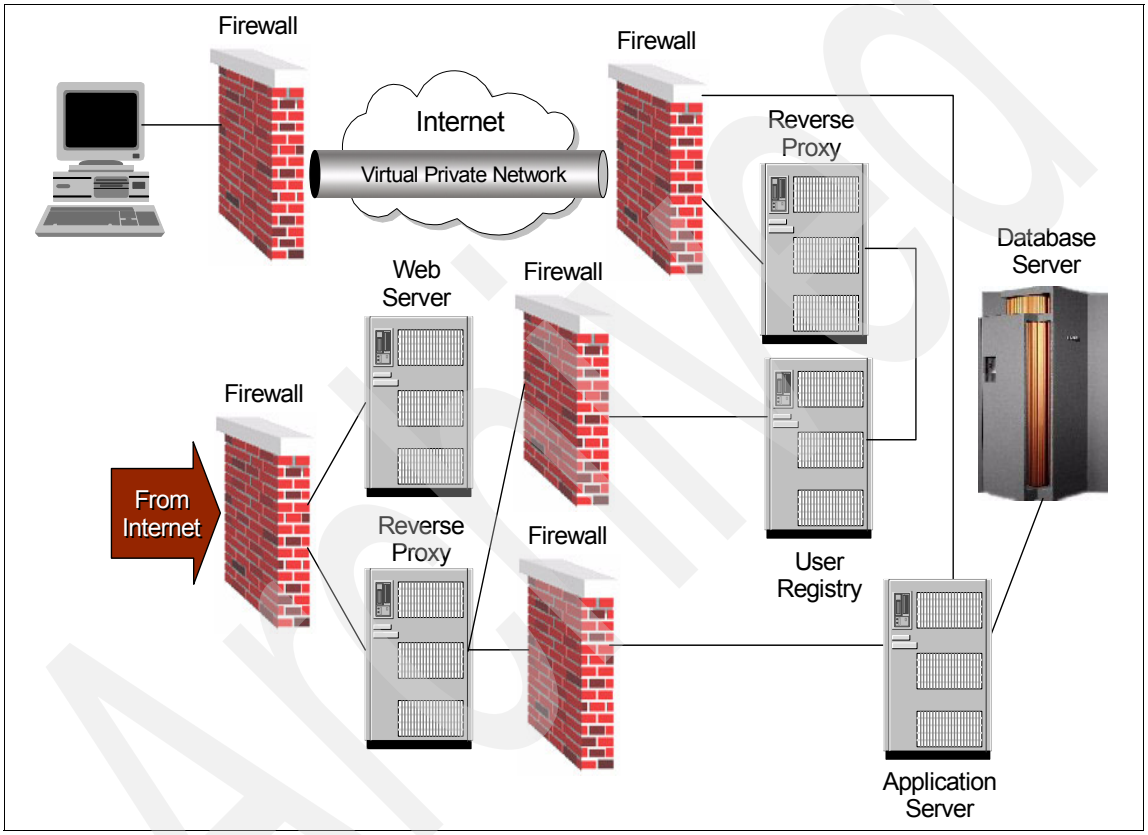


Figure 17-5 Using firewalls and VPN

This graphic shows that a single user registry supports the two reverse proxies. Each reverse proxy relies on the user registry for authenticating users, but the reverse proxy associated with the VPN does not deal with the encryption that the reverse proxy the client on the Internet uses to access the enterprise application server.

All of these pictures show the application server and the database server as separate devices. z/OS can, in fact, perform both functions, resulting in both a

performance benefit and a security benefit. From a performance point of view, there is less CPU and network latency associated with a single platform approach. From a security point of view, there are fewer layers exposed to unauthorized access.

The discussion so far has revolved around how to set up the infrastructure for Web applications, specifically when those applications have been running in a z/OS environment. You will note that every security model assumes an HTTP connection to the z/OS operating system. This was used because this is the most common communication method. While other methods are also used (IIOP and MQ are the most prevalent), they are not as common. IIOP is by its nature a heavier-weight protocol than HTTP, and MQ's asynchronous nature makes it less straightforward in a synchronous environment.

Application servers that run off z/OS will probably use different protocols to communicate with back-end systems such as CICS or DB2. This, of course, requires added software and infrastructure. Running the application server on the same platform as the back-end systems can simplify the connections and build more secure and faster running environments.

17.2 WebSphere and HTTP server plug-ins

Early versions of IBM WebSphere Application Server for z/OS were implemented using the HTTP server address space. In this configuration, WebSphere relied heavily on the HTTP server for Web requests. One of the advantage of this configuration was that a large part of the HTTP protocol process was delegated to the HTTP server. The main back draw was the fact that each server was an isolated Web container.There was no possibility provided by WebSphere to have a single access to all of the WebSphere servers (except to install all applications in the same in which case this server has the complete configuration).

In this section we describe the different flavors of WebSphere plug-ins that run in HTTP server processes separated from the Web container. All of them are based on the principle of a front-end process that reroutes HTTP requests to the Web container. This approach offers a reverse proxy engine for the DMZ, more flexibility for the WebSphere physical implementation, and opportunities for an early workload management. Even if a physical architecture using an HTTP plug-in brings additional CPU cost and usually additional hardware, it has become the most common implementation.

17.2.1 z/OS local redirector plug-in

Prior to fix level 5 (W401500, PTF UQ74160)) of WebSphere Application Server for z/OS and OS/390 V4.0.1, a RMI/IIOP-based plug-in was used in IBM HTTP

Server for z/OS. It basically converted incoming HTTP to IIOP and forwarded the request to the J2EE server's IIOP listener. This plug-in, called *local redirector plug-in* is no longer supported in WebSphere Application Server for z/OS and OS/390 Version 5.0.

17.2.2 WebSphere HTTP plug-in

In order to help traditional HTTP servers to route requests to WebSphere application servers and act as reverse proxies inside a DMZ, WebSphere Application Server for z/OS and OS/390 V4.0.1 fix level 4 introduced the support of WebSphere HTTP plug-in on distributed platforms. The WebSphere HTTP plug-in is code running inside the HTTP server that intercepts HTTP requests and forwards WebSphere only requests to the HTTP handler (see "HTTP transport handler" on page 569). Later, WebSphere Application Server for z/OS and OS/390 V4.0.1 fix level 5 delivered the WebSphere HTTP plug-in for z/OS. It is to be noted that the WebSphere HTTP plug-in for z/OS is more than a simple reverse proxy, as it used to be the decision maker for availability and scalability questions.

If the server processing the request is a reverse proxy server, the HTTP transport handler has to be able to take the request that has been authenticated by the reverse proxy and make use of the identity that has been sent. Note that this is somewhat different from authenticating a request directly from a user. The transport handler must take another server's assertion of a user's identity. This is the job of the trust association interceptor (TAI). This code must map the identity asserted by the other server to a z/OS identity. After this identity has been established, the traditional J2EE security rules can apply to the user.

It is this mapping that allows users to be authenticated off platform and maintain an identity while in WebSphere. Because the identities running in WebSphere do not log on in the traditional sense, they do not need passwords. This allows these user IDs to be restricted for use only in a WebSphere environment.

In the future these enterprise-wide security environments will blur the line between platforms. Requests will be able to pass from platform to platform and rely on enterprise security agents that are resident on the platform. WebSphere will interface with these agents directly, providing a seamless environment regardless of platform.

HTTP transport handler

With WebSphere Application Server 4.0 for z/OS and OS/390, the Web server was completely separated from the application server with an RMI/IIOP local redirector plug-in. While this created a more flexible environment, the performance of servlets and JSPs suffered from the separation. WebSphere 4.01 fixed this problem by introducing the HTTP transport handler. This lightweight

HTTP management code is significantly faster than its Web server predecessor and provides a significant performance improvement over its Web server counterpart. WebSphere Application Server for z/OS and OS/390 Version 5 can only be accessed by the HTTP Handler.

This improvement came at a price, however. Much of the traditional support found in Web servers (for instance, the ability to recognize different browsers) does not exist in the HTTP transport handler. This is because enterprises are relegating the traditional Web request handling to servers closer to the edge of the enterprise. Because the handler can expect that the customer request has been filtered by some other server, it does not need the extra code

WebSphere HTTP plug-in advantages

The new WebSphere HTTP plug-in for z/OS has the following advantages over the servlet redirector plug-in:

- The new plug-in is supported across all of the WebSphere Application Server family. For instance, you can use an WebSphere Application Server plug-in for Linux for zSeries to redirect requests to WebSphere Application Server for z/OS and OS/390 Version 5.
- The HTTP version of the plug-in is much faster than the IIOP implementation.
- HTTP is firewall friendly, that is, most firewalls will allow HTTP to pass through. Also, due to the use of HTTP 1.1, connections between the plug-in and the application server can be reused by multiple requests, which reduces the load on the firewall.
- HTTPS enables us to encrypt messages sent between the Web server and WebSphere using a standard protocol.
- It is much easier to configure the WebSphere HTTP plug-in for z/OS. A single XML file controls the WebSphere HTTP plug-in for z/OS and permits to easily control which part of the configuration will be exposed to the plug-in code.

Although WebSphere can only generate a single cell-wide plug-in xml configuration file, the xml plug-in configuration file can be reworked to include different cells

17.3 WebSphere HTTP plug-in description

The WebSphere HTTP plug-in is a module that runs as part of an HTTP server process.It acts as the gatekeeper between the HTTP server and WebSphere. It decides which HTTP requests should be passed to WebSphere and which ones are for the HTTP server alone to handle. It also acts as an intelligent router with the awareness of which HTTP transport handler is defined for each Web

container, which WebSphere application servers are up and down, and which WebSphere application servers are part of clusters. The WebSphere HTTP plug-in stands as a major component for managing availability and scalability in a cluster environment.

Note: For a complete description of plug-in workload management capabilities, refer to *IBM WebSphere V5.1 Performance, Scalability, and High Availability: WebSphere Handbook Series*, SG24-6198.

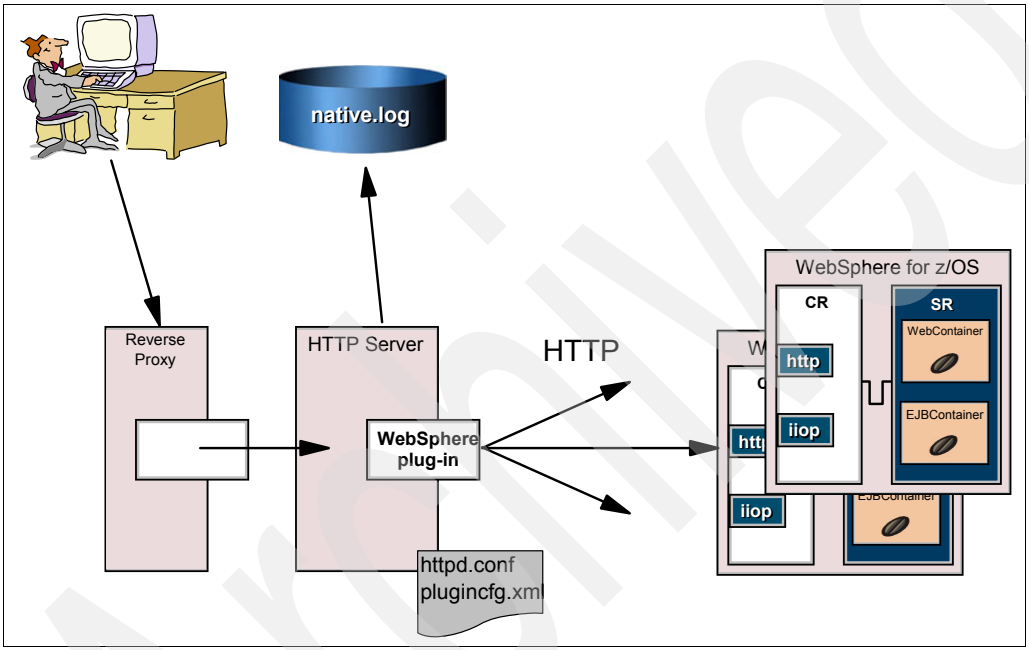


Figure 17-6 WebSphere HTTP plug-in architecture

After the plug-in has decided that an HTTP request should be passed to WebSphere, it packages up the request and sends it to an application server or a clone in a server group. It uses a set of rules in an XML file to decide which application server or clone should handle the request. This file can be generated from the WebSphere administrative console or from the command line.

The WebSphere HTTP plug-in uses the HTTP 1.1 transport to communicate between the HTTP server and the WebSphere applications.

17.3.1 WebSphere HTTP plug-in configuration

When the Web server receives an HTTP request, the plug-in needs to determine whether the request is meant for a Web application that is running in WebSphere. If the request isn't meant to be processed by WebSphere, the plug-in passes a return code to the Web server to say that the Web server should attempt to handle the request instead. If the request is meant for WebSphere, the plug-in needs to decide which node the application is running on and, in the case of a cloned application, which clone should handle the request.

The rules that the plug-in uses are in the file called plugin-cfg.xml. Example 17-1 is a simple sample.

Example 17-1 plugin-cfg.xml sample

```
<?xml version="1.0"?>
<Config>
    <Log LogLevel="Trace" Name="/WebSphere/BS05/appserver/logs/native.log"/>
    <VirtualHostGroup Name="default_host">
        <VirtualHost Name="wtsc59.itso.ibm.com:8069"/>
        <VirtualHost Name="wtsc59.itso.ibm.com:4469"/>
    </VirtualHostGroup>
    <ServerCluster Name="wd5nd5cd5sc59_Cluster">
        <Server Name="wd5nd5cd5sc59">
            <Transport Hostname="localhost" Port="9085" Protocol="http"/>
            <Transport Hostname="localhost" Port="9445" Protocol="https">
        </Transport>
        </Server>
        <PrimaryServers>
            <Server Name="wd5nd5cd5sc59"/>
        </PrimaryServers>
    </ServerCluster>
    <UriGroup Name="default_host_wd5nd5cd5sc59_Cluster_URIs">
        <Uri Name="/IBMEBizWeb"/>
        <Uri Name="/IBMEBizWeb/*"/>
        <Uri Name="/IBMForms"/>
        <Uri Name="/IBMForms/*"/>
        <Uri Name="/IBMClientSSL"/>
        <Uri Name="/IBMClientSSL/*"/>
    </UriGroup>
    <Route ServerCluster="wd5nd5cd5sc59_Cluster"
        UriGroup="default_host_wd5nd5cd5sc59_Cluster_URIs"
                VirtualHostGroup="default_host"/>
</Config>
```

The plugin-cfg.xml file is loaded at HTTP server startup and can be refreshed regularly, adding the RefreshInterval option in the Config tag.

17.3.2 WebSphere HTTP plug-in execution flow

When an HTTP request triggers into the WebSphere HTTP plug-in, the request is parsed and evaluated according the plugin-cfg.xml in order to determine the WebSphere Application Server Web container target HTTP transport handler.

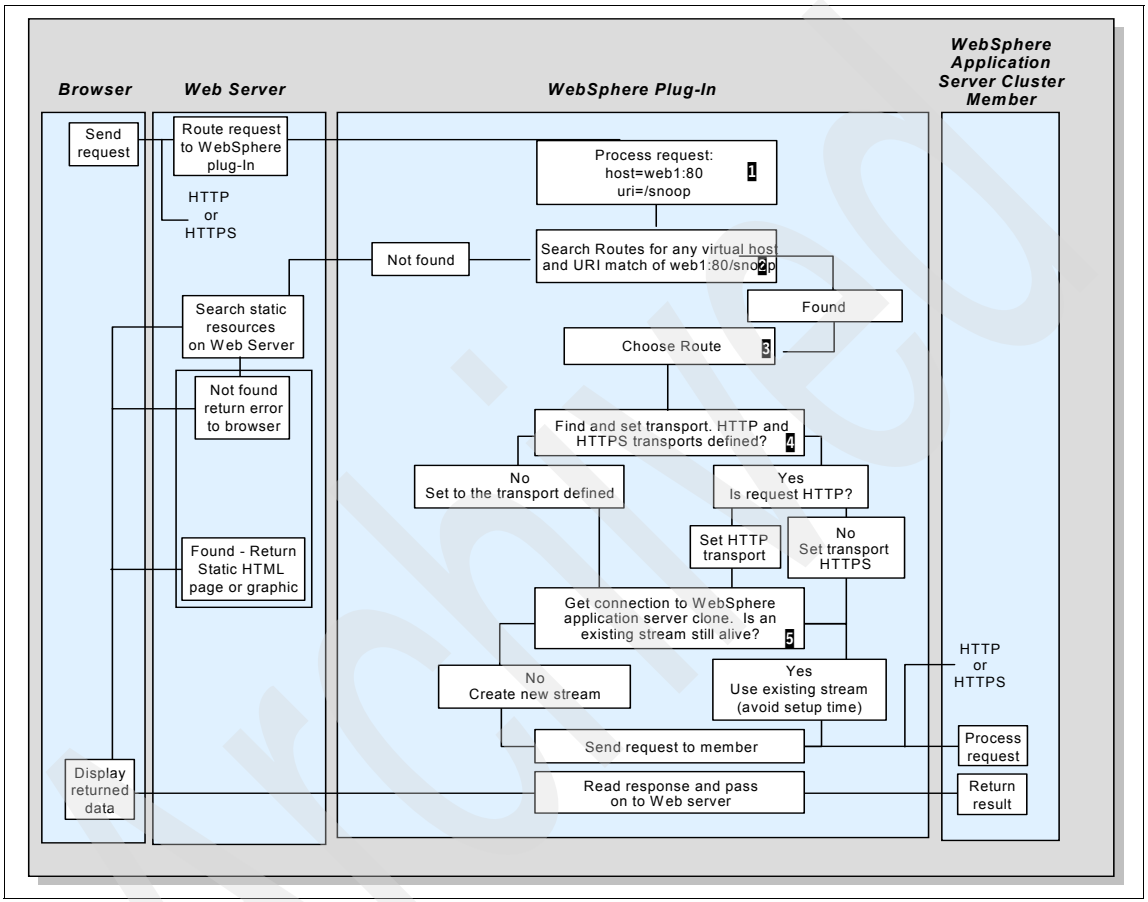


Figure 17-7 How the plug-in processes a request

Figure 17-7 describes in a simplified way how the WebSphere HTTP plug-in uses the configuration file to route requests to the HTTP handler:

1. The Web server passes the request to the plug-in (1). On HTTP distributed platforms, all requests go to the plug-in first. On IBM HTTP Server for z/OS the plug-in Service directive is processed as for any other usual directive, that is, in sequence as it appears in the HTTP configuration file.

2. The plug-in then starts by looking at all the defined Routes in plugin-cfg.xml. For each Route it searches through its configuration to find if there is any match to the virtual host and URI. It will find a match and then decide that WebSphere should handle the request (2). If there isn't any match, WebSphere will not handle the request.
3. The plug-in takes the request and separates the host name and port pair and URI. After all possible VirtualHostGroup and UriGroup are found, it selects a Route (3) and sets the ServerCluster.
4. If the application server has two transports associated to it, HTTP and HTTPS, the plug-in uses the one that matches the request transport protocol (4).Table 17-1 summarizes how the plug-in selects the transport.

Table 17-1 WebSphere HTTP plug-in selection of transport protocol

Incoming request protocol	Only HTTP transport available for the application server	Only HTTPS transport available for the application server	Both HTTP and HTTPS transports available for the application server
HTTP	HTTP	HTTPS	HTTP
HTTPS	HTTP	HTTPS	HTTPS

5. In box 5, the term *stream* is used. A stream is a connection to the Web container. Because HTTP 1.1 is used for connections between the plug-in and Web container, it is possible to maintain a connection (stream) over a number of requests. In our example, no previous connection has been created from the plug-in to the Web container, so a new stream is created. If a stream is already established, the plug-in uses the existing stream.
6. The request is sent to the application server and successfully processed. The plug-in passes the response on to the Web server, which in turn passes it back to the user's browser.

Tip: A detailed description of each XML tag and its relationships are available in the *WebSphere Information Center*. To find the appropriate section, open the *Information Center*, select the **Network Deployment** package (drop-down box in the upper-right corner). Then search for "plugin-cfg.xml file" and select the document called "plugin-cfg.xml file". Alternatively, in the left pane of the *Information Center*, select **Environment** → **Web servers** → **Configuring Web server plug-ins** → **Manually editing the plug-in configuration** → **plugin-cfg.xml file**.

17.3.3 Defining virtual hosts and HTTP transports

As previously discussed, the first criteria used by the WebSphere HTTP plug-in to route a request to a server is the matching between the presence of the Web application in the application server and the presence of the URI host in the list of hosts listed in the virtual host this application is bound to. To access an application from a WebSphere HTTP plug-in, you need to add the HTTP server host and port to the host aliases list associated with the virtual host to the plug-in configuration as it appears in Figure 17-8.

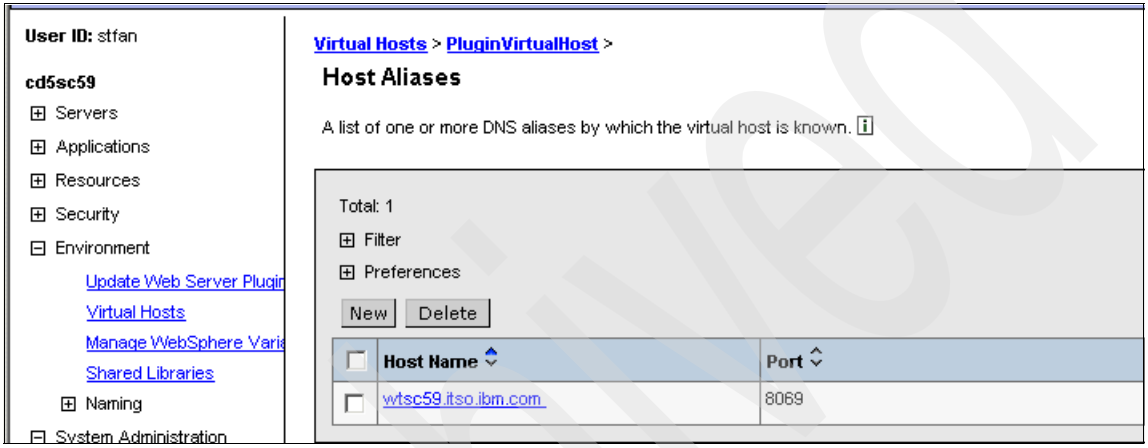


Figure 17-8 Virtual hosts definition

However, you need to take care that the virtual host mapped to your Web application will not contain host aliases you would not want to be used. For instance, you need to ensure that the administrative console application cannot be reached from outside of your administrative network.

Restriction: A good practice is to define different Web container HTTP transports of the same type (HTTP or HTTPS) in order to associate each transport port with a specific security zone. It allows not to mix different network flows on the same TCP/IP port, ease the firewall configuration, and dedicate one port for the administration that will only be listed in the administration application virtual host. The ability to define more than one HTTP transport of the same type for a Web container was not implemented in WebSphere Application Server for z/OS Version 5 at the time of writing. If you need this support now, check for the latest service levels.

17.3.4 WebSphere HTTP plug-in configuration file generation

Generating the plug-in configuration recreates the plugin-cfg.xml using the newest settings of objects in the cell. After the plug-in has reloaded its new configuration, clients are able to access the updated or new Web resources.

Note: When generating the plugin-cfg.xml file on any running node other than the deployment manager, the plugin-cfg.xml will be a subset of the cell-wide configuration. This is the result of each node only having a subset of the master configuration repository locally on the node.

Regeneration of the plug-in file can be performed at any point, whether application servers are running or not. There are two methods of manual regeneration:

- From the administrative console
- Through a command line using the `GenPluginCfg` command

From the administrative console

The administrative console can be used to regenerate the plug-in. In a distributed environment, the plug-in will only be regenerated on the deployment manager system. To regenerate the plug-in file manually from the administrative console:

1. Select **Environment** → **Update Web Server plug-in** as shown in Figure 17-9.
2. Click **OK**. A message will appear to say whether the plug-in regeneration was successful.

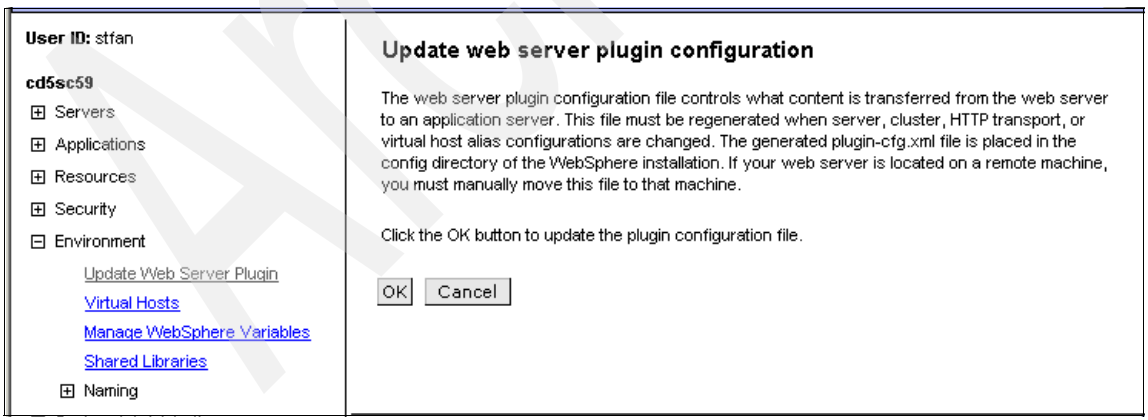


Figure 17-9 WebSphere HTTP plug-in generation page

From a command line

The command **GenPluginCfg** can be used under a z/OS UNIX session or a telnet session by changing the current directory to the /bin subdirectory under your server HFS mount point and typing:

```
./GenPluginCfg.sh -cell.name cellname -node.name nodename -server.name
servername
```

This will generate the plug-in configuration file on the node you have run the command on as file name plugin-cfg.xml in the server path config subdirectory. However, it is recommended that you run this command on the deployment manager node and then copy the file to all other nodes to make sure the complete environment's configuration is taken into account. When generating the plug-in on a node other than the deployment manager node, the plug-in will only be regenerated with a subset of the configuration that is local to the node you have run the command from. Therefore, generating the plug-in on the deployment manager node is the most convenient form of doing this.

Tip: Before level W501000 (PTF UQ80305), plugin-cfg.xml was generated in ASCII format. Because the WebSphere HTTP plug-in for z/OS only accepts a WebSphere HTTP plug-in configuration file in EBCDIC, it was necessary to convert the plugin-cfg.xml file. One easy way was to use the iconv shell s/OS UNIX command as follows:

```
iconv -f ISO8859-1 -t IBM-1047 plugin-cfg.xml >plugin-cfg.xml.ebcdic
```

Starting with level W501000 (PTF UQ80305), WebSphere generates two plug-in xml files:

- plugin-cfg.xml in EBCDIC encoding to be used by IBM HTTP Server for z/OS
- plugin-cfg.xmlascii in ASCII encoding to be used by HTTP servers on distributed platforms

17.3.5 IBM HTTP Server for WebSphere Application Server for z/OS HTTP plug-in configuration

The WebSphere HTTP plug-in for z/OS ships as part of the WebSphere Application Server for z/OS product. To use this plug-in on z/OS, you must have an IBM HTTP Server for z/OS or OS/390 Version configured as part of a z/OS system, either the same system where IBM WebSphere Application Server for z/OS runs, or another z/OS system, as long as it accesses the IBM WebSphere Application Server for z/OS data sets and HFS directories.

You need to add ServerInit, Service, and ServerTerm directives to the httpd.conf configuration file of the HTTP Server, as follows:

- The following ServerInit and ServerTerm directives to indicate the entry points to the plug-in's initialization and exit routines. These routines exist as entry points init_exit, and term_exit, respectively, in the ihs390WAS50Plugin_http.so DLL file.

  ```
  ServerInit /webspherer_install_root/bin/ihs390WAS50Plugin_http.so:init_exit
  fully_qualified_path
  ServerTerm /websphere_install_root/bin/ihs390WAS50Plugin_http.so:term_exit
  ```

  Where `websphere_install_root` is the name of the root directory where the Web server is installed, and `fully_qualified_path` is the fully qualified path to the plugin-cfg.xml file.

- The following Service directive for each application that will be using the Web server plug-in. This directive indicates the entry point to the plug-in's request routine. The request routine exists as the entry point service_exit in the ihs390WAS50Plugin_http.so Dynamic Link Library (DLL) file.

  ```
  Service /webapp_contextroot/*  /websphere_install_root/bin/
  ihs390WAS50Plugin_http.so:service_exit
  ```

  Where `webapp_contextroot` is the context root of the Web application, and `websphere_install_root` is the name of the root directory where the Web server is installed.

Using the administrative console, make sure the virtual host is configured with an alias for the port number used by the z/OS IBM HTTP Server. Specify the same port on a <Transport Hostname> element in the plugin-cfg.xml file.

To activate the configuration, stop and restart both the application server and the HTTP Server. If the WebSphere HTTP plug-in for z/OS comes up when the HTTP Server starts again, you receive the following messages:

```
WebSphere HTTP Plug-in for z/OS Version 5.00 Service Level 0 is starting
WebSphere HTTP Plug-in for z/OS initializing with configuration file:
        fully_qualified_path_to_the_plugin-cfg.xml_file
WebSphere HTTP Plug-in for z/OS initialization went OK :-)
```

Note: the SSL configuration is described in 17.4.2, “Enabling mutual authentication with IBM HTTP Server for z/OS” on page 589.

17.3.6 Distributed HTTP server WebSphere HTTP plug-in configuration

The WebSphere HTTP plug-in on distributed platforms installation files for non-z/OS HTTP servers are no longer delivered with WebSphere Application

Server for z/OS and OS/390 Version 5 as they were with WebSphere Application Server for z/OS and OS/390 V4.0.1.

The installation of the Web server plug-in is part of the distributed WebSphere installation. When running the WebSphere installation, you have two options:

- Install the IBM HTTP Server packaged with WebSphere Application Server and the associated WebSphere HTTP plug-in.
- Install just the WebSphere HTTP plug-in and specify the location of the Web server configuration file that needs to be updated.

When you launch the WebSphere installation executable, whatever option you chose, you need to select a custom installation as shown in Figure 17-10. If you do not, the installation wizard will install WebSphere Application Server and IBM HTTP Server configured with WebSphere HTTP plug-in.

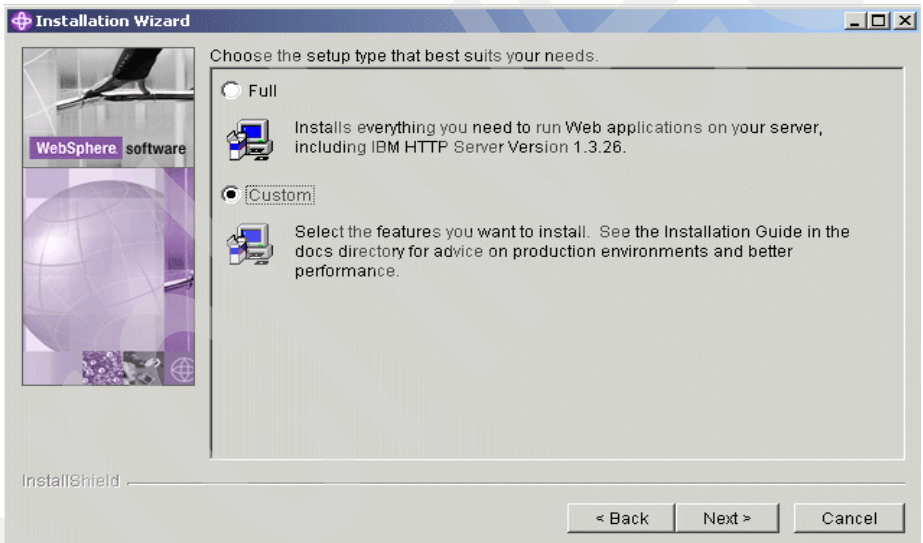


Figure 17-10 Selecting Custom installation

After the custom installation as been selected, the installation wizard lists the features of the product so you can pick which one you want to have installed.

If you select IBM HTTP Server and IBM HTTP Server plug-in as shown in Figure 17-11 on page 580, the installation wizard will install IBM HTTP Server preconfigured with the WebSphere HTTP plug-in.

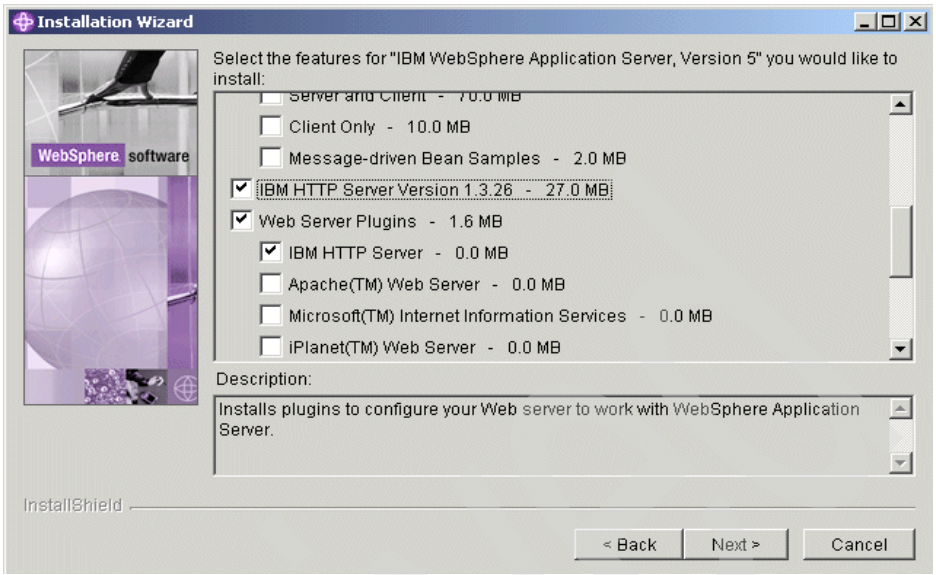


Figure 17-11 Installing IBM HTTP Server with WebSphere HTTP plug-in

If you have already configured an HTTP Server, you can only select the corresponding WebSphere HTTP plug-in and then the installation wizard will prompt you for the path name where your HTTP configuration file is in order to update it after the plug-in code is installed; see Figure 17-12.

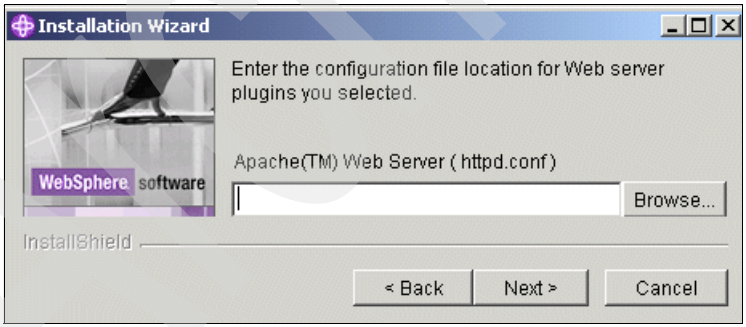


Figure 17-12 Entering the HTTP configuration file path

After the installation has been completed, you can verify that it is effective by editing the HTTP configuration file and looking for the following lines:

```
LoadModule ibm_app_server_http_module "C:\Program
Files\WebSphere\AppServer/bin/mod_ibm_app_server_http.dll"
WebSpherePluginConfig "C:\Program
Files\WebSphere\AppServer/config/cells/plugin-cfg.xml"
```

Note: The SSL configuration is described in 17.6.1, “Enabling mutual authentication with a non-z/OS HTTP server” on page 594.

17.3.7 Protection setups in the IBM HTTP Server and security credential forwarding

When the HTTP Server does not enforce any protection on WebSphere application URLs, the WebSphere HTTP plug-in appears just like a security neutral gateway that lets the authentication tokens flow between the browser and the application servers. This is fine for basic and for form-based authentication, but this disallows any HTTPS certificate authentication because the browser needs to initiate an SSL session with the HTTP Server, which can only be done when the HTTP Server enforces client certificate protection. This specific case is discussed in 17.4, “SSL authentication with the WebSphere HTTP plug-in” on page 583.

Disregarding SSL certificate authentication, the only opportunity to reuse an HTTP authentication is when WebSphere applications are protected with basic authentication. After the HTTP Server has performed a basic authentication with the browser, an Authorization HTTP header is created with an encoded user ID and password. This private header is forwarded by the plug-in to the application server, and the Web container will reuse it to log in any basic authentication-protected application. This behavior might not ever be desirable because the HTTP Server needs to share the user registry with WebSphere.

Table 17-2 summarizes the various possibilities of authentication when using a Web server plug-in.

Table 17-2 Cross authentication using a Web server plug-in

HTTP server protection	Web container login	Result
None	Basic	Web container prompts the realm through the plug-in.
	Form-based	Web container redirects request to the form login page through the plug-in.
	Certificate	Depends on HTTPS transport availability and TrustedProxy custom variable. See 17.4, “SSL authentication with the WebSphere HTTP plug-in” on page 583.

HTTP server protection	Web container login	Result
Basic	Basic	Web container reuses HTTP server authentication [a].
	Form-based	Web container ignores HTTP server authentication and redirects request to the form login page through the plug-in.
	Certificate	Depends on HTTPS transport availability and TrustedProxy custom variable. See 17.4, “SSL authentication with the WebSphere HTTP plug-in” on page 583.
SSL	Basic	Web container ignores certificate authentication and prompts the realm through the plug-in.
	Form-based	Web container ignores certificate authentication and redirects request to the form login page through the plug-in.
	Certificate	Depends on TrustedProxy custom variable. See 17.4, “SSL authentication with the WebSphere HTTP plug-in” on page 583.

a. One advantage of this configuration when using IBM HTTP Server for z/OS is that you can use the PWAPI inside IBM HTTP Server to handle expired RACF passwords (PWAPI is a program that uses the GWAPI interface).

Tip: Usually, protecting Web container requests with HTTP Server protection is not recommended (except for certificate authentication that requires the HTTP Server to handle an SSL communication with the client). Therefore, in some special cases, you might consider using HTTP protection directives. An example of such a situation is when using SSL authentication to authenticate to an HTTP Server and then using a form-based authentication in the Web container. This offers some advantages, such as:

- Certificate and user ID/password double authentication.
- Requests flowing to the Web container have been authorized by the HTTP server.
- Security applies on every layer of the physical implementation.

17.4 SSL authentication with the WebSphere HTTP plug-in

The HTTP transport handler supports mutual authentication of client and server over Secure Sockets Layer (SSL) sessions. This is illustrated in Figure 17-13.

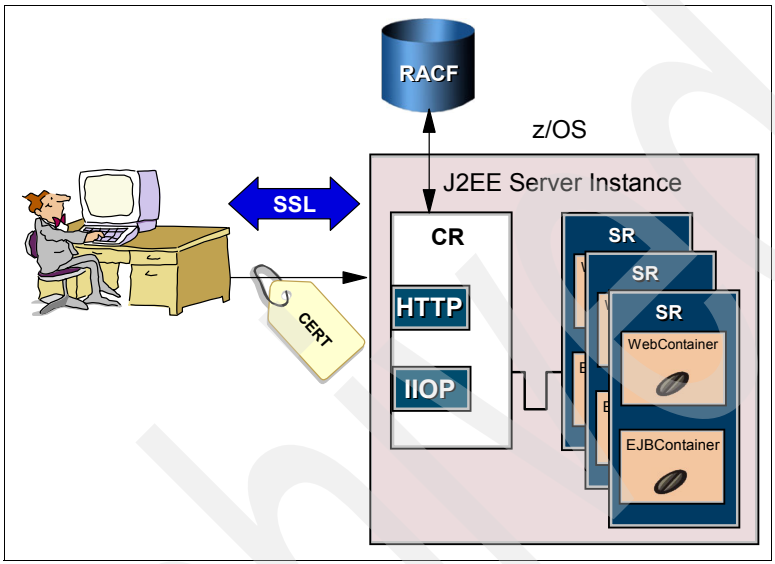


Figure 17-13 SSL mutual authentication

During the SSL handshake, the HTTP transport handler sends its certificate to the client, which responds with its own certificate. If a Web application then requests authentication by a client certificate, the client's certificate will be used. The HTTP transport handler calls the SAF interface to map the client certificate to an SAF user ID. The resulting user ID is used to build the principal for the execution of Web applications.

Note: In this configuration, the MutualAuthCBindCheck custom property on HTTPS transport indicates how a client certificate should be resolved to a SAF principal. The default is false. If this property is set to true:

- All SSL connections from a browser must have a client certificate.
- All client certificates should map to a RACF user ID.
- The user ID associated with that client certificate must have RACF CONTROL authority for CB.BIND.servername, where *servername* is the value of the *Cluster transition name*.

If these conditions are not met, the connection is closed.

Table 17-4 on page 588 summarizes how the TrustedProxy and MutualAuthCBindCheck custom properties of an HTTPS transport are changing the Web container authentication process as well as the needed RACF configuration.

Notes:

- Before the WebSphere APAR PQ76644 level, WebSphere tried to map SSL client certificates with a RACF user ID whether or not the user registry was RACF and wether or not MutualAuthCBindCheck on HTTPS was set: if the mapping failed, the request was rejected.
- If a request received by the HTTP transport handler is accepted for processing by a trust association interceptor, the client's certificate is used to establish a mutually authenticated SAF session, but it will not be used to derive an SAF user ID to build the principal. The SAF user ID returned by the TAI will be used instead.

17.4.1 WebSphere HTTP plug-in client certificate forwarding

The HTTP transport handler is able to receive a client certificate through a private header in the HTTP message. A private header of this kind is created by the plug-in when it forwards an HTTP request received from a client browser over a mutually authenticated SSL session. Figure 17-14 on page 585 illustrates this protocol.

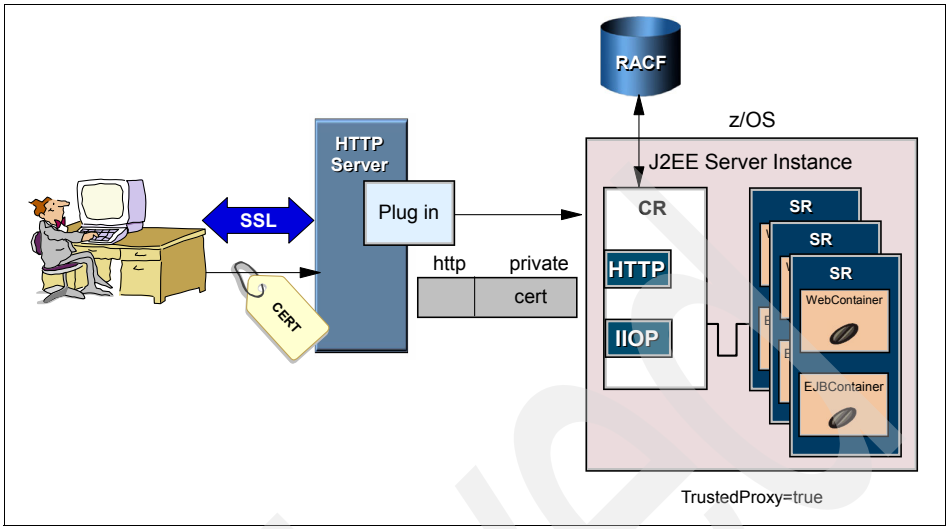


Figure 17-14 Using a Web server plug-in for certificate forwarding

SSL with mutual authentication begins with the SSL handshake, in which the Web server provides its server certificate to the browser. The browser responds with its client certificate. When the request is forwarded from the plug-in to the HTTP transport handler, it is accompanied by the browser's client certificate in a private header. This client certificate, mapped by SAF, will be used to build the principal for execution of the Web application.

Note that the connection between the plug-in and the HTTP transport handler does not have to be SSL in order for the client certificate to be forwarded. If the Web server is within a trusted network, SSL might not be necessary for this connection. However, especially if there is any question about the source of the requests that might be received by the HTTP transport handler, mutual authentication of the plug-in and HTTP transport handler would be prudent.

Remember that there is no SSL session from the client to the HTTP transport handler. The Web server is able to verify that the client owns the client certificate (2) through the SSL handshake, but the plug-in merely forwards the certificate to the HTTP transport handler. None of the messages reaching the HTTP transport handler are signed with the client's private key. Therefore, the HTTP transport handler must trust the plug-in to have verified ownership of the client certificate.

In other words, a rogue Web server or plug-in could forward any available client certificate in the private header of any HTTP message of its choosing, thus impersonating the owner of the client certificate. As a result, the HTTP transport needs a custom property, TrustedProxy, set to true in order to accept SSL certificates from private headers. At the same time this property is set to true, it

will also disable any direct HTTP transport handler SSL authentication from a client.

Note: You need to have the PQ77750 correction in order to get TrustedProxy property to behave as described here.

Table 17-3 summarizes how the TrustedProxy property modifies the Web container behavior.

Table 17-3 Authentication variation according TrustedProxy property

Configuration	TrustedProxy=false	TrustedProxy=true
Request goes directly to HTTP transport handler	Basic authentication enabled.	Basic authentication enabled
	Form-based authentication enabled.	Form-based authentication disabled[a]
	Certificate authentication enabled if the Web container has an HTTPS transport.	Certificate authentication disabled
Request passes through an WebSphere HTTP plug-in	Basic authentication enabled.	Basic authentication enabled
	Form-based authentication disabled[b].	Form-based authentication enabled
	Client certificate authentication disabled. HTTP server certificate is used to authenticate if the Web container has an HTTPS transport.	Certificate authentication enabled

a. When a user connects to a form-based protected application, the Web container first sends a redirect HTTP command to the browser. This redirect command contains the complete URL to the form login page. When using the WebSphere HTTP plug-in, the Web container should use private headers to determine the port address of the plug-in to build the redirect URL. In this configuration, the private headers are missing.
b. When a user connects to a form-based protected application, the Web container first sends a redirect HTTP command to the browser. This redirect command contains the complete URL to the form login page. When using the WebSphere HTTP plug-in, the Web container should use private headers to determine the port address of the plug-in to build the redirect URL. In this configuration, private headers are present but the Web container should ignore them.

If, therefore, the HTTP transport handler needs to verify the identity of the Web server and plug-in, to ensure that the Web server can be trusted to have verified that the originator of the HTTP message is in fact the owner of the client

certificate being forwarded in the private header, we can combine mutual authentication capabilities as shown in Figure 17-13 on page 583 with the certificate forwarding capabilities illustrated in Figure 17-14 on page 585 to obtain the capability shown in Figure 17-15.

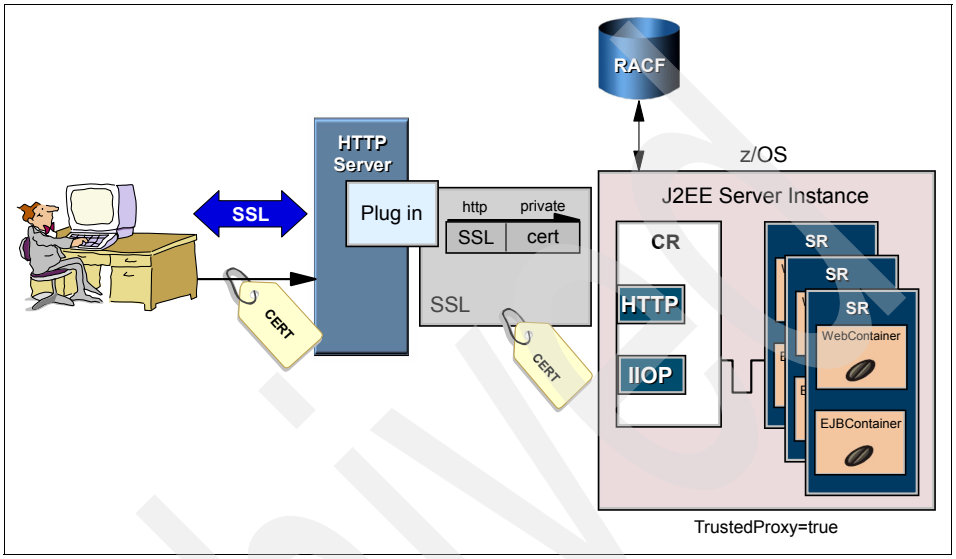


Figure 17-15 Combining mutual authentication and certificate forwarding

Here, as before, the client and Web server exchange digital certificates. When an SSL session is established between the plug-in and the HTTP transport handler, the HTTP transport handler sends its certificate to the plug-in and the plug-in responds with the Web server's certificate. When the Web server receives a request from the client over the mutually authenticated SSL session, the plug-in now forwards the client's certificate over the mutually authenticated SSL session to the HTTP transport handler. The HTTP transport handler will normally use this certificate to establish the principal for the Web application request by calling the SAF interface to map the certificate to an SAF user ID.

Note: In this configuration, the MutualAuthCBindCheck custom property on HTTPS transport controls how a plug-in certificate should be checked with SAF.

If this property is set to true:

- All SSL connections from a plug-in must have a client certificate.
- All client certificates have to map with a RACF user ID.
- The user ID associated with that plug-in certificate must have RACF CONTROL authority for CB.BIND.servername, where *servername* is the value of the *Cluster transition name*.

If these conditions are not met, the connection is closed.

This is the way to control which plug-ins are authorized to connect to WebSphere.

If the HTTP transport handler is not configured to accept private headers, the Web server's certificate will be used for this mapping.

Table 17-4 summarizes how the TrustedProxy and MutualAuthCBindCheck custom properties of an HTTPS transport are changing the Web container authentication as well as the RACF configuration setup.

Table 17-4 TrustedProxy and MutualAuthCBindCheck

Configuration	TrustedProxy	MutualAuth CBindCheck	User ID associated with the plug-in certificate needs	User ID associated with the end client certificate needs	Authentication
Requests flowing through a WebSphere HTTP plug-in with mutual SSL authentication	true	true	CB.BIND CBS390	CBS390	End user user ID
	true	false	CBS390	CBS390	End user user ID
	false	true	CB.BIND CBS390		Plug-in user ID
	false	false	CBS390		No authentication[a]

Configuration	TrustedProxy	MutualAuth CBindCheck	User ID associated with the plug-in certificate needs	User ID associated with the end client certificate needs	Authentication
Requests coming directly to the HTTP transport handler	true	true	Request rejected because of lack of private header but CB.BIND is also active for plugin certificates.		
	true	false	Request rejected because of lack of private header.		
	false	true	Not Applicable	CB.BIND CBS390	End user user ID
	false	false	Not Applicable	CBS390	No authentication[b]

a. This case is dependant on APAR PQ76644. Without this APAR, the plug-in certificate associated user ID is used for authentication.
b. This case is dependant on APAR PQ76644. Without this APAR, the client certificate associated user ID is used for authentication.

Important: Before the WebSphere APAR PQ76644 level, WebSphere tried to map SSL client certificates with a RACF user ID wether or not the user registry was RACF and wether or not MutualAuthCBindCheck on HTTPS was set: if the mapping failed, the request was rejected.

17.4.2 Enabling mutual authentication with IBM HTTP Server for z/OS

To enable mutual authentication with IBM HTTP Server for z/OS you need to generate three different SSL certificates, as illustrated in Figure 17-16 on page 590:

- A client certificate. This certificate is used to establish the SSL communication with HTTP server and then flows in a private header to the Web container.
- An HTTP server certificate that will be used by the HTTP server to establish the SSL communication with the user browser and will be used by the plug-in to establish the SSL communication with the Web container.

Important: On an IBM HTTP Server for z/OS, the plug-in uses the HTTP Server certificate for HTTPS transport. Any plugin-cfg.xml configuration to point the plug-in to key ring and stash files will be ignored.

- A WebSphere for z/OS certificate that will be used to communicate with the plug-in.

You need to ensure the consistency of certificate authorities when issuing certificates. In order to be able to establish a communication with SSL with a remote system, the SSL communication initializer needs to have its certificate authority accepted by the host he wants to connect to.

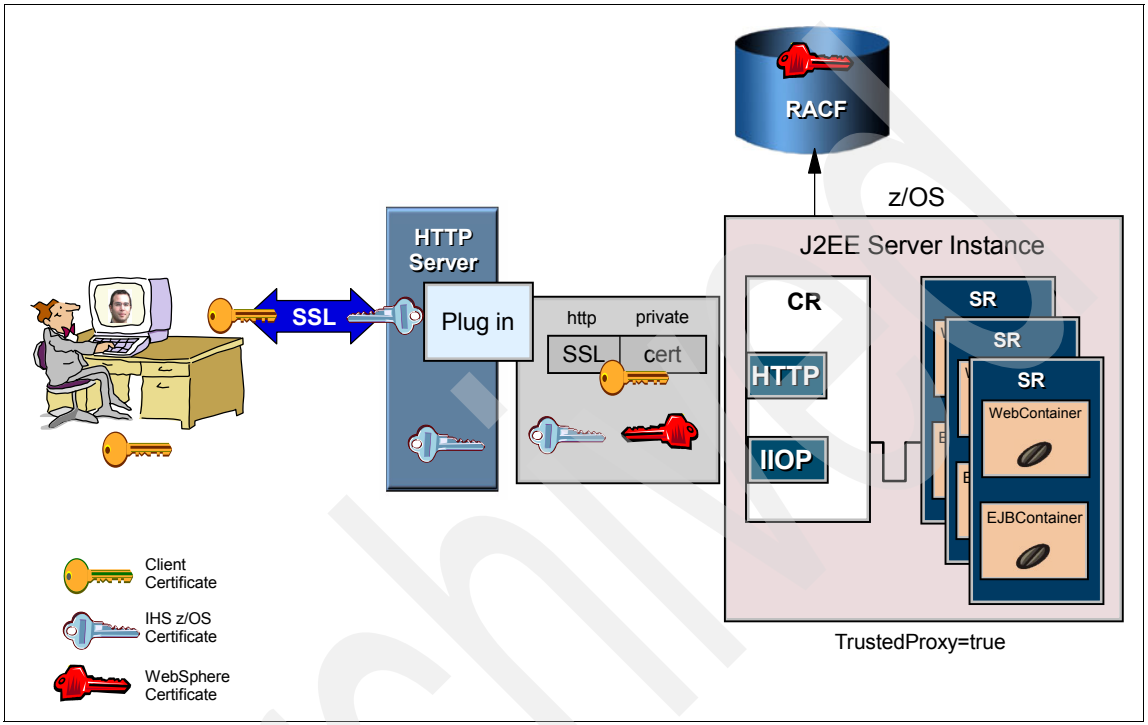


Figure 17-16 SSL plug-in mutual authentication with IBM HTTP Server for z/OS

The next sections describe how we generated the certificates for our tests.

Note: We generated different SSL certificates to accomplish our tests. Because we used a test environment, we chose the easiest way to do so.

There are several options to generate and manage SSL certificates for a large number of users. As a result, if you have to implement SSL certificates in a production environment, you have to consider using a Public Key Infrastructure (PKI).

SSL certificate management

We reused the same client certificate as for the SSL test in “Certificate-based authentication” on page 524.

We generated the IBM HTTP Server key with the following commands:

```
RACDCERT ADDRING(WASKEYRING) ID(WEBSRV)

RACDCERT ID (WEBSRV) GENCERT +
  SUBJECTSDN(CN('wtsc59.itso.ibm.com') OU('ITSO') O('IBM') c('us')) +
  WITHLABEL('WEBSTEF') SIGNWITH(CERTAUTH LABEL('WS for z/OS CA'))

RACDCERT ID(WEBSRV) CONNECT (LABEL('WEBSTEF') RING(WASKEYRING) DEFAULT)
```

HTTP Server configuration file

We protected /IBMClientSSL/ZosAndWasSecure into the IBM HTTP Server for z/OS was.conf file with the following directives:

Example 17-2 A piece of the HTTP Server configuration file

```
sslmode    on
sslport    4469
SSLClientAuth   passthru
keyfile WASKEYRING SAF
Protection POK_Secrets_SSL {
   ServerId        ZOSHTTPServerSSL
   SSL_ClientAuth  client
   AuthType        Basic
   PasswdFile      %%SAF%%
   Userid          %%CERTIF%%
   Mask            Anybody
}
Protect /IBMClientSSL/ZosAndWasSecure/*  POK_Secrets_SSL
```

WebSphere

We added the HTTP Server SSL port to the list of hosts included in our application virtual host and verified that the plug-in then contained the following lines:

```
<Transport Hostname="wtsc59.itso.ibm.com" Port="9445" Protocol="https"/>

<VirtualHost Name="tot136.itso.ibm.com:443"/>
```

17.5 Validation tests for IBM HTTP Server for z/OS

To validate IBM HTTP Server for z/OS plug-in security configurations, we performed the following tests.

17.5.1 Test case PG1: Authentication through WebSphere HTTP plug-in for z/OS with HTTP and HTTPS transport

We were able to validate authentication using the WebSphere HTTP plug-in for z/OS with base Application Servers for basic, form-based, and certificate authentications.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet principal on the first page of the application. For all of the tests, we logged on a first server (Server 1 in the test case table) and looked at the Servlet Security Info table to check that the principal was the user ID entered at the security prompt. Figure 17-17 shows the Servlet Security Info table in SWIPE.

Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 17-17 Servlet Security Info table from SWIPE

Based on the authentication method, we used the URL listed in Table 17-5.

Table 17-5 URLs to validate security with the WebSphere HTTP plug-in

Authentication method	SWIPE relative URL
Basic	/IBMEBizWeb/secure/EJBCaller
From based	/IBMForms/secure/EJBCaller
Certificate	/IBMClientSSL/secure/sslEjbCaller

Table 17-6 WebSphere HTTP plug-in Authentication Login Test Case 1 (PG1)

PG1 config: WebSphere Base, HTTP Server on z/OS, LTPA, RACF						
Transport	**Authentication method**	**Input**			**Output**	
		User ID	**RunAs**	**Map-ping**	**Principal**	**Comments**
HTTP transport	Basic	stfan	Caller	none	stfan	
	Form	stfan	Caller	none	stfan	
	Certificate	cert	Caller	none	stfan	

PG1 config: WebSphere Base, HTTP Server on z/OS, LTPA, RACF						
Transport	**Authentication method**	**Input**			**Output**	
		User ID	**RunAs**	**Map-ping**	**Principal**	**Comments**
HTTPS transport	Basic	stfan	Caller	none	stfan	
	Form	stfan	Caller	none	stfan	
	Certificate	cert	Caller	none	stfan	

17.5.2 Test case PG2: Authentication through WebSphere HTTP plug-in for z/OS using HTTP transport and authentication mechanisms

We were able to validate authentication using the WebSphere HTTP plug-in for z/OS with base Application Servers for basic, form-based, and certificate authentications.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet principal on the first page of the application. For all of the tests, we logged on a first server Server 1 in the test case table) and looked at the Servlet Security Info table to check that the principal was the user ID entered at the security prompt. Figure 17-18 shows the Servlet Security Info table in SWIPE.

Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 17-18 Servlet Security Info table from SWIPE

Based on the authentication method, we used the URLs listed in Table 17-7.

Table 17-7 URLs to validate security with WebSphere HTTP plug-in

Authentication method	Swipe relative URL
Basic	/IBMEBizWeb/secure/EJBCaller
From based	/IBMForms/secure/EJBCaller
Certificate	/IBMClientSSL/secure/sslEjbCaller

Table 17-8 WebSphere HTTP plug-in Authentication Login Test Case 2 (PG2)

PG2 config: WebSphere base, HTTP Server on z/OS, custom user registry					
Authentication mechanism	**Authentication method**	**Mutual Authentication with SSL**	**Input**	**Output**	
			User ID	**Principal**	**Comments**
SWAM	Basic	No	stfan	stfan	
	Form	No	stfan	stfan	
	Certif.	Yes	cert	CN=stfan, OU=ITSO, O=IBM, C=us	
LTPA	Basic	Yes	stfan	stfan	
	Form	Yes	stfan	stfan	
	Certif.	Yes	cert	stfan	
ICSF	Basic	No	stfan	stfan	
	Form	No	stfan	stfan	
	Certif.	Yes	cert	stfan	

17.6 WebSphere HTTP plug-in on distributed platforms

The configuration of the WebSphere HTTP plug-in on distributed platforms is the same as for the IBM HTTP Server for z/OS except that it needs an SSL key ring and a stash file to retrieve the certificates for communication with IBM WebSphere Application Server for z/OS.

SSL key ring and stash files are created by the iKeyman utility and are configured in the plugin-cfg.xml HTTPS transport tag as follows:

```
<Property name="keyring" value="keyring.kdb"/>
<Property name="stashfile" value="stashfile.sth"/>
```

17.6.1 Enabling mutual authentication with a non-z/OS HTTP server

To enable mutual authentication with a distributed HTTP server, you need to generate four different SSL certificates, as illustrated in Figure 17-19:

- A client certificate. This certificate is used to establish the SSL communication with the HTTP server and then flows in a private header to the Web container.

- An HTTP server certificate that will be used by the HTTP server to establish the SSL communication with the user browser.
- A plug-in certificate that will be used by the plug-in to establish the SSL communication with the Web container.
- A WebSphere for z/OS certificate that will be used to communicate with the plug-in.

You need to insure the consistency of certificate authorities when issuing certificates. In order to be able to establish a communication with SSL with a remote system, the SSL communication initializer needs its certificate authority to be accepted by the host it wants to connect to.

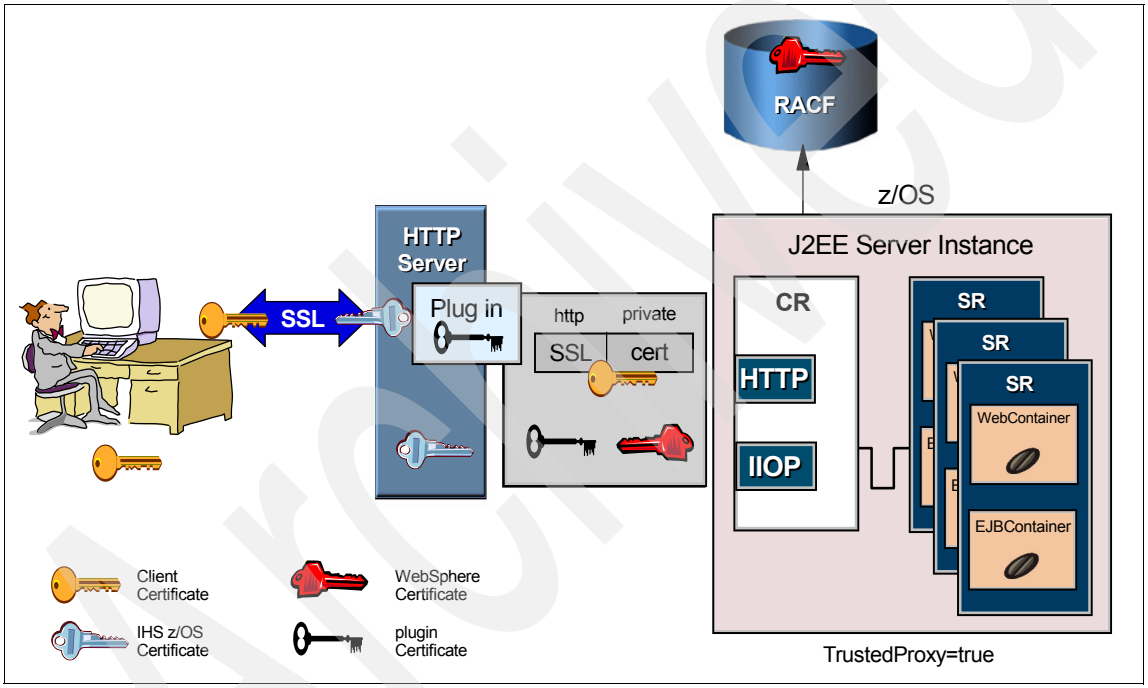


Figure 17-19 SSL plug-in mutual authentication with IBM HTTP Server for z/OS

The next sections describe how we generated the certificates for our tests.

Note: We generated different SSL certificates to accomplish our tests. Because we used a test environment, we chose the easier way to do so.

There are several options to generate and manage SSL certificates for a large number of users. As a result, if you have to implement SSL certificates in a production environment, you have to consider using a Public Key Infrastructure (PKI).

17.6.2 SSL certificate generation

We reused the same client certificate used for the SSL test in “Certificate-based authentication” on page 524.

HTTP server certificate

We generated the HTTP server key with the following commands:

```
RACDCERT ID (WEBSRV) GENCERT UBJECTSDN(CN('tot136.itso.ibm.com') OU('ITSO')
O('IBM') c('us')) WITHLABEL('TOT136') SIGNWITH(CERTAUTH LABEL('WS for z/OS
CA'))

RACDCERT ID(WEBSRV) CONNECT (LABEL('TOT136') RING(WASKEYRING))
```

WebSphere HTTP plug-in on distributed platforms certificate

We generated the WebSphere HTTP plug-in certificate with the following commands:

```
RACDCERT ID (WEBSRV) GENCERT SUBJECTSDN(CN('plugin.itso.ibm.com')
OU('ITSO') O('IBM') c('us')) WITHLABEL('PLUGIN') SIGNWITH(CERTAUTH
LABEL('WS for z/OS CA'))

RACDCERT ID(WEBSRV) CONNECT (LABEL('PLUGIN') RING(WASKEYRING))
```

This certificate is to be used by the WebSphere HTTP plug-in to establish the mutual authentication with the WebSphere Application Server Web container HTTPS transport handler.

The RACCERT command associates this certificate with the WEBSRV user ID but you might consider associating a different RACF user ID with every WebSphere HTTP plug-in.

17.6.3 Key ring generation

Here, we describe how to create a key ring and stash file to be used by IBM HTTP Server for Windows. The creation of these files for the WebSphere HTTP plug-in on Windows uses the exact same procedure.

We decided to generate the key ring and the stash files on Windows with the graphical version of the iKeyman utility installed as part of the installation of IBM HTTP Server for Windows.

Note: The *WebSphere Application Server V5 Information Center* (in the section “Configuring the IBM HTTP Server for distributed platforms and the Web server plug-in for Secure Sockets Layer”) describes the same procedure but using the iKeyman packaged with IBM HTTP Server for z/OS. Both tools, on z/OS and Windows, generate the same files. We used the Windows iKeyman version because of its graphic interface, which is easier to use and describe.

So we first exported the certification authority key:

```
RACDCERT CERTAUTH    EXPORT(LABEL('WS for z/OS CA')) +
       DSN(CERTAUTH.DERBIN) FORMAT(CERTDER)      PASSWORD('XXXXXX')
OPUT CERTAUTH.DERBIN '/tmp/certauth.der' binary convert(no)

RACDCERT ID(WEBSRV)  EXPORT(LABEL('TOT136')) +
       DSN(CERT136.P12BIN) FORMAT(PKCS12DER)    PASSWORD('XXXXXX')
OPUT CERT136.P12BIN '/tmp/ctot136b.p12' binary convert(no)
```

With FTP, we downloaded (in binary file format) the two files on the Windows machine hosting the HTTP Server.

In Programs, IBM HTTP Server 1.3.26, we clicked **Start key management utility**, which opened the window shown in Figure 17-20 on page 598.

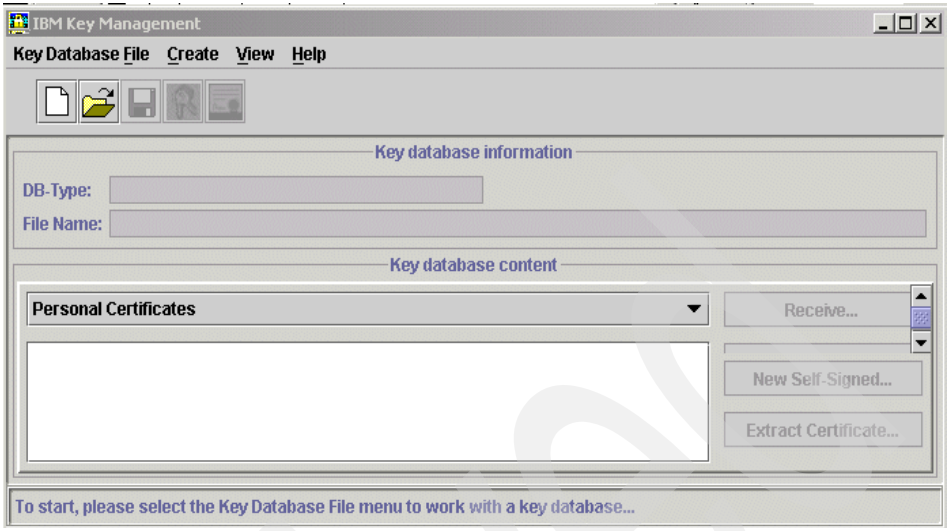


Figure 17-20 iKeyman welcome page

The first step was to create a key ring with option **New** under Key Database File. This opened the window shown in Figure 17-21.

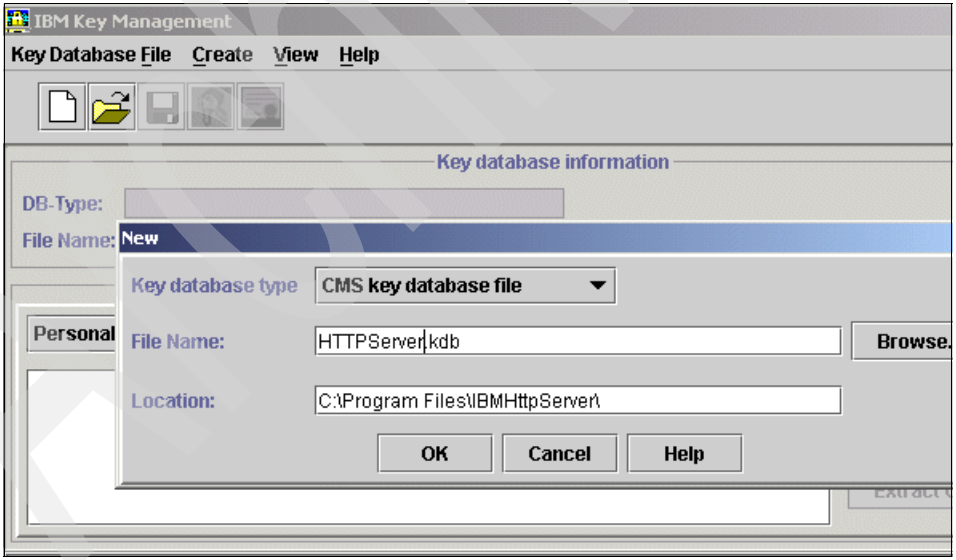


Figure 17-21 iKeyman new key ring

We entered the name of the key ring file, and after clicking **OK**, iKeyman opened a second menu to enter the password protecting the key ring. The menu is shown

in Figure 17-22. On this menu, we entered the password and selected the **Stash the password to a file** check box.

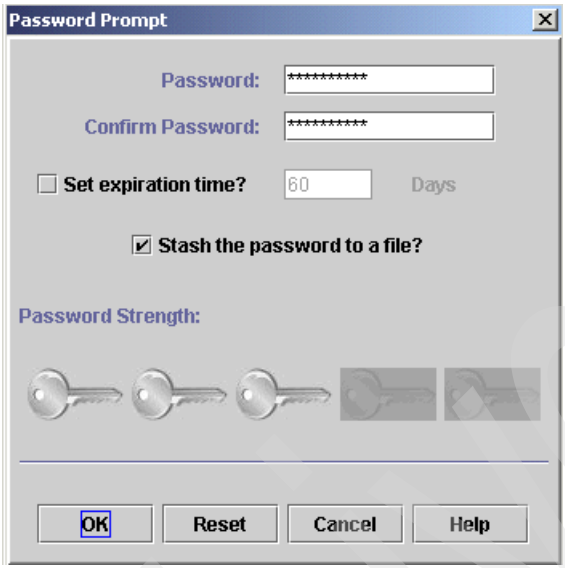


Figure 17-22 Additional data to create a key ring

After the key ring file was created, iKeyman presented the list of certificate authorities that are valid for the key ring. We then needed to add the WebSphere certificate authority to this list. We clicked **Add**, which opened the menu shown in Figure 17-23 on page 600, selected the **Der** format, picked the downloaded der file, entered the password created at key exportation time, and finally gave a label identifier to this certificate. The WebSphere certification authority certificate was then added to the list of signer certificates.

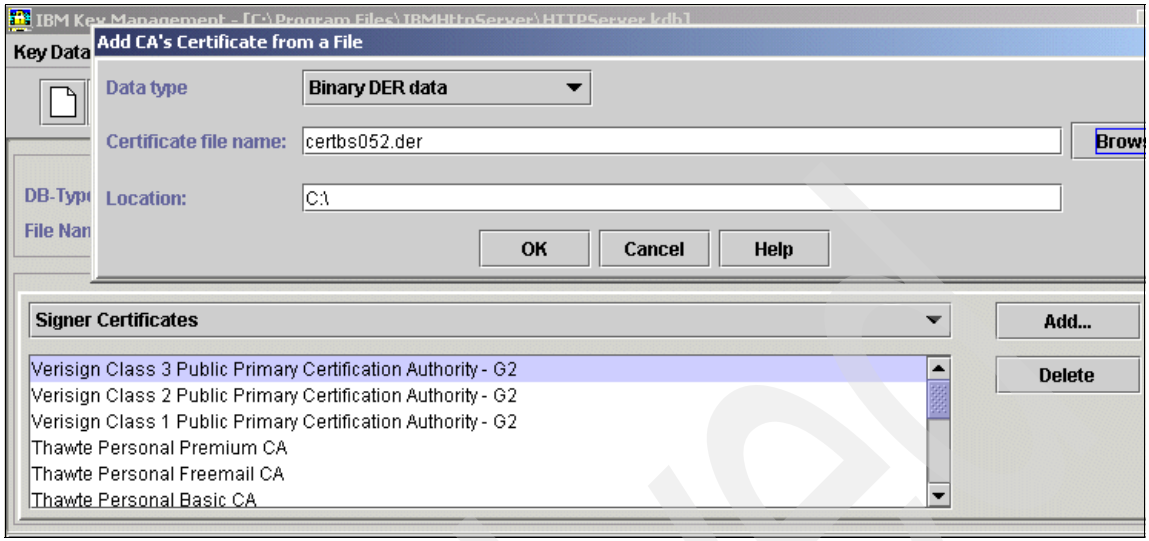


Figure 17-23 Adding a signer certificate

We then imported the HTTP Server certificate by selecting Personal Certificates on the bar in the middle of the iKeyman window, and clicked **Import**.

On the Import key menu, we selected **PKCS12** as the file format, picked the downloaded HTTP Server certificate and entered the password used at key exportation time.

The HTTP Server certificate then appeared on the client certificate list as shown in Figure 17-24 on page 601.

We just had to close the key ring and go to configure IBM HTTP Server for Windows.

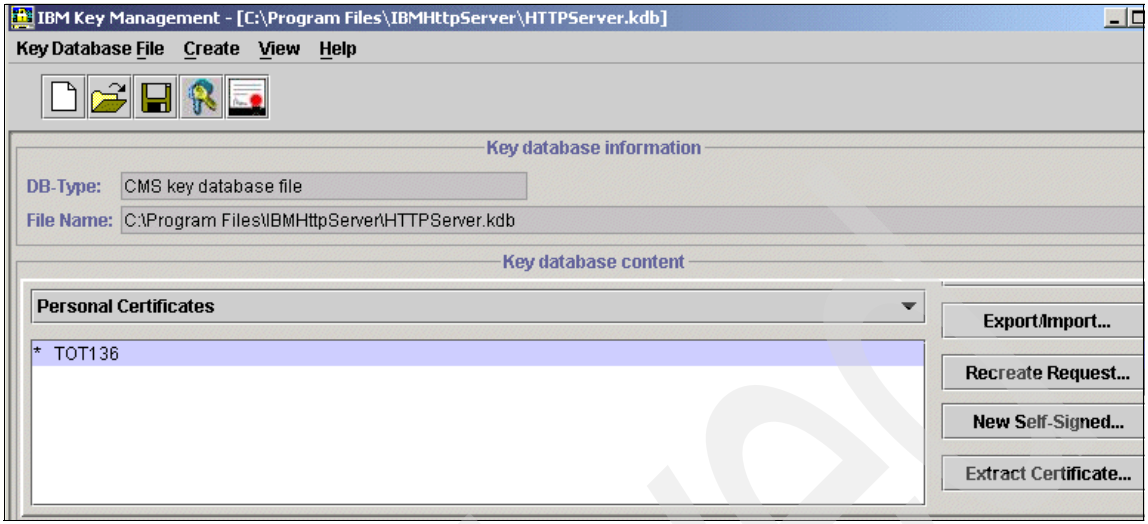


Figure 17-24 List of Personal Certificates

17.6.4 IBM HTTP Server configuration file

We modified the IBM HTTP Server configuration on our Windows machine to support SSL:

```
Listen 443
<VirtualHost tot136.itso.ibm.com:443>
SSLEnable
SSLServerCert TOT136
SSLClientAuth Required
SSLStashfile "c:\Program Files\IBMHTTPServer\key.sth
</VirtualHost>
Keyfile "C:\Program Files\IBMHttpServer/key.kdb
```

17.6.5 WebSphere configuration

We added the HTTP Server SSL port to the list of hosts included in our application virtual host and modified the plugin-cfg.xml file to contain the following lines:

```
<VirtualHost Name="tot136.itso.ibm.com:443"/>
<Transport Hostname="wtsc59.itso.ibm.com" Port="9445" Protocol="https">
  <Property name="keyring" value="C:\IBMHttpServer/plugin.kdb"/>
  <Property name="stashfile" value="C:\IBMHttpServer/plugin.sth"/>
</Transport>

<VirtualHost Name="tot136.itso.ibm.com:443"/>
```

17.7 Security validation for the WebSphere HTTP plug-in on distributed platforms: Test cases

This section describes the security validation for WebSphere HTTP plug-in on distributed platforms.

17.7.1 Test case WN1: Authentication through WebSphere HTTP plug-in on Windows using HTTP transport

We were able to validate authentication using the WebSphere HTTP plug-in on distributed platforms with base Application Servers, for basic, form-based, and certificate authentications.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet principal on the first page of the application. For all of the tests, we logged on a first server (Server 1 in the test case table) and looked at the Servlet Security Info table to check that the principal was the user ID entered at the security prompt. Figure 17-25 shows the Servlet Security Info table in SWIPE.

Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 17-25 Servlet Security Info table from SWIPE

According the authentication method, we used the URL listed in Table 17-9.

Table 17-9 URLs to validate security with WebSphere HTTP plug-in

Authentication method	Swipe relative URL
Basic	/IBMEBizWeb/secure/EJBCaller
From based	/IBMForms/secure/EJBCaller
Certificate	/IBMClientSSL/secure/sslEjbCaller

Table 17-10 Windows WebSphere HTTP plug-in Authentication Login Test Case 1 (WN1)

WN3 config: WebSphere base, IBM HTTP Server Windows 2000, ICSF, RACF, HTTP transport					
Transport	**Authentication method**	**Mutual authentication with SSL**	**Input**	**Output**	
			User ID	**Principal**	**Comments**
HTTP transport	Basic	No	stfan	stfan	
	Form	No	stfan	stfan	
	Certificate	No	cert	stfan	

17.7.2 Test case WN2: Authentication through distributed HTTP server with mutual SSL authentication

We were able to validate authentication using the WebSphere HTTP plug-in on distributed platforms with base Application Servers, for basic, form-based, and certificate authentications.

How SWIPE validates the security context

SWIPE validates the security context for this test case by displaying the servlet principal on the first page of the application. For all of the tests, we logged on a first server (Server 1 in the test case table) and looked at the Servlet Security Info table to check that the principal was the user ID entered at the security prompt. Figure 17-26 shows the Servlet Security Info table in SWIPE.

Remote Userid	STFAN
principalId	STFAN
Access to role: Worker	true

Figure 17-26 Servlet Security Info table from SWIPE

Based on the authentication method, we used URL listed in Table 17-11.

Table 17-11 URLs to validate security with WebSphere HTTP plug-in

Authentication method	Swipe relative URL
Basic	/IBMEBizWeb/secure/EJBCaller
From based	/IBMForms/secure/EJBCaller
Certificate	/IBMClientSSL/secure/sslEjbCaller

Table 17-12 Windows WebSphere HTTP plug-in Authentication Login Test Case 2 (WN2)

WN2 config: WebSphere base, IBM HTTP Server Windows 2000, ICSF, RACF, HTTPS transport					
Transport	**Authentication method**	**Mutual authentication with SSL**	**Input**	**Output**	
			User ID	**Principal**	**Comments**
HTTPS transport	Basic	Yes	stfan	stfan	
	Form	Yes	stfan	stfan	
	Certificate	Yes	cert	stfan	

18

EJB container security

This section looks at the EJB container from a security perspective.

In 18.1, “EJB container authentication protocols” on page 606 we look at how WebSphere Application Server for z/OS V5 compares to WebSphere Application Server for z/OS V4.01.

Then in 18.2, “Common Secure Interoperability Version 2 (CSIv2)” on page 608 we look at the new support in WebSphere Application Server for z/OS V5 for CSIv2, which are the open standard protocols that provide secure interoperability between application servers even when running on different platforms.

Finally, in 18.4, “zSAS authentication protocol” on page 627, we look briefly at the authentication methods that were provided in WebSphere Application Server for z/OS V4.01 and remain supported in WebSphere Application Server for z/OS V5. These are now collectively referred to as the zSAS protocol.

18.1 EJB container authentication protocols

When global security is enabled, the EJB container enforces access control on EJB method invocations. Authentication takes place regardless of whether a method permission is defined for the specific EJB method. If method permissions are defined in the application's EJB deployment descriptor, an authorization check will be performed to ensure that the authenticated user can execute the method.

It is the job of the authentication protocol during a method invocation to compare the server authentication requirements (determined by the object IOR) with the client authentication requirements (determined by the client configuration) and come up with an authentication policy suitable for both that client and server pair.

The authentication policy makes the following decisions, among others, which are all based on the client and server configurations:

- What kind of connection can you make to this server, SSL or TCP/IP?
- If Secure Sockets Layer (SSL) is chosen, how strong is the encryption of the data?
- If SSL is chosen, should the client be authenticated using client certificates?
- Should the client be authenticated using a user ID and password? Is there an existing credential to use?
- Should the client identity be asserted to downstream servers?
- Given the configuration of the client and server, should a secure request proceed?

You can configure both protocols (zSAS and CSIv2) to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for zSAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and zSAS. If the client supports both and the server supports both, CSIv2 is used. However, if the server supports zSAS (for example, it is a previous WebSphere Application Server release) and the client supports both, the client chooses zSAS for this request, because the zSAS protocol is what both have in common. Choose a protocol by specifying the com.ibm.CSI.protocol property on the client side and configuring through the administrative console on the server side.

18.1.1 WebSphere Application Server for z/OS V4 versus V5

In WebSphere Application Server for z/OS Version 4.01, you could authenticate to the EJB container in a number of ways. The authentication methods were selected when you defined the J2EE application server using the Systems

Management End User Interface (SMEUI). In WebSphere Application Server for z/OS V5 there is now support for Common Secure Interoperability (CSI) V2 (CSI v2) which is an open standard defining how systems can interoperate in terms of security.

The authentication methods available in WebSphere Application Server for z/OS V4.01 are still available in WebSphere Application Server for z/OS V5. They are:

- Basic authentication
- Client certificates
- Kerberos
- User ID/password
- User ID/PassTicket
- Identity assertion

In order to be able to talk about these authentication methods collectively and to distinguish them from the new CSIv2 protocol, the term “zSAS Protocol” is used. Note that in WebSphere distributed the term SAS (Secure Association Service) is used to describe an authentication protocol for the EJB container, but it is not correct to assume that zSAS is the WebSphere Application Server for z/OS version of SAS. Both SAS and zSAS provide authentication protocols for the EJB container, but they do so in different ways.

The differences between SAS and zSAS are one of the reasons why it was hard to provide security interoperability between J2EE applications on different platforms. In WebSphere Application Server for z/OS Version 5 CSIv2 makes IIOP-based security interoperability possible.

The features of zSAS and CSIv2 are compared in Table 18-1.

Table 18-1 zSAS versus CSIv2 feature comparison

zSAS features	**CSIv2 features**
SSL	SSL, TCP/IP choice
Identity assertion	Identity assertion (expanded)
Kerberos authentication	No Kerberos authentication
Stateful required	Stateful/stateless choice
Authentications on nonexistent() flow	nonexistent() call not used
IBM proprietary protocol	More flexible configuration (claim/perform, required/supported)
BasicAuth client login with user ID and password	Authentication retry

CSIv2 is implemented in WebSphere with more features than SAS and is considered the strategic protocol, so we strongly recommend that you use CSIv2.

Within the EJB container an authenticated user principal can access the requested EJB method if it has one of the required security roles. EJB application code can use the EJBContext methods isCallerInRole and getCallerPrincipal.

In WebSphere Application Server for z/OS Version 5, EJB application code also can use the JAAS programming model to perform JAAS login and WSSubject doAs and doAsPrivileged methods. The code in the doAs and doAsPrivileged PrivilegedAction block executes under the Subject identity. Otherwise, the EJB method executes under either the RunAs identity or the caller identity, depending on the RunAs configuration. The J2EE RunAs specification is at the enterprise bean level. When RunAs identity is specified, it applies to all bean methods. The method level IBM RunAs extension introduced in WebSphere Application Server for z/OS Version 4.01 is still supported in this version.

18.2 Common Secure Interoperability Version 2 (CSIv2)

In this section, we explain what Common Secure Interoperability is and how to enable it, and give some scenarios in which CSIv2 is most commonly run.

18.2.1 Overview

The Common Secure Interoperability (CSI) security specification is defined by the OMG (see http://www.omg.org). It is currently in its second version. CSIv2 is part of the J2EE 1.3 requirements and addresses security for interoperability. It is the successor to the various authentication methods of WebSphere Application Server for z/OS Version 4.01 that are now referred to collectively as zSAS protocols. Table 18-1 on page 607 shows us a comparison of the features between CSIv2 and zSAS.

The CSIv2 specification is defined as an open secure interoperability framework to support different secure interoperability service layers. The CSIv2 security service protocol has authentication, attribute, and transport layers.

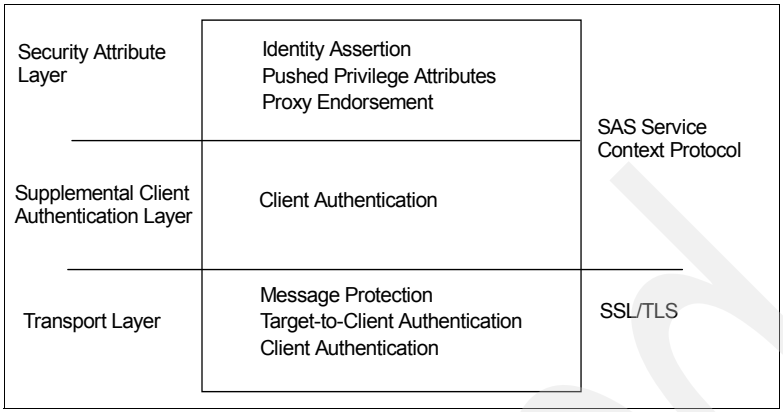


Figure 18-1 CSIv2 security architecture

In using CSIv2, conformance level 0 is implemented. This means that it must pass SUN J2EE Compliance Test Suite (CTS), uses Security Attribute Service (SAS) protocol, IORs and contexts, supports SSL 3.0 and TLS 1.0, and it must support stateless security. In stateless security, the security context gets sent always, leading the server to authenticate every time the request comes in. With stateful sessions the first method request between a client and a server is authenticated. All subsequent requests (or until the credential token expires) reuse the session. Performance is optimum when using stateful sessions.

Security Attribute Service (SAS)

As we can see from Figure 18-1, the SAS protocol is made up of two layers:

- The authentication layer is used to perform client authentication where sufficient authentication could not be accomplished in the transport.
- The attribute layer can be used by a client to deliver security attributes, such as identity and privilege, to the target where they can be applied in access control decisions.

> **Note:** z/OS only supports identity assertion at this time (conformance level 0). Privilege attributes are not supported.

The attribute layer also provides the means for a client to assert identity attributes that differ from the client's authentication identity (as established in the transport or SAS authentication layers). This identity assertion capability is the basis of a general-purpose impersonation mechanism that makes it possible for an intermediate to act on behalf of some identity other than itself. This can improve the performance of a system because the authentication of

a client is relatively expensive. The server can validate the request by checking its trust rules.

18.2.2 CSIv2 authentication in WebSphere Application Server

The following Common Secure Interoperability Version 2 features are available for WebSphere:

- Transport layer protection
- Supplemental client authentication
- Identity assertion

Transport layer protection

CSIv2 is only supported from a Java client and not from a C++ client. Transport layer protection can use SSL client authentication. One benefit of SSL client certificate authentication is that it optimizes authentication performance, because an SSL connection is typically created anyway. The extra overhead of sending the client certificate is minimal. While the client-side request interceptor performs no activity, the server-side request interceptor maps the certificate to a credential. One disadvantage to this type of authentication is the complexity of setting up the keystore file on each client system.

Supplemental client authentication

Supplemental client authentication authenticates credential information and sends that information across the network so that a receiving server can interpret it. Sending authentication information across the network using a token is considered message layer authentication because the data is sent along with the message inside a service context.

To specify that we want to use authentication over the message layer, the following parameters must be set:

- com.ibm.CSI.performClientAuthenticationRequired=(true or false)
- com.ibm.CSI.performClientAuthenticationSupported=true

Setting com.ibm.CSI.performClientAuthenticationRquired=true implies that it must be done for every request.

Identity assertion

Identity assertion is the process wherein the invocation credential is asserted to the downstream server during a call.

When a client authenticates to a server, the *received credential* is created. When authorization checks the credential to see that access is allowed, it also sets the *invocation credential* so that if the component calls an EJB method located on another server, the invocation credential can be used as the identity with which to

invoke the downstream method. Depending on the run-as mode for the EJB, the invocation credential will be set as:

- The originating client identity
- The server's identity
- A specified different identity

Regardless of the identity that is set, when identity assertion is enabled it is the invocation credential that is asserted to the downstream server. If the nonauthenticated_client_allowed variable is set to true, the default user ID is going to be used.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server's identity is sent in the client authentication token. Both are needed by the receiving server to accept the asserted identity. The receiving server does the following to accept it:

1. Determines whether the sending server's identity is on the trusted principal list of the receiving server. That is, it determines whether the sending server is the one that is allowed to send an identity token to the receiving server. The trust is verified through CBIND checks. The server determines whether the sending server can send an identity token to the receiving server.
2. Makes sure that it is truly the sending server by authenticating it. The server is authenticated by comparing the user ID and password from the sending server to the receiving server, or it might require a real authenticate call. If the credentials of the sending server are authenticated and on the trusted principal list, the server proceeds to evaluate the identity token. If the sending server's credentials are authenticated and on the trusted principal list, evaluation of the identity token can proceed. There are four formats of identities that can be present in an identity token:
   - Principal name
   - Distinguished name
   - Certificate chain
   - Anonymous identity

   The WebSphere Application Servers that receive authentication information typically support all four identity types. The sending server decides which one will be chosen based on how the original client authenticated. For example, if the client uses SSL client authentication to authenticate to the sending server, the identity token to the downstream server will contain the certificate chain. This is important because it allows the receiving server to perform its own mapping of the certificate chain. It enables a high level of interoperability with other vendors and platforms.

3. After the identity format is understood and parsed, the identity is simply mapped to a credential. All identity token types map to a user ID in the active user registry. For instance, a Distinguished Name (DN) from a client certificate can be mapped to an SAF user ID with filters.

Some user registry methods are called to gather additional credential information used by authorization. In a stateful server, this is done one time for the sending server/receiving server pair where the identity tokens are the same. Subsequent requests will be made with a session ID.

Note: This section describes identity assertion using CSIv2, which is available on all WebSphere Application Server V5 platforms. Another form of identity assertion, which is implemented through zSAS, is available solely on the z/OS and OS/390 platforms. This capability is described in *z/OS WebSphere and J2EE Security Handbook*, SG24-6846.

18.2.3 Enabling CSIv2

There are a few things that need to be done to enable CSIv2 security. There are configuration considerations on the client side and configurations on the WebSphere side.

In order for a secure Java client to run with CSIv2, the client requires properties to determine how to perform the security. These properties are set somewhere on the z/OS machine and referenced or called by the Java client.

Example 18-1 CSIv2 properties file

```
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
com.ibm.CSI.performTransportAssocSSLTLSSupported=true
com.ibm.CSI.performTLClientAuthenticationRequired=false
com.ibm.CSI.performTLClientAuthenticationSupported=false
com.ibm.CSI.performClientAuthenticationRequired=false
com.ibm.CSI.performClientAuthenticationSupported=true
com.ibm.CSI.performMessageConfidentialityRequired=false
com.ibm.CSI.performClientAuthenticationtype=SAFUSERIDPASSWORD
com.ibm.CSI.performSSL.Keyring=WASKEYRING
com.ibm.CSI.Rem.Userid=WASDFTU
com.ibm.CSI.Rem.Password=WASDFTU
```

The distributed world uses the sas.client.props file in the <WAS_HOME>/properties directory to specify these values. You can do the same or just create a file similar to Example 18-1. For our purposes we will call the file CSIv2Properties and will place it in the following directory:

```
/WebSphere/V5ROMO/CSIv2/CSIv2Properties
```

It doesn't matter what property file you use. However, we must specify the com.ibm.CORBA.ConfigURL property to point to the file. It will not look for any file by default.

Important: Depending on what scenario you want to configure, the values in the property file will need to be adjusted.

CSIv2 configuration file unleashed

As we have seen above, there are a number of different properties that need to be configured before running with CSIv2.

performTransportAssocSSL

These properties are needed to enable CSIv2 to run with SSL. "Required" set to true means we must use CSIv2 with SSL. The property term "Supported" simply says we can use this, but we do not have to. The correct value for the properties is either true or false.

```
com.ibm.CSI.performTransportAssocSSLTLSSupported
com.ibm.CSI.performTransportAssocSSLTLSRequired
```

performTLClientAuthentication

There are other property files that allow SSL Client Authentication. The same principles apply for the Required and Supported properties. The correct value for the two properties is either true or false.

```
com.ibm.CSI.performTLClientAuthenticationRequired
com.ibm.CSI.performTLClientAuthenticationSupported
```

performClientAuthentication

Basic authentication also requires property files for running in CSIv2. The correct value for these properties is either true or false.

```
com.ibm.CSI.performClientAuthenticationRequired
com.ibm.CSI.performClientAuthenticationSupported
```

performMessageConfidentiality

The next property determines if 128-bit ciphers must be used for ssl connections. This property is only valid if SSL is enabled.

```
com.ibm.CSI.performMessageConfidentialityRequired
```

performClientAuthenticationtype

To determine what authentication method you want to use, the following property is used:

```
com.ibm.CSI.performClientAuthenticationtype
```

A typical value for com.ibm.CSI.performClientAuthenticationtype is SAFUSERIDPASSWORD, which simply means user ID and password. If we use user ID and password, we need to specify those values.

```
com.ibm.CSI.Rem.Userid
com.ibm.CSI.Rem.Password
```

Note: SAFUSERIDPASSWORD is the only value supported at the moment.

performSSL.Keyring

When using ssl, there needs to be a definition of the key ring being used. The property for that is as follows:

```
com.ibm.CSI.performSSL.Keyring
```

RACF profile for CSIv2 realm

There needs to be a realm in which CSIv2 is run. To do this, issue the following command:

```
RDEFINE REALM SAFDFLT APPLDATA("WTSC59.ITSO.IBM.COM")
```

Important: We chose to use wtsc59.itso.ibm.com as our realm. This is our system name. Make sure you supply the correct value for your system configuration.

CSIv2 administrative console configuration

Under the Security tag on the left side on the administrative console, there is a section labeled Authentication Protocol. Click it to see four CSIv2 options.

The first is called CSIv2 Inbound Authentication. This specifies properties for incoming requests for the server. It can be seen in Figure 18-2 on page 615 and Figure 18-3 on page 616.

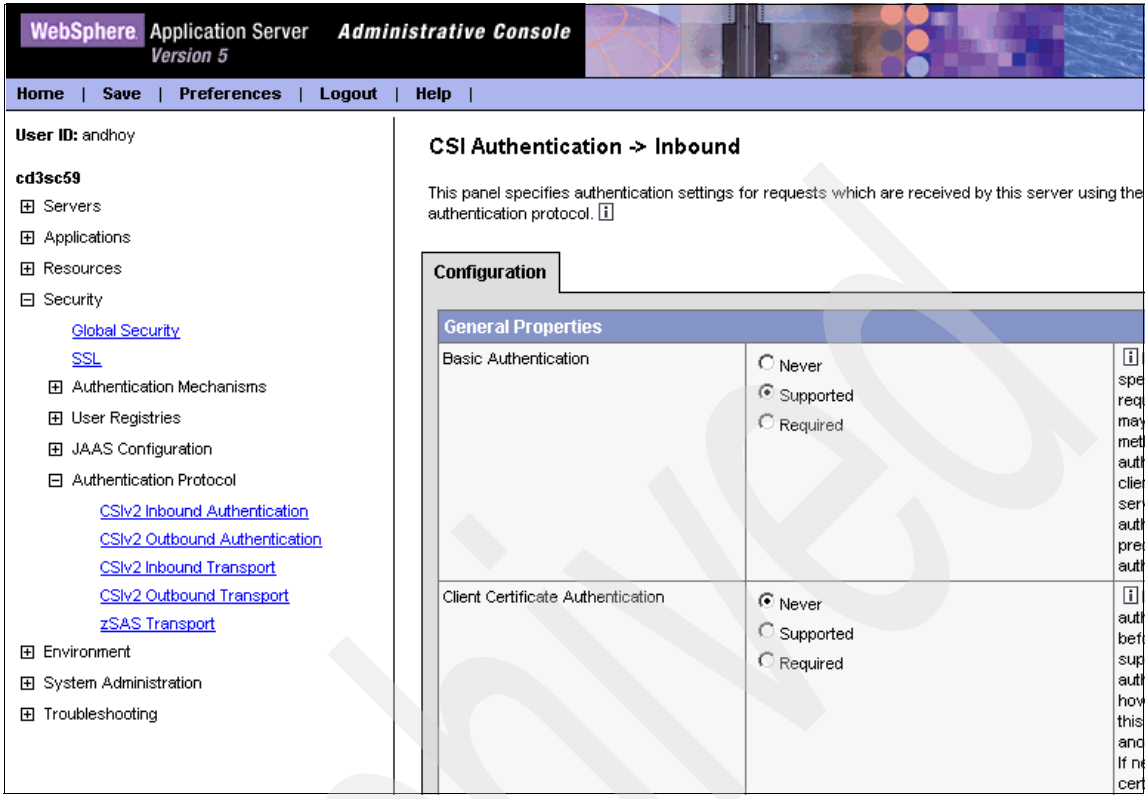


Figure 18-2 Enabling CSIv2 Inbound Authentication for Scenario1

Under CSIv2 Inbound Authentication in the General Properties section there are five properties that need to be addressed:

- Basic Authentication

  If Required is selected, the client must specify a user ID and password. If Supported is selected, the client may supply a user ID and password, but does not have to. If Never is selected, the server will not accept a user ID and password.

- Client Certificate Authentication

  This has the same three choices as Basic Authentication with all the same meanings. If Client Certificate Authentication and Basic Authentication are both selected, Basic Authentication takes precedence.

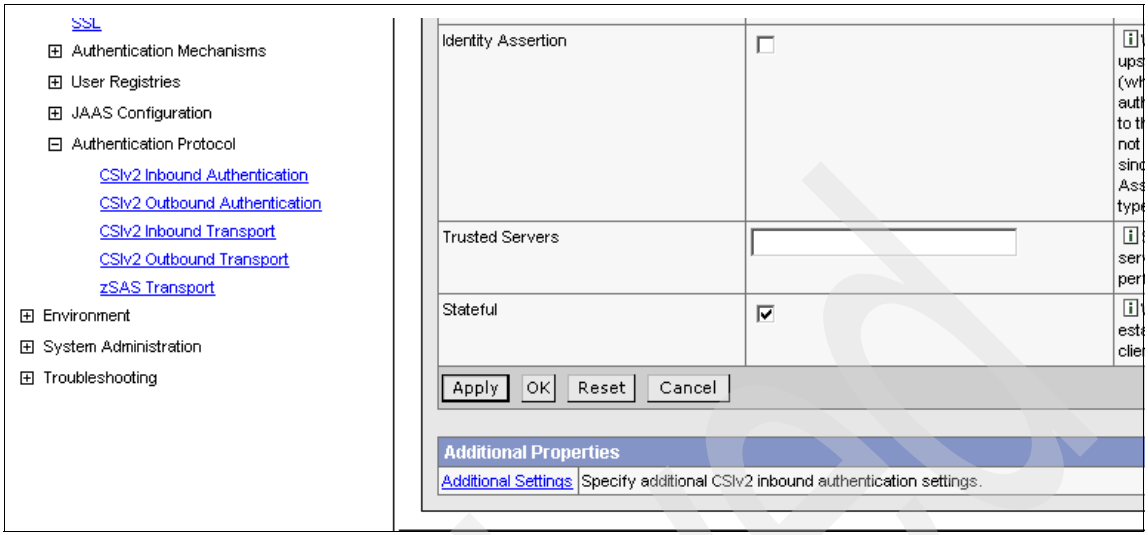


Figure 18-3 Enabling CSIv2 Inbound Authentication for Scenario1

- Identity Assertion

  Selecting the box enables this feature. Identity Assertion takes precedence over the other authentication mechanisms.

- Trusted Servers

  z/OS does not support this. Leave this box blank.

- Stateful

  When selected, stateful sessions are established for client and server. This check box is ignored. Stateful is always supported.

The next protocol that we will look at is CSIv2 inbound transport. This specifies the SSL information for CSIv2. Figure 18-4 on page 617 shows the configuration options for this.

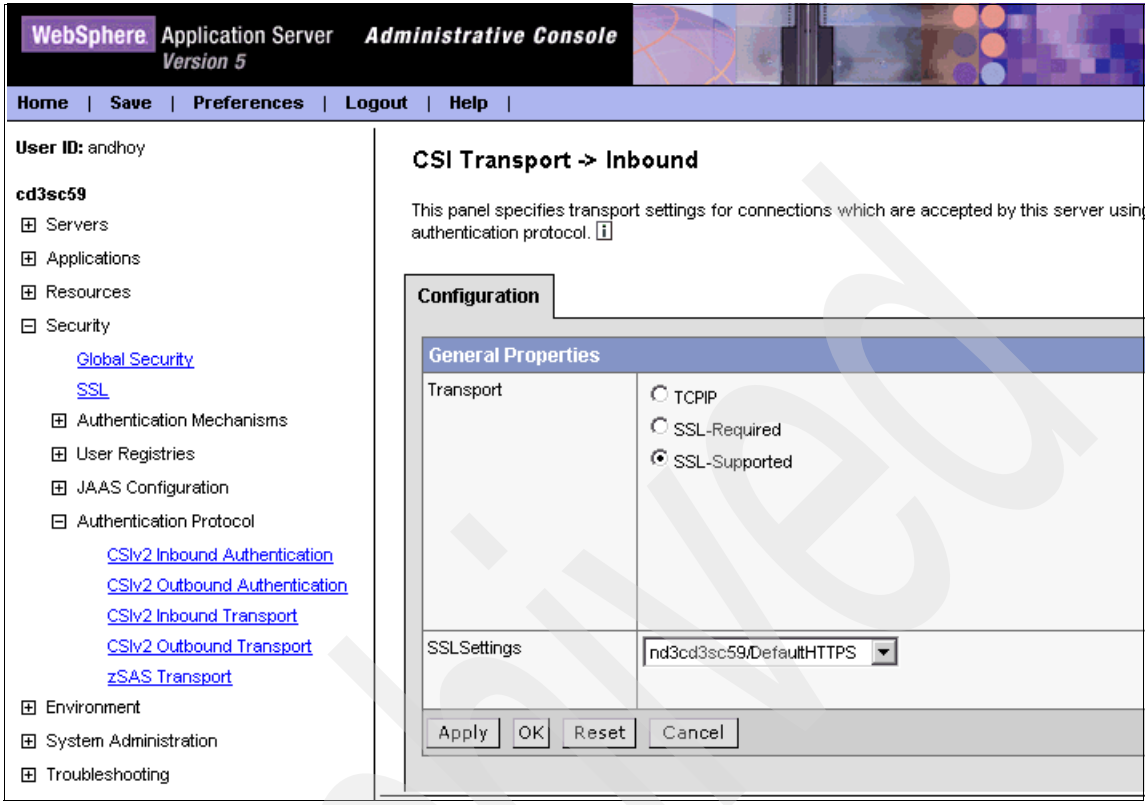


Figure 18-4 Enabling CSIv2 Inbound Transport for Scenario1

There are two properties that are to be set under the General Properties for CSIv2 inbound transport:

- Transport

  There are three choices for the type of transport that you want.

  - TCPIP: When this is selected, only the TCP/IP listener port will be available for requests.
  - SSL-Required: If this is chosen, only the SSL listener port will be available for requests.
  - SSL-Supported: This allows both TCP/IP and SSL listener ports to be available for requests.

- SSLSettings

  Here is where we specify the SSL Repertoire that we want. The SSL Repertoire is defined under the SSL in the Security panel on the administrative console.

The third protocol is CSIv2 Outbound Authentication. Under General Properties in the Configuration section, there are four properties that need to be set. Figure 18-5 and Figure 18-6 on page 619 display these properties.

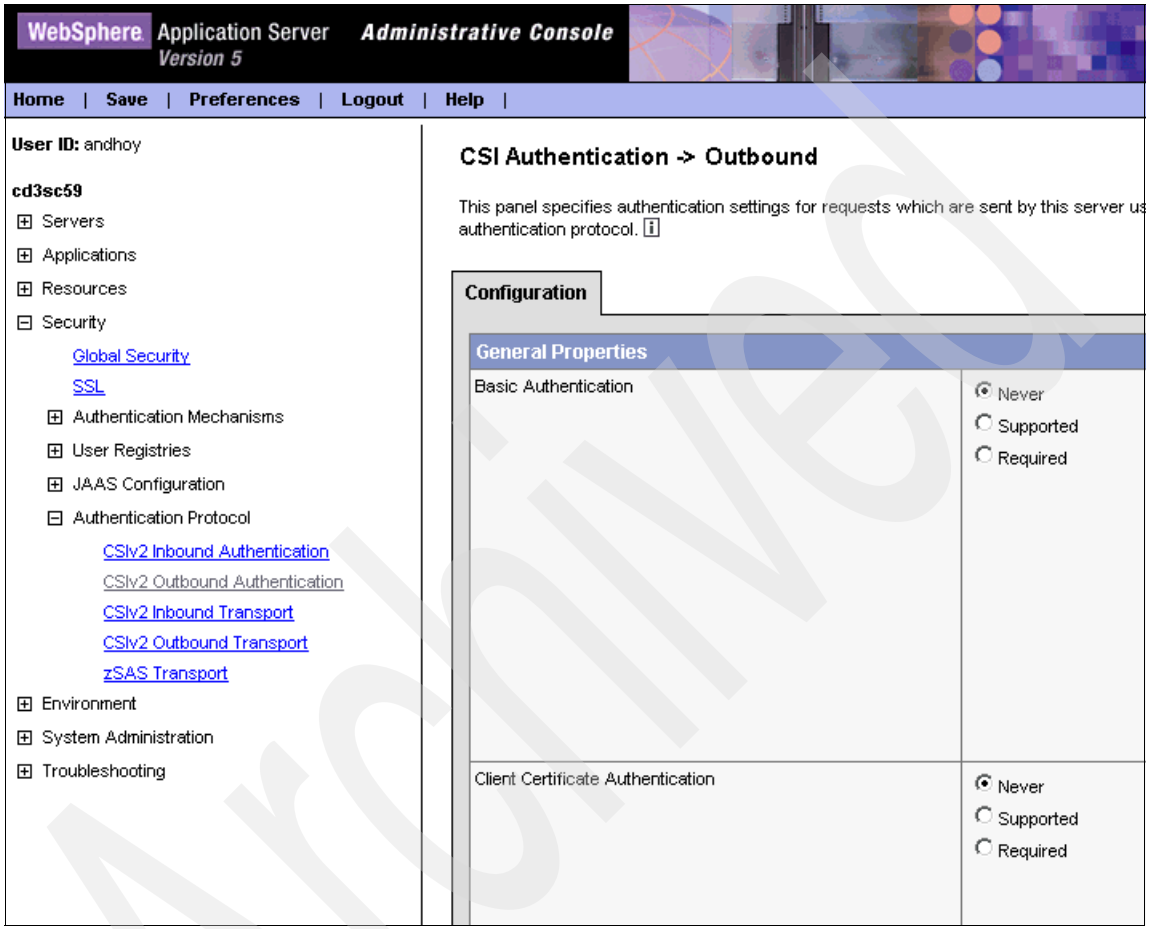


Figure 18-5 Enabling CSIv2 Outbound Authentication for Scenario1

- Basic Authentication

  This has three options. Basic Authentication is not supported for outbound servers.

- Client Certificate Authentication

  This has the same three choices as Basic Authentication. Basic Authentication takes precedence over SSL Client Certificate Authentication.

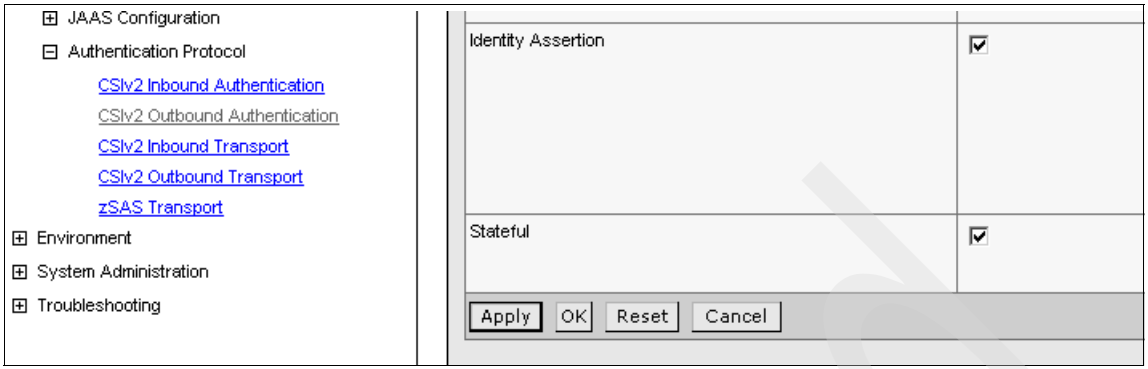


Figure 18-6 Enabling CSIv2 Outbound Authentication for Scenario1

- Identity Assertion

  When enabled, the server can assert received client identities to downstream servers as a method of authentication.

- Stateful sessions

  Stateful is always supported, so the check box is ignored.

The fourth CSIv2 authentication protocol is CSIv2 outbound transport. This specifies which transport settings will be used for the server. Figure 18-7 on page 620 shows this protocol.

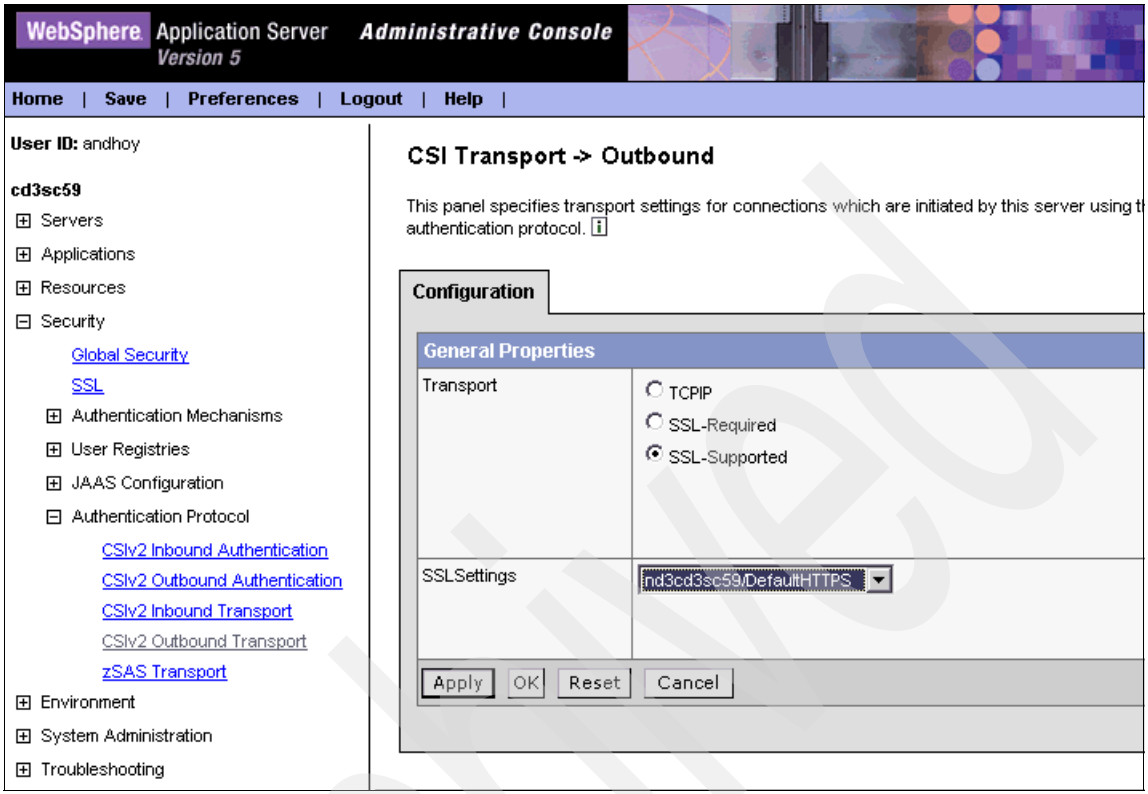


Figure 18-7 Enabling CSIv2 Outbound Transport for Scenario1

There are two properties that need to be addressed:

- Transport

  There are three choices here:

  - TCPIP: This specifies that only a TCP/IP connection will be open to a downstream server.
  - SSL-Required: This specifies that only an SSL connection will be used.
  - SSL-Supported: This specifies that both a TCP/IP and a SSL connection will be open to a downstream server.

- SSLSettings

  This is a list of SSL repertoires that have specific SSL settings. Here we want to select the one that we have created for our use.

18.3 CSIv2 solution scenarios

In this section, we provide a few user scenarios in which CSIv2 is typically used.

18.3.1 Scenario 1: Basic authentication and identity assertion

This scenario involves a Java client accessing a secure EJB on server1with the user “tom”. The EJB on server1 accesses another EJB on server2. Here, we use identity assertion to propagate the identity of “tom” to server2. Server2 trusts that “tom” is already authenticated by server1. To gain this trust, the identity of server1 also flows to server2 simultaneously, and server2 validates the identity to see if it is a trusted server principal. In the case of z/OS, this means that the identity passed by server1 must have CONTROL access to the CB.BIND.<server name> SAF profile in the CBIND class. Server2 also authenticates server1 using the password provided. Figure 18-8 displays the setup for the scenario.

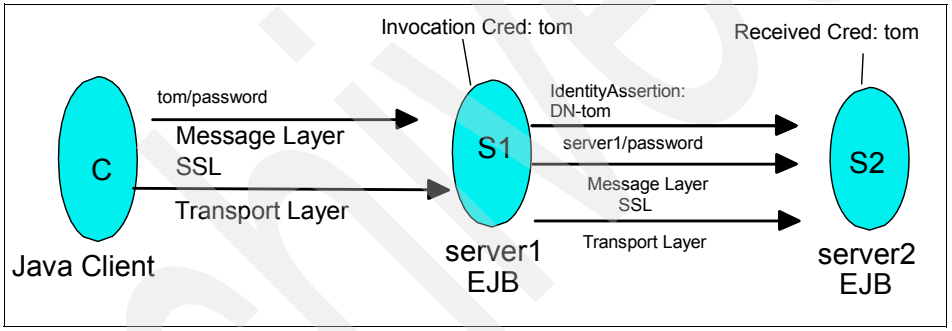


Figure 18-8 Basic authentication and identity assertion

Enabling CSIv2 with basic authentication and identity assertion

This section describes how to enable CSIv2 with basic authentication and identity assertion.

Client-side configuration

The client needs configuration in order to complete the scenario. It needs message layer authentication with a Secure Sockets Layer (SSL) transport. The first thing to do is to have the Java client point to the property file. To do this we must specify the following:

```
com.ibm.CORBA.ConfigURL=file:/WebSphere/V5ROMO/CSIv2/CSIv2Properties
```

In the CSIv2Properties file, specify the following to enable SSL:

```
com.ibm.CSI.performTransportAssocSSLTLSSupported=true
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
```

Next we want to enable client authentication at the message layer:

```
com.ibm.CSI.performClientAuthenticationRequired=false
com.ibm.CSI.performClientAuthenticationSupported=true
```

The third thing is to set the user ID and password. For our purposes we will use tom and tom1 for the user ID and password, respectively.

```
com.ibm.CSI.Rem.Userid=tom
com.ibm.CSI.Rem.Password=tom1
```

Now set the property to determine the authentication mechanism:

```
com.ibm.CSI.performClientAuthentication=SAFUSERIDPASSWORD
```

That is all that is needed on the client side.

Server-side configuration

Note that in this scenario, we have two servers. Server1 is configured for incoming requests to support message layer authentication and incoming connections to support SSL without client certificate authentication. It is also configured for outgoing requests for identity assertion. Here are the steps for setting up the server:

1. Log on to the administrative console.
2. On the left panel, select **Servers**.
3. Select **Application Servers**.
4. Select your server under Configuration.
5. Select **Server Security** under Additional Properties.
6. Select **CSIv2 Inbound Authentication** under Additional Properties. Use the following values:
   - Basic Authentication: Select **Supported**.
   - Client Certificate Authentication: **Select Never**.
   - Identity Assertion: Leave cleared.
   - Trusted Server: Leave blank.
   - Stateful: Select the option.
7. Select **CSIv2 Outbound Authentication** under the same directory structure. Use the following values:
   - Basic Authentication: Select **Supported**.
   - Client Certificate Authentication: Select **Never**.
   - Identity Assertion: Select this option to enable it.

- Stateful: (Optional) Select this option.

8. Select **CSI Transport Inbound** under the same directory structure. Use the following values:
   - Transport: Select **SSL-Supported**.
   - SSL Settings: Select the SSL Repertoire that was set up previously.
9. Select **CSI Transport Outbound**. Use the following values:
   - Transport: Select **SSL-Supported**.
   - SSL Settings: Select the SSL Repertoire that was set up previously.

Note: You can also configure these settings globally by selecting **Security** → **Authentication Protocol**.

Server2 needs to be configured in a similar fashion. Complete the following steps:

1. Log on to the administrative console.
2. On the left panel, select **Servers**.
3. Select **Application Servers**.
4. Select your server under Configuration.
5. Select **Server Security** under Additional Properties.
6. Select **CSIv2 Inbound Authentication** under Additional Properties. Use the following values:
   - Basic Authentication: Select **Supported**.
   - Client Certificate Authentication: Select **Never**.
   - Identity Assertion: Select this option.
   - Trusted Server: Leave blank.
   - Stateful: (Optional) Select this option.
7. (Optional) Select **CSIv2 Outbound Authentication** under the same directory structure. Use the following values:
   - Basic Authentication: Select **Never**.
   - Client Certificate Authentication: Select **Never**.
   - Identity Assertion: Leave this option cleared.
   - Stateful: Select this option.
8. Select **CSI Transport Inbound** under the same directory structure. Use the following values:
   - Transport: Select **SSL-Supported**.
   - SSL Settings: Select the SSL Repertoire that was set up previously.

9. (Optional) Select **CSI Transport Outbound**. Use the following values:

- Transport: Select **SSL-Supported**.
- SSL Settings: Select the SSL Repertoire that was set up previously.

Note: You can also configure these settings globally by selecting **Security** → **Authentication Protocol**.

18.3.2 Scenario 2: SSL and identity assertion

This scenario is very similar to scenario 1 and requires a configuration similar to the first scenario. We take a Web client that accesses a servlet on server1. It reaches that connection over TCP/IP or SSL and is challenged for a user ID and password. Server1 authenticates the user with the registry that is configured. The servlet then invokes a method on an EJB on server2. Figure 18-9 illustrates this scenario.

Note: In order for this scenario to work reliably, we needed to apply maintenance at the W510004 level.

Server-side configuration

The server-side configuration is similar to that in 18.3.1, “Scenario 1: Basic authentication and identity assertion” on page 621, except that for CSIv2 outbound authentication on server1 and CSIv2 inbound authentication on server 2, we selected the Client Certificate Authentication of **Supported** instead of **Never**. This allows server1 to be authenticated by server2 using server1’s digital certificate rather than the user ID and password, as illustrated in Figure 18-9.

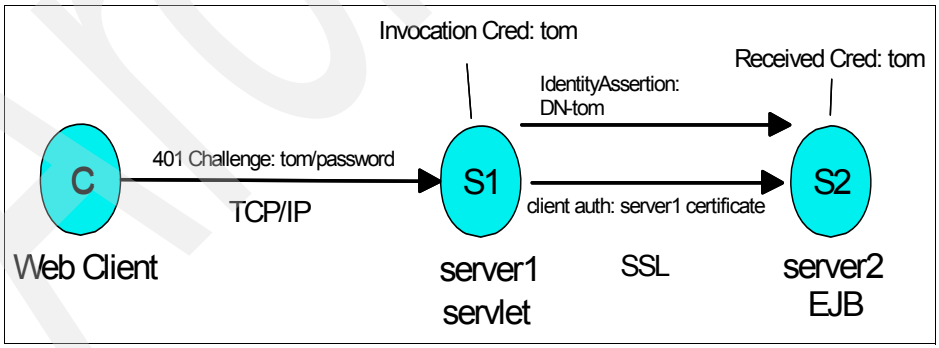


Figure 18-9 SSL and identity assertion

18.3.3 Scenario 3: Client certificates

This scenario is very similar to scenario 1. It requires only a few configuration changes. We take a Java client that accesses an EJB on server1. It reaches that connection over SSL using client certificates. Server1 authenticates the user with the registry that is being used. The servlet then invokes a method on an EJB on server2. Server2 trusts that “tom” is already authenticated by server1. To gain this trust, the identity of server1 also flows to server2 simultaneously, and server2 validates the identity to verify that it is a valid server principal. Server2 also authenticates server1using the password supplied. Figure 18-10 shows the configuration for this scenario.

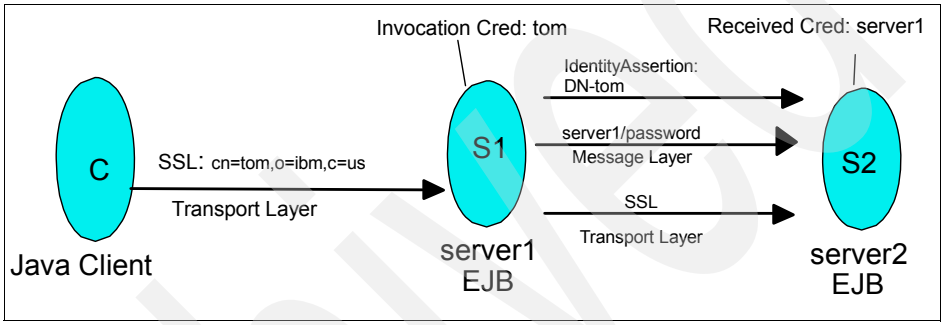


Figure 18-10 Client certificate scenario

Enabling CSIv2 with client certificates

To configure the client, we must do some of the same setup that was done in scenarios 1 and 2. First, point the Java client to the property file. To do this, specify the following:

```
com.ibm.CORBA.ConfigURL=file:/WebSphere/V5ROMO/CSIv2/CSIv2Properties
```

In the CSIv2Properties file, specify the following to enable SSL:

```
com.ibm.CSI.performTransportAssocSSLTLSSupported=true
com.ibm.CSI.performTransportAssocSSLTLSRequired=false
```

Next, enable client authentication at the message layer:

```
com.ibm.CSI.performClientAuthenticationRequired=false
com.ibm.CSI.performClientAuthenticationSupported=true
```

To configure the server, log on to the administrative console. First enable server1:

1. Log on to the administrative console if you have not already done so.
2. On the left panel, select **Servers**.

3. Select **Application Servers**
4. Select your server under Configuration.
5. Select **Server Security** under Additional Properties.
6. Select **CSIv2 Inbound Authentication** under Additional Properties. Use the following values:
   - Basic Authentication: Select **Never**.
   - Client Certificate Authentication: Select **Supported**.
   - Identity Assertion: Select this option.
   - Trusted Server: Leave blank.
   - Stateful: Select this option.
7. Select **CSIv2 Outbound Authentication** under the same directory structure. Use the following values:
   - Basic Authentication: Select **Supported**.
   - Client Certificate Authentication: Select **Never**.
   - Identity Assertion: Select this option to enable it.
   - Stateful: (Optional) Select this option.
8. Select **CSI Transport Inbound** under the same directory structure. Use the following values:
   - Transport: Select **SSL-Supported**.
   - SSL Settings: Select the SSL Repertoire that was set up previously.
9. Select **CSI Transport Outbound**. Use the following values:
   - Transport: Select **SSL-Supported**.
   - SSL Settings: Select the SSL Repertoire that was set up previously.

Note: You can also configure these settings globally by selecting **Security** → **Authentication Protocol**.

Set up server2:

1. Log on to the administrative console.
2. On the left panel, select **Servers**.
3. Select **Application Servers**.
4. Select your server under Configuration.
5. Select **Server Security** under Additional Properties.
6. Select **CSIv2 Inbound Authentication** under Additional Properties. Use the following values:
   - Basic Authentication: Select **Supported**.
   - Client Certificate Authentication: Select **Never**.

- Identity Assertion: Select this option.
- Trusted Server: Leave blank.
- Stateful: (Optional) Select this option.

7. (Optional) Select **CSIv2 Outbound Authentication** under the same directory structure. Use the following values:
   - Basic Authentication: Select **Never**.
   - Client Certificate Authentication: Select **Never**.
   - Identity Assertion: Leave this option cleared.
   - Stateful: Select this option.
8. Select **CSI Transport Inbound** under the same directory structure. Use the following values:
   - Transport: Select **SSL-Supported**.
   - SSL Settings: Select the SSL Repertoire that was set up previously.
9. Select **CSI Transport Outbound**. Use the following values:
   - Transport: Select **SSL-Supported**.
   - SSL Settings: Select the SSL Repertoire that was set up previously.

Note: You can also configure these settings globally by selecting **Security** → **Authentication Protocol**.

Why should I use this function? Authentication is required for requests that arrive over IIOP into the EJB container. CSIv2 is the strategic protocol for providing the secure authentication of Java clients and propagation of already-authenticated user IDs over IIOP from other application servers.

Unless your server must communicate with another server or client that does not support CSIv2, you should use CSIv2 in preference to the zSAS protocol.

18.4 zSAS authentication protocol

The zSAS protocol is the name given to the collection of authentication methods that were used in WebSphere Application Server for z/OS V4.01 and continue to be supported in WebSphere Application Server for z/OS V5.

Authentication is the process of establishing a client's identity and determining that this identity is indeed authentic and not an imposter that poses as the real client. Authentication in the Web container is mostly about authenticating a browser-based client, while authentication in the EJB container is about authenticating a J2EE application client that makes an RMI/IIOP request to run a method on an EJB in WebSphere. The authentication process is therefore very

much dependent on the nature of the client application making the request and the security features available to its runtime environment.

18.4.1 zSAS overview

Authentication requirements for Web applications are specified on a per application basis. Authentication requirements for EJB applications are specified differently. Here, the supported authentication methods are defined on a control region basis. The actual authentication method used is subject to negotiation between a client and a server or between two servers; which method is actually used depends on which methods are supported by both partners and on their preferences.

Although the J2EE specification has detailed information about the authentication mechanisms that must be implemented for the Web container, it does not specify authentication for the EJB container. It merely states that the EJB container must have a means of authentication principals. In many cases this is of little consequence, because EJBs are typically invoked by servlet code (where the caller has been authenticated by the Web container) and not by end users directly.

The WebSphere Application Server for z/OS Version 5 administrative console is used to specify the authentication methods that apply to a base server or to a node if running with a network Deployment configuration. To configure one or more zSAS authentication methods, navigate to **Security** → **zSAS Transport**. This is shown in Figure 18-11 on page 629.

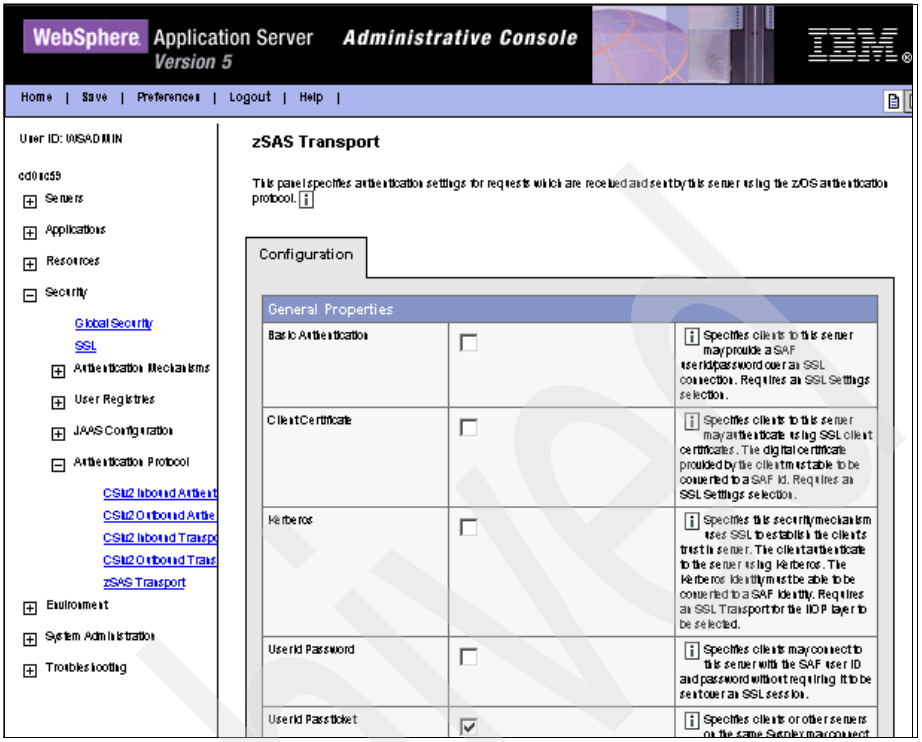


Figure 18-11 Selecting zSAS authentication protocols for the EJB container

When a connection is to be made, a method supported by both partners is negotiated. If the negotiation ends without establishing a common authentication method, no communication is established and the request fails (unless both partners allow no security to be used).

Apart from the differences in the way authentication requirements are determined, it should also be noted that different communication protocols are used for Web applications and EJB applications. Web applications use the HTTP protocol, while EJB applications use the Internet Inter-ORB Protocol (IIOP).

In the following, we introduce the authentication methods that can be used to authenticate EJB applications first before showing which methods can be used in the different environments.

18.4.2 zSAS authentication methods

The EJB container authentication methods that were available in WebSphere Application Server for z/OS V4.01 are still available in WebSphere Application Server for z/OS V5. They are:

- Basic authentication
- Client certificate
- Kerberos
- User ID/password
- User ID/PassTicket
- Identity assertion

Each of these is now described in more detail.

zSAS basic authentication, user ID/password and PassTicket

In 16.2.2, “Basic authentication” on page 507 we describe the principles of basic authentication in Web applications. In EJB applications, basic authentication is not really a challenge/response protocol. When a client and a WebSphere Application Server for z/OS EJB container agree to use basic authentication as their authentication method, the client needs to supply a RACF user ID (remote user ID) and password.

The difference between basic authentication and user ID/password authentication is that basic authentication in the EJB container takes place on an SSL connection, while the user ID/password authentication does not.

Where the user ID and password (or PassTicket) come from depends on the client. If the client is a Java application on z/OS, you can set the environment variables REM_USERID and REM_PASSWORD in the client environment file.

Basic authentication in the EJB container uses SSL type 1 authentication in which the WebSphere server proves its identity to the client by passing a digital certificate to the client. By authenticating the server’s certificate, the client can be sure it is talking to the correct server. Then the client sends the user ID/password to the server and the server uses these to authenticate the client.

In a variation of the basic authentication method, the Java client can use a RACF PassTicket instead of a password. This is possible with Java clients running on z/OS and OS/390 systems because RACF provides a PassTicket generator service so PassTickets can be readily used. If RACF PassTickets should be used by clients on other platforms, a PassTicket generation service would have to be implemented first.

A drawback of PassTickets is that the granularity of the time stamp involved in the creation of PassTickets is one second. PassTickets created within the same

one-second interval will be identical and RACF will reject the second and subsequent identical PassTicket as a replay attempt. Therefore, PassTickets in their original implementation cannot be used if the possibility exists that more than one PassTicket for the same user ID and application name might be needed within the same second. It is possible to disable replay checking for PassTickets if the PTF for APAR OW44393 is installed, but allowing this can create a security exposure.

Why should I use this authentication mechanism? You would only use these zSAS authentication methods if you need to communicate from a client that does not support CSIv2. In addition, the client should probably be on z/OS. Because basic authentication requires user ID and password, your Java client will need some way to manage password expiry/change and so on. If the client is on z/OS, the assumption is that the user has some other means (perhaps through TSO) of maintaining their password. If the client is remote, user ID and password will be set in a file somewhere and you have the problem of keeping this secure and managing password expiry. RACF PassTickets only applies to clients on z/OS capable of asking RACF to generate a PassTicket.

Enabling basic authentication, user ID/password, and RACF PassTickets

You enable basic authentication in the EJB container using the WebSphere administrative console. Navigate to **Security** → **Authentication Protocol** → **zSAS Transport** to reach the zSAS Transport Configuration page. Then, select the **Basic Authentication** option. You also need to select an SSL repertoire in the drop-down list box at the bottom of the zSAS Transport configuration page.

The client must also support basic authentication for this to be selected by the negotiation of security protocols. In addition, if client and server both support CSIv2 and basic authentication, CSIv2 will be used.

User ID/password and PassTicket authentication is enabled by checking the respective boxes further down the zSAS Transport configuration page.

zSAS certificate-based authentication

Authentication using X.509 user certificates is briefly described in 16.2.7, “WebSphere authentication mechanisms” on page 531. Because client authentication with a certificate is part of the SSL protocol, an SSL connection is a prerequisite for certificate-based authentication.

When using an SAF-based user registry, WebSphere on z/OS needs to use a key ring in RACF for its certificates and you will probably want to map these to

RACF user IDs, which will become the identity associated with the Java thread in the EJB container. A key database in the HFS (created by GSKKYMAN or iKeyman) is not supported for WebSphere Application Server for z/OS certificates unless you are using a non-SAF user registry.

A Java client on z/OS or a WebSphere Application Server for z/OS server acting as a client to another server (by calling an EJB) needs to be the owner of a RACF key ring that has a client certificate (for itself) and a CA root certificate (for the target server's certificate) connected. A server receiving requests from clients and other servers needs to be the owner of a key ring that has its own server certificate and the CA root certificates for all accepted client certificates (from the clients and requesting servers) connected.

A remote Java client or WebSphere server that is not using an SAF user registry (for example, WebSphere on distributed platforms) can store its client certificate and the CA root certificate of the target WebSphere Application Server for z/OS server in a .jks file using iKeyman.

Note: The WebSphere Application Server for z/OS control region user ID needs to be the owner of the RACF key ring and needs to be associated with all client and server certificates (parameter ID(...)) in the key ring.

Servers can be hybrid in a sense that they accept requests from clients (and other servers) but also call EJBs in other servers. In this case, the server needs a key ring that has connections to both the CA root certificates for the client certificates of the server's callers and the CA root certificates for the server certificates of the servers that the server itself calls.

Why should I use this authentication mechanism? You would only use the zSAS SSL authentication when your client does not support CSIv2. SSL provides authentication that includes confidentiality of the connection and is therefore to be preferred over basic authentication.

Enabling certificate-based security

You enable client certificate authentication in the EJB container using the WebSphere administrative console. Navigate to **Security** → **Authentication Protocol** → **zSAS Transport** and then select the **Client Certificate** option. The client must also support client certificate authentication for this to be selected by the negotiation of security protocols. In addition, if client and server both support CSIv2 and zSAS client certificate authentication, CSIv2 will be used.

zSAS Kerberos authentication

WebSphere Application Server for z/OS and OS/390 supports Kerberos authentication for EJB applications, but not for Web applications. In this implementation, Kerberos and the GSS-API are used for authentication only, and the GSS-API functions for data integrity and privacy are not used. The session between client and server is protected using the SSL protocol. Before discussing the details of the Kerberos over SSL implementation, we briefly introduce the principles of Kerberos authentication.

The Kerberos system consists of three components: a client, a server, and a trusted third party, which is also known as a Key Distribution Center (KDC). KDC interacts with both a client and a server to accept the client's request, authenticate its identity, and issue tickets to it.

The domain served by a single KDC is referred to as a realm. A principal identifier is used to identify each client and server in a realm. The principal name is uniquely assigned for all clients and servers by the Kerberos administrator. All principals must be known to the KDC.

Although the Kerberos protocol consists of several sub protocols, three exchanges would be the most interesting for most readers, as shown in Figure 18-12 on page 634.

The first exchange is taking place between a client and the authentication server (AS), in which a client asks the AS that knows the secret keys of all clients in the realm to authenticate itself and gives it a ticket called ticket-granting ticket (TGT). The TGT is used to get a secret shared with an application server the client wants to access.

Upon receiving the TGT, the client sends a request, which contains the TGT, for a service ticket to the ticket-granting server (TGS), and waits for a service ticket to be returned. Having the session ticket available, the client is allowed to communicate with the server that is providing a service he wants to use. The application server can verify the ticket without a need to contact the KDC.

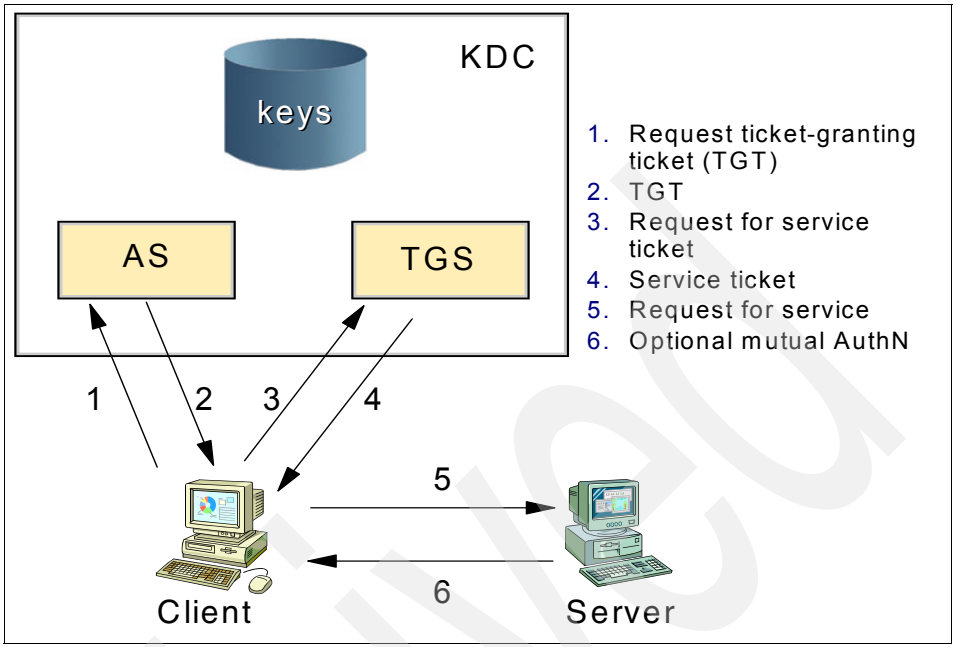


Figure 18-12 Kerberos authentication protocol overview

Inter-realm operation

The Kerberos protocol is designed to operate across organizational boundaries. Each organization wanting to run a Kerberos server can establish its own realm. The name of the realm in which a client is registered is part of the client's name and can be used by the application server to decide whether to honor a request.

By establishing inter-realm keys, the administrators of two realms can allow a client authenticated in one realm to use its credentials in the other realm. The exchange of inter-realm keys registers the ticket-granting service of each realm as a principal in the other realm. A client is then able to obtain a ticket-granting ticket for the remote realm's ticket-granting service from its local ticket-granting service. Tickets issued to a service in the remote realm indicate that the client was authenticated from another realm.

This method can be repeated to authenticate throughout an organization across multiple realms. To build a valid authentication path to a distant realm, the local realm must share an inter-realm key with the target realm or with an intermediate realm that communicates with either the target realm or with another intermediate realm.

Realms are typically organized hierarchically. Each realm shares a key with its parent and a different key with each child. If an inter-realm key is not directly

shared by two realms, the hierarchical organization allows an authentication path to be easily constructed. If a hierarchical organization is not used, it might be necessary to consult some database in order to construct an authentication path between realms.

Although realms are typically hierarchical, intermediate realms might be bypassed to achieve cross-realm authentication through alternate authentication paths. It is important for the end-service to know which realms were transited when deciding how much trust to place in the authentication process. To facilitate this decision, a field in each ticket contains the names of the realms that were involved in authenticating the client.

Why should I use this authentication mechanism? You would only use this mechanism when you have a client that cannot use CSIv2. In addition, because Kerberos implementations vary between platforms, you can only use Kerberos when communicating between z/OS systems. In practice, this means you could use Kerberos when communicating from earlier WebSphere Application Server for z/OS releases. Owing to the complexity of the configuration you would only do this if you already understand and have configured Kerberos.

Enabling Kerberos authentication for WebSphere

Servers can use Kerberos authentication if a local Kerberos realm has been set up and the z/OS system where WebSphere Application Server is running is part of that realm. All clients and servers that take part in connections using Kerberos need a Kerberos principal and a RACF user ID. In the local realm, a user ID in the RACF database needs a KERB segment that defines its principal and associates it with the RACF user ID. A server or client in the local realm will use the Kerberos principal that is defined in the KERB segment of its RACF user profile. For any clients and servers in foreign realms, the following needs to be set up:

- Inter-realm relationships between the KDCs of the local and foreign realm (in both directions) using profiles in class KERBLINK.
- Profiles for the foreign principals need to be defined in class KERBLINK to associate a user ID with the principal (either individually or a generic user ID for all principals in the foreign realm).

Note that an SSL session between client and server or between the partners in a server-to-server connection is a prerequisite for Kerberos authentication.

You enable Kerberos authentication in the EJB container using the WebSphere administrative console. Navigate to **Security** → **Authentication Protocol** → **zSAS Transport**, and then select the **Kerberos** option. You must also select an SSL Repertoire from the drop-down list at the bottom of the zSAS Transport

Configuration page. The client must also support Kerberos authentication and be part of the same Kerberos realm as WebSphere Application Server for z/OS for this to be selected by the negotiation of security protocols. In addition, if client and server both support CSIv2 and zSAS client certificate authentication, CSIv2 will be used.

zSAS asserted identity

In server-to-server connections on z/OS, a server who calls another server on behalf of a client is, under specific circumstances, able to transmit the identity of the client to the server that is called. A highly trusted connection between the two servers must exist for this transfer of identity without further authentication to be allowed. If the conditions are satisfied, the requesting server simply sends the client's user ID in a process called *asserted identity*. The conditions are:

- An SSL connection must exist between the two servers.
- The requesting server (the one that sends the client identity) must be authenticated to its peer with an X.509 client certificate that is associated with the WebSphere server control region's RACF user ID.
- The user ID of the control region for the requesting server must have at least CONTROL authority to resource CB.BIND.target_server in RACF class CBIND, where target_server is the name of the server being called.

Asserted identity is, in a simplified way, similar to the form of delegation supported by Kerberos called *impersonation*. The server who requires services from another server to fulfill the client's request passes on the identity of the client, in a way impersonating the client to the target sever.

Why should I use this authentication mechanism? You would use zSAS identity assertion only if the server wanting to perform identity assertion does not support CSIv2. The client most likely to need this protocol is a WebSphere Application Server for z/OS V4.01 system.

Enabling asserted identity security

You enable zSAS asserted identity in the EJB container using the WebSphere Application Server for z/OS V5 administrative console:

- Check the property Identity Assertion Inbound under **Security** → **Authentication Protocol** → **zSAS Transport**.
- For outbound identity assertion, check the property Identity Assertion Outbound under **Security** → **Authentication Protocol** → **zSAS Transport**.
- If the requesting server is a WebSphere Application Server for z/OS V4.01 system, set the property "Send asserted identities allowed" for the server using the SMEUI of WebSphere Application Server for z/OS V4.01.

- The client must also support asserted identity authentication and SSL client certificate authentication for this to be selected by the negotiation of security protocols.
- You must also select an SSL Repertoire from the drop-down list at the bottom of the zSAS Transport Configuration page.

If client and server both support CSIv2 and zSAS client certificate authentication, CSIv2 will be used.

zSAS non-authenticated or default identities

If the negotiation for a common authentication protocol between client and server or between two EJB servers fails, the only possibility left is to use no security. In this case, the EJB application on the target server runs with a default user ID.

Why should I use this authentication mechanism? You would run unauthenticated only when you are not interested in security, possibly for a test environment. In any other situation you should not run a server without authentication clients unless you are absolutely certain that your network security prevents undesirables from accessing your server.

Enabling non-authenticated requests

To run an application with an unauthenticated client user ID:

- Select the property **Allow non-authenticated clients** under **Security** → **Authentication Protocol** → **zSAS Transport**.
  This property must be set for both partners of the communication.
- If the client is a WebSphere Application Server for z/OS V4.01 system, set “Allow non-authenticated clients” for the server using the SMEUI.
- You also need to set a default remote identity for the EJB container. Navigate to Global Security, and then click **z/OS Security Options** and enter a user ID in the default remote identity field.

18.4.3 zSAS authentication in a single system environment

The most easy and straightforward environment to look at is one where all clients and servers are located in a single z/OS image. In this case, all communication is taking place between address spaces with little chance for any security exposures. Therefore, there is no need to protect the communication channels between clients and servers. SSL or TLS do not need to be used within a single z/OS system. In the diagram in Figure 18-13 on page 639, this type of connection is labeled **1**.

If an identity is passed from a client to a server or between servers, a portable format of the RACF control block Access Control Environment Element (ACEE) called RACO or environment element is used to transfer the user identity from one address space to another.

From a performance point of view, this is an authentication method that is able to pass an identity with a minimum of overhead while causing no loss of security. For the typical WebSphere application where a servlet in a Web container is calling an EJB in an EJB container, having both servers in the same z/OS image provides the best performance without any loss in security.

18.4.4 Authentication in a sysplex

The next environment we are looking at is multiple systems within a sysplex, where a client calls an EJB server in a different z/OS image or where a server sends a request to an EJB server in a different z/OS image. These connections are shown in the diagram in Figure 18-13 on page 639 with labels **2** and **3**. It is assumed that all systems in a sysplex are sharing a single RACF database. Therefore, the sysplex will be in a single Kerberos realm.

For SSL sessions in a single sysplex, WebSphere Application Server for z/OS and OS/390 automatically selects a cipher suite that provides message authentication without encryption (a so-called null encryption cipher suite).

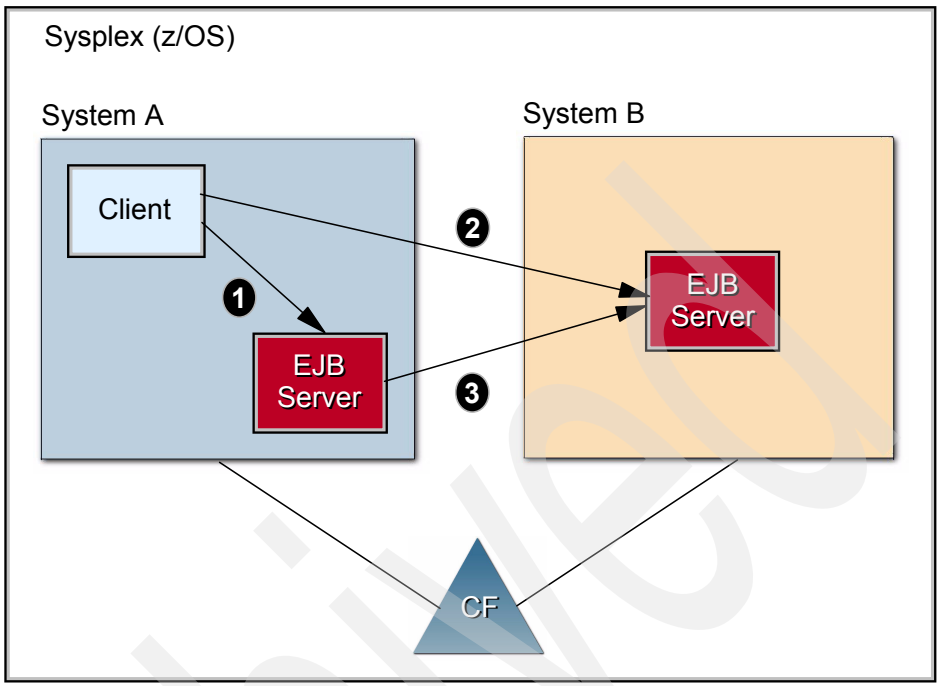


Figure 18-13 EJB client and server connections in a sysplex

Client-to-server authentication

For a connection between a WebSphere client and an EJB server on a different z/OS image, the following authentication methods are supported and will be selected in the order shown:

1. SSL client certificates

   The client certificate and private key need to be available to the address space running the client program. If the client is a Web container, it will only be able to provide the certificate of its control region, not the client certificate of the end user on the Web.

2. Kerberos over SSL

   The client's Kerberos principal is defined in the KERB segment of the RACF user profile for the RACF user ID of the client. A stand-alone client application can pass the credentials of any authenticated Kerberos principal. If the client is a Web container, the client principal is determined by the user ID of the Web container's control region.

3. Basic authentication over SSL

   User ID and password are sent over an SSL connection. For a discussion of the user IDs that can be used in this situation, see item 5.

4. User ID and PassTicket

   This is a good solution if it can be determined with certainty that there will not be more than one request for the same RACF user ID within the same second.

5. User ID and password

   Which user ID could possibly be used here depends on the type of client program. While a stand-alone WebSphere client could prompt its end user for user ID and password quite easily, this would be more difficult for a servlet in a Web container. If basic authentication or form-based authentication is used, the RACF password is not available to the Web application.

6. No security

   If the property “Allow non-authenticated clients” is selected in the WebSphere administrative console under **Security → Authentication Protocol → zSAS Protocol**, no authentication is done if the negotiation did not result in an authentication method supported by both partners to be selected. In this case, the EJB application runs under the user ID defined in property “Remote identity” (the default is WASDFTU).

Server-to-server authentication

Next we look at the connection labeled **2** in Figure 18-13 on page 639. For a connection between an EJB server and another EJB server on a different z/OS image, the following authentication methods are supported and will be selected in the order shown. Note that SSL connections are always used between control regions. RACF key rings and the certificates connected to the key rings must be owned by the control region user IDs.

1. Kerberos over SSL

   The Kerberos principal for the sending server’s control region is defined in the KERB segment of the RACF user profile for the control region’s user ID. If the sending server’s client is an authenticated Kerberos principal, the sending server can use delegation and send the client’s principal.

2. Asserted identity

   The sending server can transmit an already verified user ID (no password is sent) to the target server. This is only allowed if the sending server has been authenticated with an SSL client certificate and the necessary level of trust has been set up in RACF profiles in class CBIND.

3. User ID and PassTicket

   A good solution if it can be determined with certainty that there will not be more than one request for the same RACF user ID within the same second.

4. SSL client certificates

   The certificate of the control region for the sending server is used as the client certificate. The client user ID of the EJB request does not come into play here as the authentication is strictly between the control regions.

5. User ID and password

   Whichever user ID the target server should use for the EJB needs to be set in variables REM_USERID and REM_PASSWORD in the sending server's environment. What user ID to specify and where to get the password from has to be decided when designing the individual application. It depends on the design of the client application whether it is able to get an appropriate RACF user ID and password and set the appropriate environment variables to have the variables transferred to the EJB server on z/OS.

   Note: The client delivered with WebSphere Application Server Advanced Edition on UNIX and Windows platforms only sends a user ID and password over an RMI/IIOP connection if the connection is protected using SSL. The same is true for the J2EEClient_NT that is delivered with WebSphere Application Server for z/OS and OS/390.

6. No security

   If the property "Allow non-authenticated clients" is selected in the WebSphere administrative console under **Security** → **Authentication Protocol** → **zSAS Protocol**, no authentication is done if the negotiation did not result in an authentication method supported by both partners to be selected. In this case, the EJB application runs under the user ID defined in property "Remote identity" (the default is WASDFTU).

18.4.5 Authentication between z/OS systems outside a sysplex

Multiple z/OS systems that are not members of the same sysplex are a potentially much less secure environment than systems in the same sysplex. TCP/IP connections that can use CTC connections or Hipersockets within a sysplex will have to be made over not that secure communication lines. Furthermore, while it is assumed that systems within a sysplex are using the same RACF database, this cannot be assumed in the case of systems outside a sysplex. (Although you could be using some form of RACF database replication technology to keep the two systems in sync, WebSphere assumes the remote

z/OS is less trustworthy and the communications between the systems insecure.) Figure 18-14 shows a diagram of such an environment.

Within a sysplex, WebSphere Application Server for z/OS V5 is using a null encryption cipher suite for SSL. This means that data integrity is protected with a Message Authentication Code (MAC), but no encryption is done. This is different between systems that are not members of the same sysplex. In this environment, both message authentication and encryption are used in SSL sessions.

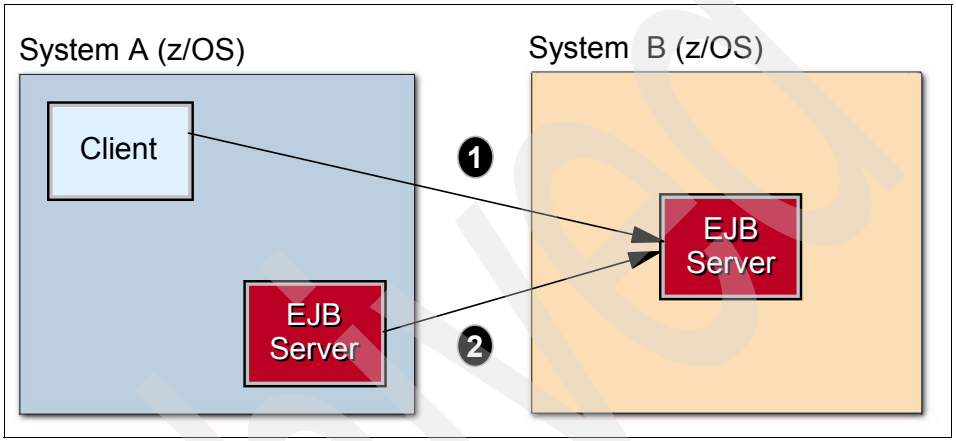


Figure 18-14 EJB client/server connections between z/OS systems outside sysplex

For a connection between a WebSphere client and an EJB server on a different z/OS image not in the same sysplex (labeled 1 in Figure 18-14), the following authentication methods are supported and will be selected in the order shown:

1. SSL client certificates

   The client certificate and private key need to be available to the address space running the client program. If the client is a Web container, it can only provide the certificate of its control region, not the client certificate of the end user on the Web.

2. Kerberos over SSL

   The client's Kerberos principal is defined in the KERB segment of the RACF user profile for the RACF user ID of the client. In this configuration, client and server may or may not be in the same realm. If the client is in a different realm, appropriate profiles in RACF class KERBLINK must have been defined to establish trust between the realms and to associate a RACF user ID with the client principal on the server's system.

3. Basic authentication over SSL

   User ID and password are sent over an SSL connection. For a discussion of the user IDs that can be used in this situation, see item 4.

4. User ID and password

   Which user ID could possibly be used here depends on the type of client program. While a stand-alone WebSphere client could prompt its end user for user ID and password quite easily, this would be more difficult for a servlet in a Web container. If basic authentication or form-based authentication is used, the RACF password is not available to the Web application.

5. No security

   If the property “Allow non-authenticated clients” is selected in the WebSphere administrative console under **Security** → **Authentication Protocol** → **zSAS Protocol**, no authentication is done if the negotiation did not result in an authentication method supported by both partners to be selected. In this case, the EJB application runs under the user ID defined in property “Remote identity” (the default is WASDFTU).

For a connection between two EJB servers on different z/OS systems, which are not members of the same sysplex (labeled 2 in Figure 18-14 on page 642), the following authentication methods are supported and will be selected in the order shown. Note that SSL connections are always used between control regions. RACF key rings and the certificates connected to the key rings must be owned by the control region user ID.

1. Kerberos over SSL

   The Kerberos principal for the sending server’s control region is defined in the KERB segment of the RACF user profile for the control region’s user ID. In this configuration, client and server may or may not be in the same realm. If the sending server’s client is an authenticated Kerberos principal, the sending server can use delegation and send the client’s principal. Should the client (the sending server’s control region or the end user’s Kerberos principal) be in a different realm, appropriate profiles in RACF class KERBLINK must have been defined to establish trust between the realms and to associate a RACF user ID with the client principal on the server’s system.

2. Asserted identity

   The sending server can transmit an already verified user ID (no password is sent) to the target server. This is only allowed if the sending server has been authenticated with an SSL client certificate and the necessary level of trust has been set up in RACF profiles in class CBIND. For this option to be meaningful, it is necessary that at least the USER and GROUP profiles in the RACF databases of both systems are synchronized, for instance using RRSF.

3. SSL client certificates

   The certificate of the control region for the sending server is used as the client certificate. The client user ID of the EJB request does not come into play here as the authentication is strictly between the control regions.

4. User ID and password

   Whatever user ID the target server should use for the EJB needs to be set in variables REM_USERID and REM_PASSWORD in the sending server's environment. What user ID to specify and where to get the password from will have to be decided when designing the individual application.

5. No security

   If the property "Allow non-authenticated clients" is selected in the WebSphere Application Server for z/OS and OS/390 Administration Application, no authentication is done if the negotiation did not result in an authentication method supported by both partners to be selected. In this case, the EJB application will run under the user ID defined in property "Remote identity" (the default is WASDFTU).

18.4.6 Authentication with EJB applications on non-z/OS platforms

Connections between clients and EJB applications, or between J2EE components where one system is a z/OS system and the other system runs WebSphere Application Server on a UNIX or Windows system, are problematic from a security standpoint. Not only are the security systems on individual platforms vastly different, there are also not too many authentication protocols that have been implemented across multiple platforms.

The diagram in Figure 18-15 on page 645 shows the different possibilities for connections across multiple platforms:

1. Connections between a workstation client and an EJB container on z/OS (labeled 1)
2. Connections between a Web container on a workstation client and an EJB container on z/OS (labeled 2)
3. Connections between a z/OS client and an EJB container on a workstation (labeled 3)
4. Connections between a z/OS Web container and an EJB container on a workstation (labeled 4)
5. Connections between an EJB server on a workstation and an EJB server on a z/OS platform (labeled 5)
6. Connections between an EJB server on a z/OS and an EJB server on a non-z/OS platform (labeled 6)

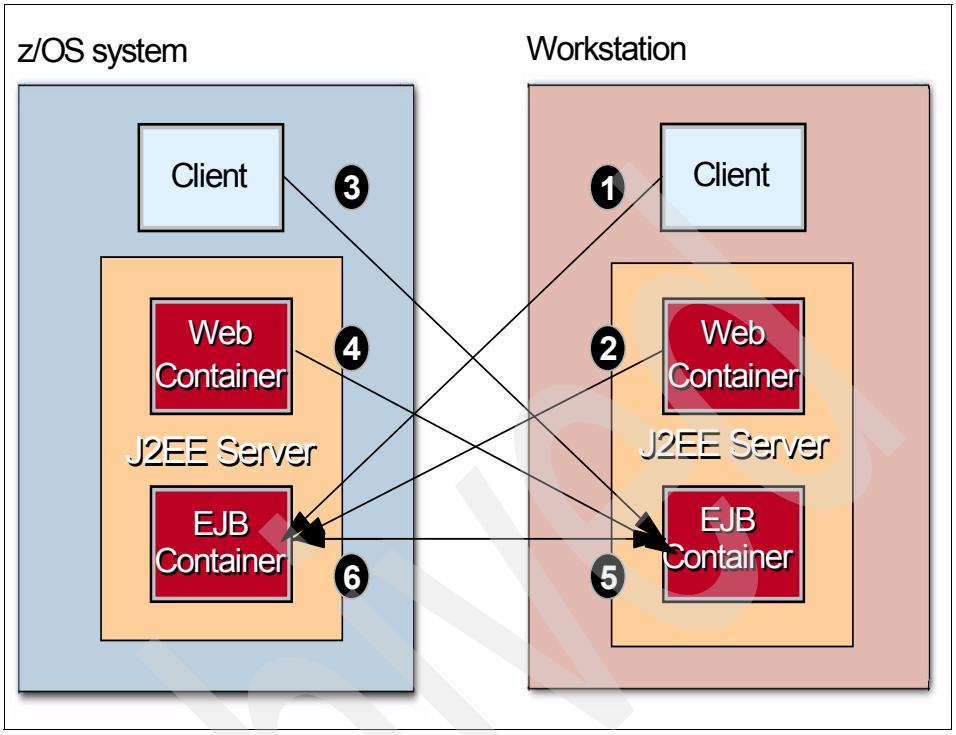


Figure 18-15 EJB client and server connections between z/OS and distributed systems

For a connection between a WebSphere workstation client and an EJB container on z/OS (case 1), the authentication methods that can be used very much depend on the capabilities and configuration of the workstation client. A number of possibilities that can most likely be made to work are listed here:

1. Mutual authentication using SSL client certificates

   The client certificate and private key need to be available to the client program. An association between the client certificate and a RACF user ID needs to be set up.

2. Basic authentication (user ID and password over SSL)

   The client's user ID and password can be obtained by prompting the end user or from a configuration file. The user ID and password must be valid for the SAF registry of the z/OS server. The client establishes an SSL session with the EJB container. A detailed example of this configuration was provided in the Redbook *z/OS WebSphere and J2EE Security Handbook*, SG24-6846.

3. No security

   If the property “Allow non-authenticated clients” is selected in the WebSphere Application Server for z/OS and OS/390 Administration Application, no authentication is done if the negotiation did not result in an authentication method supported by both partners to be selected. In this case, the EJB application will run under the user ID defined through the WebSphere administrative console under **Security → Global Security → z/OS Security Options**. Set the property “Remote identity.”

For a connection between a Web container on a distributed platform and an EJB container on z/OS (case 2), the authentication method used will depend on the configuration of the two servers.

1. User ID and password over SSL

   As with case 1, a user ID and password, valid in the z/OS SAF registry, can be passed from the Web container on the distributed platform to the EJB container on the z/OS platform over an SSL session. In this case, the user ID and password are associated with the J2EE server rather than with the end user. The user ID and password values are stored in a WebSphere Application Server AE key file.

2. No security

   Similar to case 1, requests from a distributed Web container can be made to a z/OS EJB server with no authentication if the “Allow non-authenticated clients” property is specified for the z/OS J2EE server.

For a connection between a client on z/OS and an EJB container on a non-z/OS platform (case 3), the following authentication methods are supported and will be selected based on the configuration of the distributed J2EE server.

1. SSL client certificates

   The default certificate configured to the SAF profile of the user ID under which the client is running is used as the client certificate.

2. No security

   Requests from a z/OS client can be made to a distributed EJB server with no authentication if the distributed J2EE server does not require authentication (for example, global security is turned off).

For a connection between a Web container on z/OS and an EJB container on a non-z/OS platform (case 4), the possibilities are similar to case 3.

1. SSL client certificates

   The certificate of the control region for the sending server is used as the client certificate. The client user ID of the EJB request does not come into play here as the authentication is strictly between the control regions.

2. No security

   Requests from a z/OS Web container can be made to a distributed EJB server with no authentication if the distributed J2EE server does not require authentication (for example, global security is turned off).

Connections between an EJB container on a distributed server and an EJB container on z/OS (case 5) and between an EJB container on z/OS and an EJB container on a distributed server will be similar to cases 2 and 4, respectively.

18.5 EJB container authorization

Authorization means checking whether a user or principal is allowed to access a protected resource. For authorization to be meaningful, the user or principal needs to have been authenticated or identified by a trusted upstream server. There is no point in checking the authority of a user whose identity has not been trustfully verified.

18.5.1 Overview

The EJB container supports two APIs for programmatic authorization. These come from the javax.EJB.EJBContext interface. They are similar to those supported by the Web container. getCallerPrincipal() is similar to the getUserPrincipal() of the HttpServletRequest interface.

isCallerInRole(), which takes a role name in the form of a java.lang.String as an argument, is similar to the isUserInRole() method of the HttpServletRequest interface.

The declaration of roles and authorization of methods are similar in concept to those found in the Web container, although the elements and tags have different syntax. Example 18-2 shows an example of the security elements in an EJB deployment descriptor.

Example 18-2 Security elements in an EJB deployment descriptor

```
<security-role-ref>
  <description>Any employee of the company</description>
  <role-name>Employees</role-name>
  <role-link>AllWorkers</role-link>
</security-role-ref>
<security-role>
  <description>All the companies worker</description>
<role-name>AllWorkers</role-name>
</security-role>
<method-permission>
```

```
  <role-name>Worker</role-name>
<method>
    <ejb-name>Tools</ejb-name>
    <method-intf>Home</method-intf>
    <method-name>*</method-name>
  </method>
  <method>
    <ejb-name>EJBSample</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>runAsServer</method-name>
    <method-params>
      <method-param>java.util.Vector,itso.utility.Tools</method-param>
    </method-params>
  </method>
  <method>
    <ejb-name>Tools</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>*</method-name>
  </method>
</method-permission>
```

18.5.2 Resource authorization at the application level

This section describes resource authorization at the application level.

Declarative security

The J2EE specification defines declarative security as the means of expressing an application's security structure, including security roles, access control, and authentication requirements in a form external to the application. The deployment descriptor is the primary vehicle for declarative security in the J2EE platform.

In the deployment descriptor of EJB applications, method permissions can be defined that need to be satisfied before the EJB's methods can be executed. The method permissions can be defined on the level of individual methods.

If a EJB1 has a RunAs role set, and tries to invoke a method on EJB2 which has a Method Permission defined, the user ID associated with EJB1's role must be in the role that is authorized to the methods of EJB2.

In WebSphere Application Server for z/OS V5, the mapping of a role to a user ID can be performed when the application is deployed, or if you have set the following properties in **Security** → **User Registries** → **LocalOS** → **Custom Properties**:

```
com.ibm.security.SAF.authorization =true
com.ibm.security.SAF.delegation=true
```

Your SAF security manager will be called to map the role to a user ID and to perform the authorization check using that user ID prior to invoking the methods of an EJB. For further details concerning these two properties, see 10.10.1, “Security Customization panel settings” on page 309, referring to the Version 5.1 field named “Use SAF EJBROLE profiles to enforce J2EE roles.”

Programmatic security

It is desirable to use declarative security wherever possible to relieve the application programmer of the task of having to add security-relevant code to an application.

In cases where this is not feasible or where the program logic requires security to be handled by the program, programmatic security can be used, as defined by the J2EE specification: Programmatic security refers to security decisions made by security-aware applications. Programmatic security is useful when declarative security alone is not sufficient to express the security model of the application.

An API is available with two methods for the EJB container.

- getCallerPrincipal
- isCallerInRole

Methods getUserPrincipal and getCallerPrincipal, respectively, are used to retrieve the user ID associated with the current EJB. Methods isUserInRole and isCallerInRole, respectively, can be used to check whether the current user has been given access to a certain role. Based on the result, the program can make decisions in its processing path.

18.6 Security propagation

The propagation of security identity from one object to another is supported both in the Web container and the EJB container. The security identity actually propagated depends on the run-as setting as described in 3.2.4, “RunAs versus run-as: Identity propagation” on page 52. In summary, if an EJB (let us call it EJB1) has a run-as defined in its EJB deployment descriptor (the ejb.xml file), the identity associated with that run-as will become the “caller” for any objects that EJB1 calls.

In addition, a unique feature of WebSphere Application Server for z/OS V5 is the ability to propagate the identity of the current thread to the J2CA resource adapter. If EJB1 uses a J2CA connector to an EIS system and a local-mode, container-managed connection, the run-as identity of EJB1 will be propagated to the connector. (This assumes that no container-managed JAAS authentication alias has been defined for the connection factory, because the JAAS alias will be used rather than the current thread identity if it is defined.)

If an EJB1 has a run-as role set, and tries to invoke a method on EJB2 which a Method Permission defined, the user ID associated with EJB1's role must be in the role that is authorized to the methods of EJB2.

Note: In WebSphere Application Server for z/OS V5 a role can also be assigned to a servlet. See "Web container identity delegation and propagation" on page 556 for a discussion about run-as in relation to servlets

A WebSphere application server can use three different places to determine the association with a role:

- The local SAF registry when the local OS registry is selected and the local OS custom variable com.ibm.security.SAF.delegation is set to true. In this case, EJBROLE or GEJBROLE profiles need to be modified to contain a RACF user ID within the APPLDATA segment of the profile.
- Binding, extensions, and deployment descriptors located in the application's .ear file and created either under WebSphere Studio Application Developer or the AAT. This applies when the user registry is not the OS registry or when the registry is the OS registry but the local OS custom variable com.ibm.security.SAF.delegation is set to false. This is controlled by the application configuration check box "Use Metadata From Binaries", which needs to be selected.
- Binding, extensions, and deployment descriptors created during the application installation in the "Map security roles to users/groups" task. This applies when the user registry is not the OS registry or when the registry is the OS registry but the local OS custom variable com.ibm.security.SAF.delegation is set to false. This is controlled by the application configuration check box "Use Metadata From Binaries", which needs not to be selected.

An example of using the AAT to set run-as role for an EJB in the deployment descriptor is shown in Figure 18-16 on page 651.

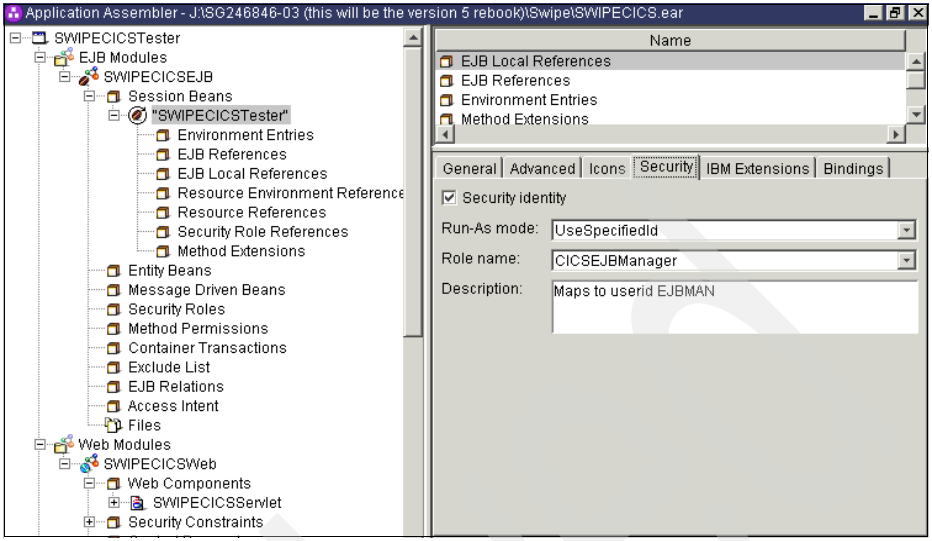


Figure 18-16 Setting the run-as role for an EJB using the AAT

18.7 EJB container security verification using SWIPE

The SWIPE application can be used to explore interoperability between WebSphere servers and the use of CSIv2.

By installing SWIPE into different WebSphere servers, you can run it in one server to invoke the SWIPE EJB in another server.

18.7.1 Java 2 security

When you enable global security in WebSphere, you can chose to enable Java 2 security as well. If you do not, you can run SWIPE to invoke the SWIPE EJB in a remote server without having to modify the Java 2 security policy files.

If you do have Java 2 security enabled, you will need to modify the app.policy file of the server to enable SWIPE to invoke the SWIPE EJB in a remote server.

Example 18-3 on page 652 shows the Java security permissions you need to add to be able to do this.

Example 18-3 Java 2 security, app.policy: Settings for allowing remote interoperability

```
grant codeBase
"file:/WebSphere/BSO6/appserver/installedApps/cd6sc59/SWIPE.ear/-" {
permission java.util.PropertyPermission "java.naming.factory.initial", "write";
permission java.util.PropertyPermission "java.naming.provider.url", "write";};
```

In 7.5.2, “SWIPE and Java 2 security” on page 159, we explain how to find the app.policy file.

18.7.2 IBMEBizEnv

7.5.3, “Setting the IBMEBizEnv environment variable” on page 168 explains how to set up a JVM Property called IBMEBizEnv to a value that is displayed by the SWIPE example, and helps to prove that you have access to the SWIPE EJB in a remote server.

18.7.3 SWIPE input fields

When SWIPE is run in a browser, there are two input fields on the form displayed that must be filled in to have SWIPE invoke a method on the SWIPE EJB deployed in a different WebSphere server. These two fields are explained in 7.6.1, “SWIPE: EJBCaller - Input Part A” on page 174. 7.6.6, “Remote JNDI example” on page 182 explains how to find the fully qualified JNDI name for an EJB deployed into a WebSphere server.

18.7.4 Test Case 1: Both servers on the same z/OS image

To verify that we could have a servlet in one WebSphere server invoke a method on an EJB in another WebSphere server, we set up two WebSphere servers on an LPAR running z/OS 1.4. One was named WASD4 and the other WASD6. Both servers had global security enabled using the Local OS approach. The WASD4 server did not have Java 2 security enabled. SWIPE was installed into both servers.

We started a browser and ran the EJBCaller SWIPE servlet in the WASD4 server using this request:

```
http://wtsc59.itso.ibm.com:9084/IBMEBizWeb/secure/EJBCaller
```

We started another browser window and ran the EJBCaller SWIPE servlet in the WASD6 server using this request:

```
http://wtsc59.itso.ibm.com:9084/IBMEBizWeb/secure/EJBCaller
```

In both cases we logged on as user USRCEO.

In the output produced from running SWIPE in the WASD4 server, we then filled in the input fields as shown in Table 18-2.

Table 18-2 Values entered to invoke SWIPE in the WASD6 server

Input field	Value
Remote URL	localhost:2806
Remote JNDI Name (EJBSample)	cell/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59/ejb/EJBSample

The above values tell the SWIPE program to locate the SWIPE EJBSample EJB by doing a lookup to the WASD6 server, the daemon for which is listening on port 2806.

We then pressed **Submit** and SWIPE ran the EJBCalled servlet in the WASD4 server.

Figure 18-17 on page 654 shows some of the SWIPE output displayed in the browser. The output shows information about the RunAsCaller method in the EJBSample EJB being run in the WASD4 server. Each of the methods invoked in the local EJBSample EJB invokes the following methods on the EJBSample EJB in the WASD6 server:

- WhereAmI
- getEjbSecurityInfo

In the output between the lines with the text `Remote SWIPE EJB Call - Start` and `Remote SWIPE EJB Call - End` is shown:

- The user ID of the client that called the method of the remote EJB
- The user ID that the method in the remote EJB is running under
- The value of the IBMEBizEnv JVM Property from the remote server

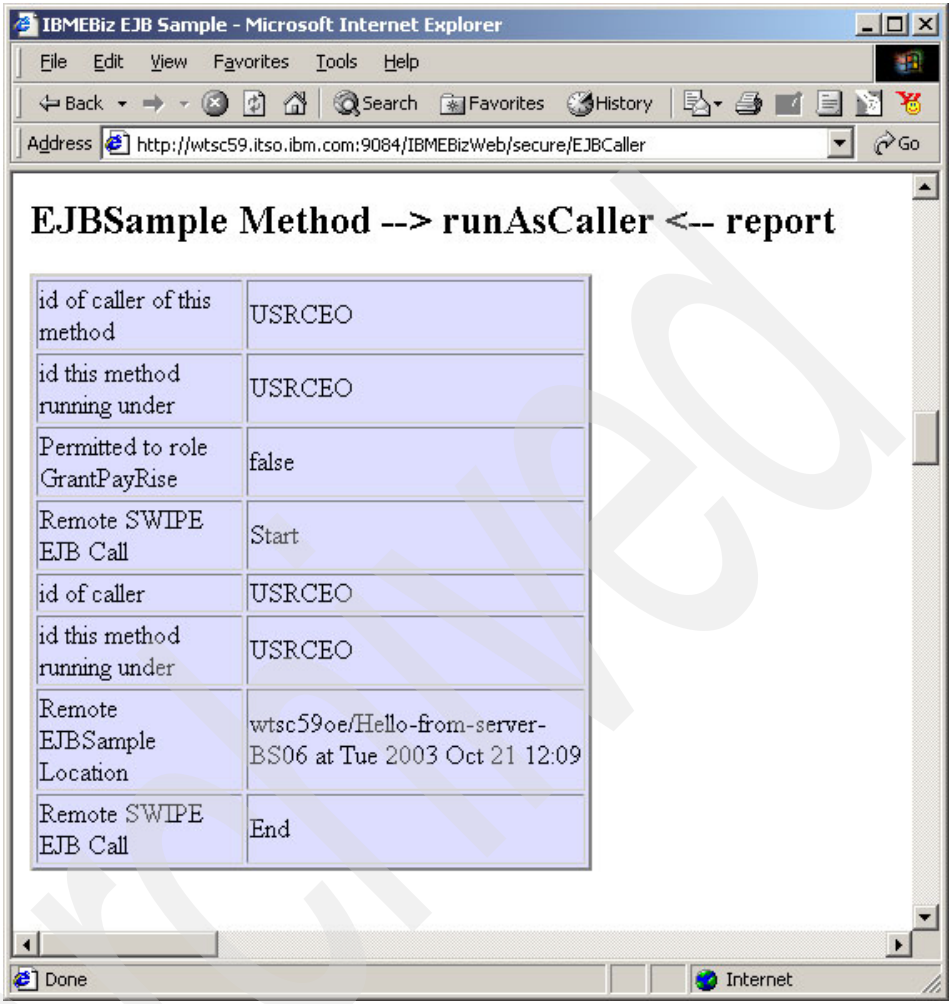


Figure 18-17 Result of the WASD4 server calling SWIPE in the WASD6 server

The above output shows that the methods of the EJBSample SWIPE EJB executed in the WASD6 server knew the user ID of the called was indeed the user ID that the method was running under in the WASD4 server.

Also displayed is the value of the IBMEBizEnv JVM Property setting of the WASD6 server, to show that we have indeed accessed the EJBSAMPLE EJB in the WASD6 server.

This shows that the user ID was successfully propagating between servers.

We did the same test, invoking the EJBSample SWIPE EJB in the WASD4 server from SWIPE in the WASD6 server. The result was the same as above.

18.7.5 Test Case 2: Windows to z/OS

To test interoperability between Windows and z/OS, we installed SWIPE into WebSphere V5 on a Windows 2000 system.

We then ran the EJBCaller SWIPE servlet in a browser on the Windows machine using this URL:

```
http://localhost:9085/IBMEBizWeb/secure/EJBCaller
```

In the output produced from running SWIPE in the Windows server, we then filled in the input fields as shown in Table 18-3.

Table 18-3 Values entered to invoke SWIPE in the WASD6 server

Input field	Value
RemoteURL	wtsc59.itso.ibm.com:2806
Remote JNDI Name (EJBSample)	cell/nodes/nd6cd6sc59/servers/wd6nd6cd6sc59/ejb/EJBSample

Results were similar to those shown in Figure 18-17 on page 654.

18.8 CSIv2 security verification using SWIPE

This section demonstrates the use of CSIv2, described in 18.2.2, “CSIv2 authentication in WebSphere Application Server” on page 610, when applications in different WebSphere environments interact.

18.8.1 SWIPE to SWIPE

To demonstrate the principles of CSIv2 we used the SWIPE application. By installing the SWIPE application into different WebSphere servers, you can explore and test the interaction of an application across the same or different platforms.

7.6.1, “SWIPE: EJBCaller - Input Part A” on page 174 describes the input parameters that trigger the SWIPE application to in effect invoke itself in a remote location.

When you try this, many things happen when **Submit** is pressed, Figure 18-18 depicts the actions the SWIPE application performs when it interacts with another copy of SWIPE deployed in a second server.

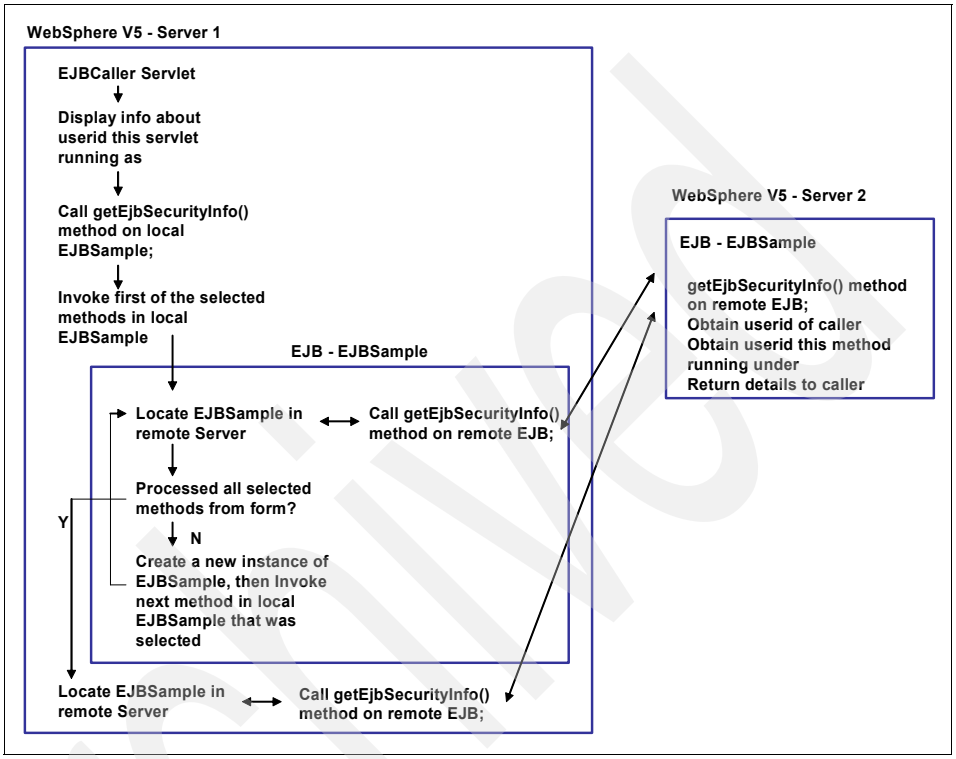


Figure 18-18 SWIPE to SWIPE interaction

The flow depicted above is as follows:

1. **Submit** is clicked and the EJBCaller servlet starts in WebSphere.
2. The servlet displays information about what user ID the servlet is running under.
3. The servlet creates a local EJBSample EJB, and calls the getEjbSecurityInfo() method on it.
4. Then, based on the value selected on the SWIPE input form, the servlet calls the method selected on the EJBSample EJB.
5. The method in the local EJB locates the remote EJBSample EJB.
6. The local EJB invokes the getEjbSecurityInfo() method on the remote EJBSample EJB.

7. The EJB creates a new instance of the EJBSample EJB and invokes the method corresponding to the next one specified in the SWIPE input form, causing the above two steps to be repeated; this loop continues until all the selected methods have been processed.
8. Control returns to the servlet, which then locates the remote EJBSample EJB.
9. The servlet calls the getEjbSecurityInfo() method on the remote EJBSample EJB.
10. The servlet displays in the browser output to show the results of this processing.

18.8.2 WebSphere on Windows 2000 to WebSphere on z/OS: No SSL, asserted identity

In this example, we wanted to show that we could have an application that was deployed into WebSphere Application Server V5 on a distributed system and invoke an EJB in an application deployed into WebSphere Application Server V5 on z/OS. In this example, we did not use SSL between the WebSphere servers.

Asserted identity

In addition, we wanted to show an asserted identity being passed between the two. By that we mean that the user ID the application is running under in WebSphere on the distributed platform will flow to the application running in WebSphere on z/OS.

This requires that the user ID the application runs under on the distributed system is defined in RACF on the z/OS system.

Setup

On the z/OS system, we used a base server called WASD4, which had security enabled, using Local OS, as the Active User Registry. The SWIPE application was deployed into this server.

It is also necessary that the user ID of the started task on the z/OS system has CONTROL access to the CB.BIND.<clusterName> rule in the CBIND RACF class.

On Windows 2000 running WebSphere V5 with Fix Pack 2, in a base server configuration, the SWIPE application was installed. Additionally, APAR PQ77960 is necessary for the WebSphere V5 system on the distributed side. At the time of writing we had a test fix for this APAR we used for testing.

The WebSphere V5 system on the Windows 2000 system had security enabled with a local LDAP used as the Active User Registry. The following users were defined to the local LDAP:

- WINWORKR
- WINMGR
- WINCEO

Because the aim was to show the use of asserted identities, these user IDs were also defined in RACF on the z/OS system.

WebSphere Application Server on Windows 2000 configuration

In WebSphere Application Server V5 on Windows 2000, the CSIv2 outbound authentication and outbound transport were set as shown in Table 18-4.

Table 18-4 CSIv2 settings in WebSphere V5 on Windows for asserted identity

Panel	Property	Setting
CSIv2 Outbound Authentication	Basic Authentication	Never.
	Client Certificate Authentication	Never.
	Identity Assertion	Selected.
	Stateful	Selected.
CSIv2 Outbound Transport	Transport	TCP/IP.
	SSL Settings	Can be anything, but not used since using TCP/IP.

WebSphere Application Server on z/OS configuration

In WebSphere Application Server V5 on z/OS, the CSIv2 inbound authentication and inbound transport were set as shown in Table 18-5.

Table 18-5 CSIv2 settings in WebSphere V5 on z/OS for asserted identity

Panel	Property	Setting
CSIv2 Inbound Authentication	Basic Authentication	Never
	Client Certificate Authentication	Never

Panel	Property	Setting
	Identity Assertion	Selected.
	Trusted Servers	'*'
	Stateful	Selected.
CSIv2 Inbound Authentication - Additional Properties	Client Authentication Type	SAFUSERIDPASSWORD.
	SAF Identity Assertion	Selected (1).
	DN Identity Assertion	Not selected.
	Certificate Identity Assertion	Not selected.
CSIv2 Inbound Transport	Transport	TCP/IP.
	SSL Settings	Can be anything, but not used since using TCP/IP.

Verifying asserted identity

We then entered the following URL to start the SWIPE application in a Web browser on the Windows 2000 system:

```
http://localhost:9085/IBMEBizWeb/secure/EJBCaller
```

We logged on when prompted as user *winceo*.

This displayed the initial SWIPE output in the browser. We then filled in the two fields in the SWIPE input form that cause the SWIPE application to invoke SWIPE EJB methods in a remote location, as shown in Figure 18-19 on page 660.

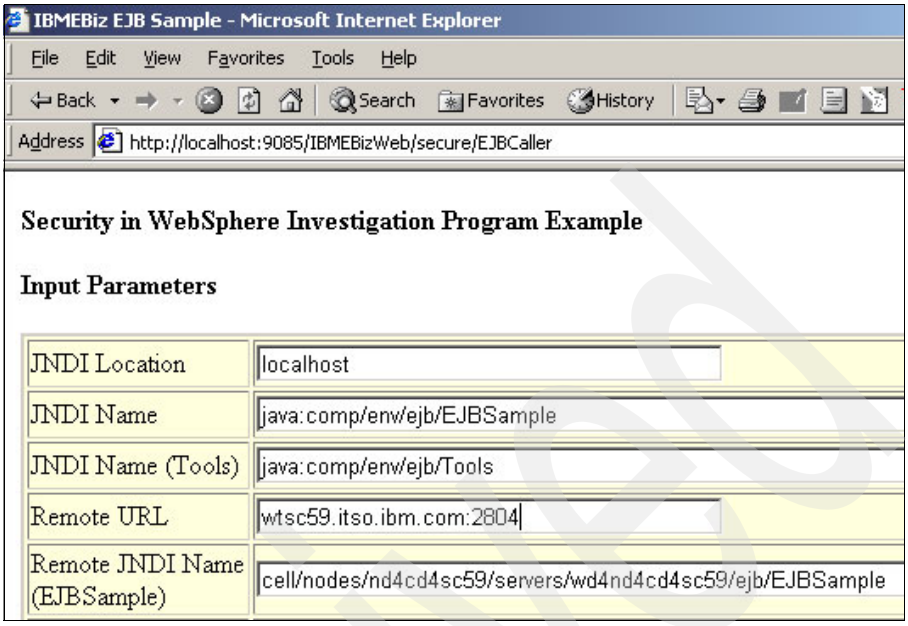


Figure 18-19 SWIPE: Remote SWIPE EJB fields filled in

In 7.6.6, “Remote JNDI example” on page 182, we explain how to determine the full JNDI name of an EJB in a WebSphere V5 server on a z/OS system.

Figure 18-20 on page 661 shows some of the output produced when **Submit** was pressed.

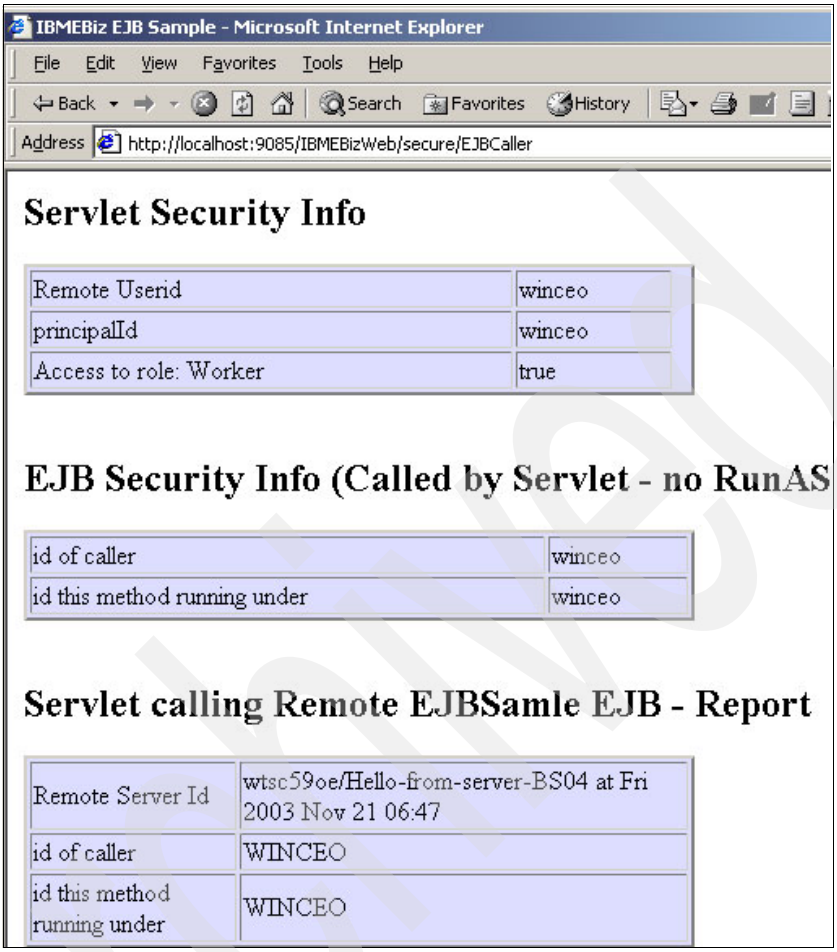


Figure 18-20 SWIPE on Windows 2000 to SWIPE on z/OS: Output

The output shows that the servlet on the Windows 2000 system was running under the user ID winceo.

Servlet to remote EJB

It also shows that the servlet in WebSphere V5 on the Windows 2000 system has invoked the SWIPE EJB deployed in the WebSphere V5 server on the z/OS system. It shows that the method called ran under the user ID WINCEO. This shows that the asserted identity approach worked.

Note also that the value of the IBMEBizEnv JVM property has also been returned to verify that communication to the remote server was achieved.

18.8.3 WebSphere Application Server to WebSphere Application Server on the same LPAR

On a z/OS LPAR, we had two separate base servers called WASD4 and WSASD6. No changes were made to the CSIv2 settings after the base servers were created.

The SWIPE application was deployed into each.

We tested whether the SWIPE application in one of the servers could invoke the SWIPE application in the other server. We found that this occurred without error.

19

WebSphere Application Server logging and auditing

In this chapter, we discuss the logs that are created in WebSphere Application Server for z/OS and describe how these logs can be used to audit the environment.

19.1 Sources of WebSphere Application Server for z/OS log data

The first step in establishing your logging and audit policy is to identify where events can be logged. Both WebSphere and RACF can log events to the Systems Management Facility (SMF) log. In 10.2, “WebSphere Application Server SAF integration” on page 294 we show the WebSphere architecture and its relationship to the System Authorization Facility (SAF).

There are two sources of WebSphere log data: SMF type 120 records, which are written by WebSphere, and records that are written by your external security management product. RACF uses SMF type 80 records for its logging.

19.1.1 WebSphere SMF data

WebSphere uses the type 120 record to record events related to:

- Capacity planning, such as the transaction volumes and completion times
- Application structure, such as what components of an application are being called when
- Error reporting

By default, all of the type 120 recording is disabled. You can select which events should be recorded for your application by setting the appropriate environment variables for your application. This is most easily done from the WebSphere administrative console. For our SWIPE application this was done from the Environment Entries panel in the Additional Properties dialog for our WebSphere server. Additional information is available in *WebSphere Application Server for z/OS Information Center*.

19.1.2 RACF SMF data

From an authentication and authorization point of view, the RACF log records are far more interesting. RACF has several options that control event logging. In the following sections we describe the logging controls that are available and how these log records can be processed.

Overview

The first step in setting up your logging options is to determine what events you want to record. This requires understanding why RACF writes an audit record. The events include:

- Events that are always logged, such as failed LOGON attempts

- Users for whom logging has been requested by specifying the UAUDIT attribute on the RACF ADDUSER or ALTUSER command
- RACF profiles that have the AUDIT options specified by the owner of the profile
- RACF profiles that have the GAUDIT option specified by an auditor
- General resource classes that have SETROPTS LOGOPTIONS set with a value other than NEVER

RACF log records are written to the SMF data sets, which are SYS1.SC59.MAND and SYS1.SC59.MANE on our systems.

Logging the use of EJBROLES

When a user requests the use of a role, WebSphere performs an authorization request to see if the user is permitted to access the requested role. This check is performed against the resource name <role-name> in the EJBROLES general resource class. Reviewing these records allows you to see who is using or attempting to use particular roles.

In our test environment, we asked RACF to log accesses to EJBROLE by specifying AUDIT(ALL) on selected EJBROLE profiles using the RALTER command:

```
RALTER EJBROLE <role_name> AUDIT(ALL)
```

Our initial approach was to AUDIT(ALL) all of the EJBROLE profiles, which we did by using the RACF SEARCH command to find all of the EBJROLE profiles and to create the RALTER commands with AUDIT(ALL) specified. We did this with the command:

```
SEARCH CLASS(EJBROLE) CLIST('RALTER EJBROLE ' ' AUDIT(ALL)')
```

We then executed the CLIST created by the SEARCH command:

```
EXEC 'MARNEL.EXEC.RACF.CLIST'
```

Note: The SEARCH command was issued by the user MARNEL and created the data set MARNEL.EXEC.RACF.CLIST with one RALTER command (with AUDIT(ALL)) for each of the EJBROLE profiles.

As we began to see a volume of not-so-interesting log records for roles such as “monitor”, we quickly changed our log specifications to AUDIT(FAILURES) for these non-interesting roles.

Logging CBIND accesses

The CBIND class is used to control the interprocess connections between the WebSphere servers. We recommend logging only failures for these accesses. Because this is the default audit action, enabling AUDIT(FAILURES) for your CBIND profiles is done automatically.

WebSphere's use of the X500_NAME field

WebSphere Version 5 uses the X500NAME audit field to contain information about the authentication mechanisms used by the user. The RACF type 80 access audit records contain this X500NAME information in the X500_ISSUER and X500_SUBJECT fields. The values that these fields can contain are shown in Table 19-1.

Table 19-1 X500NAME values

Authentication mechanism	Service (X500_ISSUER)	Authenticated (X500_SUBJECT)
Custom Registry	WebSphere custom registry	Custom registry principal name
Kerberos	WebSphere Kerberos	Kerberos principal name
RunAs Rolename	WebSphere role name	Role name
RunAs Server	WebSphere server credential	MVS user ID
RunAs User ID with no password	WebSphere authorized login	MVS user ID
RunAs User ID/Password	WebSphere user ID/password	MVS user ID
RunAs Unauthenticaed User	WebSphere unauthenticated user	UNAUTHENTICATED

The JCL in Example 19-6 on page 670 produced the report shown in Table 19-2 on page 671 shows how these values appear when processed by IRRADU00 and ICETOOL.

Processing the RACF SMF records

After creating the log records, the next step is to process and analyze them. RACF has two utilities that process the RACF-generated SMF records: the RACF Report Writer (RACRFRW) and the RACF SMF Data Unload Utility (IRRADU00).

IRRADU00 versus RACFRW: Which to use?

While the RACF Report Writer is a fully-supported RACF utility, we decided to use the RACF SMF Data Unload Utility (IRRADU00) for our reports and log record analysis. IRRADU00 provides greater record selectivity and more report format options.

Processing the RACF log data using the IRRADU00 utility

IRRADU00 and your sort utility are a powerful combination for producing useful reports and analysis of your SMF log data. Using the IRRADU00 utility is a two-step process:

1. Unloading the records from the SMF data set.
2. Analyzing the unloaded records. For our analysis, we used the IBM DFSORT™ program product.

The IRRADU00 utility was run against our SMF data sets with the output going into the data set named MARNEL.SC59.IRRADU00 using the JCL shown in Example 19-1.

Example 19-1 IRRADU00 JCL that unloads the SMF type 80 records

```
//MARNELAD JOB (POK,999),'M-NELSON',CLASS=A,REGION=0M,NOTIFY=&SYSUID
//SMFDUMP  EXEC PGM=IFASMFDP
//SYSPRINT DD  SYSOUT=*
//ADUPRINT DD  SYSOUT=*
//SMFDATA  DD  DISP=SHR,DSN=SYS1.SC59.MAND
//OUTDD    DD  DISP=OLD,DSN=MARNEL.SC59.IRRADU00
//SYSUDUMP DD  SYSOUT=*
//SMFOUT   DD  DISP=(NEW,CATLG,DELETE),DSN=MARNEL.SC59.SMFDATA,
//         SPACE=(CYL,(10,2,0)),DCB=(LRECL=32760,BLKSIZE=0,RECFM=VB),
//         UNIT=SYSALLDA
//SYSIN    DD  *
  INDD(SMFDATA,OPTIONS(DUMP))
  OUTDD(SMFOUT,TYPE(080:080))
  USER2(IRRADU00) USER3(IRRADU86)
/*
```

The unloaded SMF data was then used as input into the DFSORT's ICETOOL utility, which was used to perform the data analysis and to create our reports.

In the following sections, we show some of the reports that can be created.

Report: Show the accesses to EJBROLES

The JCL in Example 19-2 on page 668 shows how to create a report that shows the accesses to EJBROLEs.

Example 19-2 ICETOOL example that reports on EJBROLE accesses

```
//MARNELIC JOB (POK,999),'M-NELSON',CLASS=A,REGION=0M,NOTIFY=&SYSUID
//RACFICE   EXEC PGM=ICETOOL,PARM='MSGPRT=ALL'
//TOOLMSG   DD SYSOUT=*
//PRINT     DD SYSOUT=*
//DFSMSG    DD SYSOUT=*
//ADUDATA   DD DISP=SHR,DSN=MARNEL.TEMP.IRRADU00
//TEMP0001 DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(20,5,0)),
//             UNIT=SYSALLDA
//TOOLIN    DD *
 SORT    FROM(ADUDATA) TO(TEMP0001) USING(EJBR)
 DISPLAY FROM(TEMP0001) LIST(PRINT) -
         PAGE -
         TITLE('EJBR: USE OF EJBROLES') -
         DATE(YMD/) -
         TIME(12:) -
         BLANK -
         ON(14,8,CH)     HEADER('Qualifier') -
         ON(32,10,CH)    HEADER('Date') -
         ON(23,8,CH)     HEADER('Time') -
         ON(184,8,CH)    HEADER('Jobname') -
ON(286,30,CH)   HEADER('Role')
/*
//EJBRCNTL DD *
 SORT    FIELDS=(10,08,CH,A)
 INCLUDE COND=(5,8,CH,EQ,C'ACCESS',AND,
               578,8,CH,EQ,C'EJBROLE')
 OPTION VLSHRT
```

The report that is created is shown in Example 19-3.

Example 19-3 EJBROLE access report

```
- 1 -       EJBR: USE OF EJBROLES       03/07/30       02:49:02 pm

   Qualifier   Date         Time Jobname Role
   ---------   ---------- -------- ------- ----------------------------
   SUCCESS    2003-07-30  12:45:32  WASD5S    administrator
   SUCCESS    2003-07-30  12:45:35  WASD5S    administrator
   SUCCESS    2003-07-30  12:45:37  WASD5     monitor
   SUCCESS    2003-07-30  12:45:37  WASD5     monitor
   SUCCESS    2003-07-30  12:45:37  WASD5     monitor
   SUCCESS    2003-07-30  12:45:37  WASD5     monitor
   SUCCESS    2003-07-30  12:45:37  WASD5     monitor
   SUCCESS    2003-07-30  12:45:37  WASD5     monitor
   SUCCESS   2003-07-30   12:45:38  WASD5     monitor
   SUCCESS 2003-07-30    12:45:38   WASD5      monitor
```

You can use the X500_ISSUER and X500_SUBJECT to see the authentication mechanism that was used by WebSphere.

Report: Show a count of accesses to EJBROLES by user ID

One of the powers of ICETOOL is its ability to analyze data. In Example 19-4 we use ICETOOL to count the access to roles by user ID.

Example 19-4 ICETOOL example that counts access to roles by user ID

```
//MARNELIC JOB (POK,999),'M-NELSON',CLASS=A,REGION=0M,NOTIFY=&SYSUID
//RACFICE   EXEC PGM=ICETOOL,PARM='MSGPRT=ALL'
//TOOLMSG   DD SYSOUT=*
//PRINT     DD SYSOUT=*
//DFSMSG    DD SYSOUT=*
//ADUDATA   DD DISP=SHR,DSN=MARNEL.TEMP.IRRADU00
//TEMP0001  DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(20,5,0)),
//             UNIT=SYSALLDA
//TOOLIN    DD *
OCCURS  FROM(TEMP0001) LIST(PRINT) -
        PAGE -
        TITLE('Roll Access Count') -
        DATE(YMD/) -
        TIME(12:)  -
        BLANK -
        ON(63,8,CH)    HEADER('User ID ') -
        ON(286,30,CH)   HEADER('Role ') -
        ON(VALCNT)     HEADER('Number of Access')
//EJBRCNTL DD *
 SORT    FIELDS=(10,08,CH,A)
 INCLUDE COND=(5,8,CH,EQ,C'ACCESS',AND,
               578,8,CH,EQ,C'EJBROLE ')
 OPTION  VLSHRT
```

The output of this report is shown in Example 19-5.

Example 19-5 EJBROLE access count report

```
- 1 -       Roll Access Count        03/08/13        01:09:

User ID    Role                              Number of Access
--------   -------------------------------   ----------------
MARELI     administrator                                  677
MARELI     monitor                                       1806
MARELI     operator                                        18
MARELI     Employee                                         1
SEC2       Employee                                        17
STFAN      administrator                                  192
STFAN      monitor                                        624
STFAN      Employee                                         4
```

```
STFAN      WasSSLUser                                      3
STFAN      Worker                                          6
TOMHAC     administrator                                 672
TOMHAC     monitor                                      2432
TOMHAC     operator                                        5
USERA      Employee                                        4
WDC2STU    monitor                                       116
WDC2STU    CosNamingDelete                               168
WDC5STU    administrator                                   1
WDC5STU    monitor                                        87
```

Example 19-6 ICETOOL example that shows X500NAME information

```
//MARNELIC JOB (POK,999),'M-NELSON',CLASS=A,REGION=0M,NOTIFY=&SYSUID
//RACFICE   EXEC PGM=ICETOOL,PARM='MSGPRT=ALL'
//TOOLMSG   DD SYSOUT=*
//PRINT     DD SYSOUT=*
//DFSMSG    DD SYSOUT=*
//ADUDATA   DD DISP=SHR,DSN=MARNEL.TEMP.IRRADU00
//TEMP0001  DD DISP=(NEW,DELETE,DELETE),SPACE=(CYL,(20,5,0)),
//             UNIT=SYSALLDA
//TOOLIN    DD *
 DISPLAY FROM(TEMP0001) LIST(PRINT) -
         PAGE -
         TITLE('EJBR: USE OF EJBROLES') -
         DATE(YMD/) -
         TIME(12:)  -
         BLANK -
         ON(14,8,CH)     HEADER('Qualifier') -
         ON(32,10,CH)    HEADER('Date') -
         ON(23,8,CH)     HEADER('Time') -
         ON(63,8,CH)     HEADER('User ID') -
         ON(1597,16,CH)  HEADER('X500 Subject') -
         ON(1853,20,CH)  HEADER('X500 Issuer ') -
         ON(286,30,CH)   HEADER('Role')
//EJBRCNTL DD *
 SORT    FIELDS=(10,08,CH,A)
 INCLUDE COND=(5,8,CH,EQ,C'ACCESS',AND,
               578,8,CH,EQ,C'EJBROLE ')
 OPTION  VLSHRT
```

Table 19-2 X500NAME information.report

```
1- 1 -        EJBR: USE OF EJBROLES        03/08/13        10:56:15 pm

Qualifier  Date        Time      User ID   X500 Subject      X500 Issuer                                Role
---------  ----------  --------  --------  ----------------  -----------------------------------------------------
INSAUTH    2003-08-13  09:50:01  MARELI    MARELI            WebSphere Userid/Password                  CICSServletManager
INSAUTH    2003-08-13  09:50:25  SEC2      SEC2              WebSphere Userid/Password                  CICSServletManager
INSAUTH    2003-08-13  13:48:33  WASDFTU   UNAUTHENTICATED   WebSphere Unauthenticated User             administrator
INSAUTH    2003-08-13  14:01:35  MARNEL    MARNEL            WebSphere Userid/Password                  CEO
INSAUTH    2003-08-13  14:31:19  TAI       TAI               WebSphere Authorized Login                 monitor
SUCC SUCCESS    2003-08-13  14:11:24  WDS2STU   WDS2STU           WebSphere Server Credential                operator
```

20

Web services security

Web services provide a flexible, language-neutral way for executing application logic in a remote manner. This logic usually entails business entities. As with any computing technology that exposes functionality across the network, there is a risk of unauthorized use of the information.

This chapter provides an overview of what is currently available to secure Web services in a WebSphere Application Server for z/OS environment. For an explanation of the concepts behind Web services, see one of the several Redbooks about the topic, available from:

`http://www.redbooks.ibm.com`

You can perform a search on “Web services” or directly request *WebSphere Web Services Information Roadmap*, REDP-3854 for an overview of the information available.

The standards in this arena are in a state of constant flux, so refer to the latest documentation of the specific elements to be used before committing to a key strategy.

Support for JAX-RPC, J2EE Web services deployment model (JSR 109), WS-Security, and several other advanced Web services capabilities were introduced in WebSphere Application Server for z/OS and OS/390 V5.02.

The tooling to support these new features is contained in WebSphere Studio Application Developer Version 5.1.

20.1 Web services security

The core reasons as to why we want to secure Web services are no different than the reasons why we would want to protect any other distributed technology. Its distributed nature means that there are more parts that can be possibly compromised. The level of security depends on the context in which the application runs and the requirements of the application itself. In general, security brings overhead to application responsiveness, so it should not be applied without thought.

20.1.1 Security aspects

The following concepts define key security aspects that need to be understood and planned in any robust security strategy:

- *Integrity* refers to the assurance that the data sent from one party has not been tampered with by a third party before it arrives at its destination. If such a breach occurred, it can be detected. This is typically achieved by the use of digital signatures.
- *Confidentiality* refers to the requirement for data that is transmitted from one entity to another not to be used by a third party. This is typically accomplished by the use of encryption algorithms applied to the data itself.
- *Authentication* refers to the obtainment of an acceptable identity for the execution of a specific service. This identity can then be delegated to other systems that in turn might require authentication as well. There are many implementation-specific ways to accomplish the enforcement of this security aspect. For example, WebSphere for z/OS allows for Basic, Form-Based, Client Certificate and JAAS authentication.
- *Authorization* refers to what a valid user of a service is able to use. For example, just because you have access to a specific server does not mean you can stop it. This type of control is usually managed by specialized software, either being from a sub-system perspective like z/OS RACF, or from an application layer like Tivoli Access Manager.
- *Non-repudiation* means that in the case of a transaction between two parties (a provider and a consumer), both parties can provide legal proof to a third party that the provider delivered and the consumer received the services.

  This is typically accomplished by a combination of many technologies. For example, a seller could send a receipt digitally signed to a buyer. The buyer then could send it back with its own signature. This way the seller cannot say it did not send the item because it signed the receipt, and the buyer cannot claim it did not receive it because by signing the receipt back, it acknowledged that it did[1].

One way to implement this strategy is through a PKI infrastructure.

- *Auditing* refers to the ability to discern from gathered data who did what, when, and for what reason. This is usually made possible by the usage of monitoring tools and traces, which are then analyzed for patterns. This type activity has been engaged in for several hundred years, across many fields, so most of the general concepts applied in other fields apply. A robust auditing and traceability implementation could dissuade potential attackers.

20.1.2 Security terminology

As we discuss Web services security scenarios and their implications, a common terminology helps to deliver a more focused understanding. The following terms are often used when discussing Web services security topics:

- *Subjects* are groups of related information, each representing a single entity (for example, a person). A subject could have many principals, depending on the context. *Principals* are associated with *credentials*, which are what is actually delegated throughout a system.

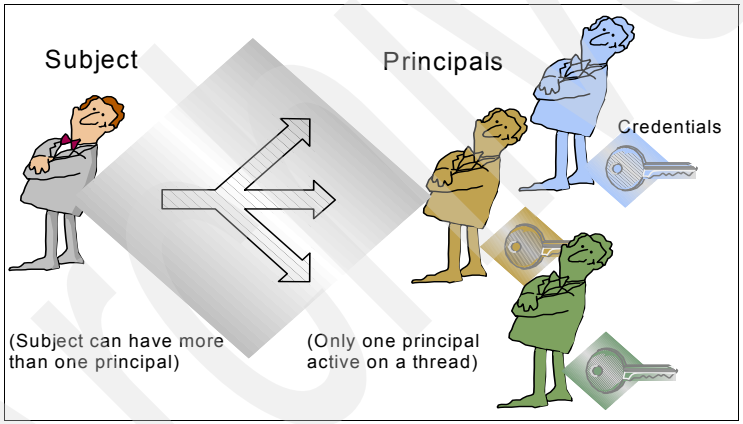


Figure 20-1 Subject and its principals

- *Security tokens* are portable entities used to carry security-related information. They are typically used to assert the authenticity of the claims asserted for a given subject.
- *Claims* are security-related statements that assert a specific right or behavior for their associated subjects.
- *Endpoint policies* are the claims required by a Web service in order for it to process a request.

[1] This does not represent a complete, mature scenario; it is mentioned only as an example.

- *Intermediaries* are applications that handle a message between its originator and its intended destination. They are very useful for enabling trust across disparate domains, ensuring scalability and helping deliver special services along the SOAP request path.

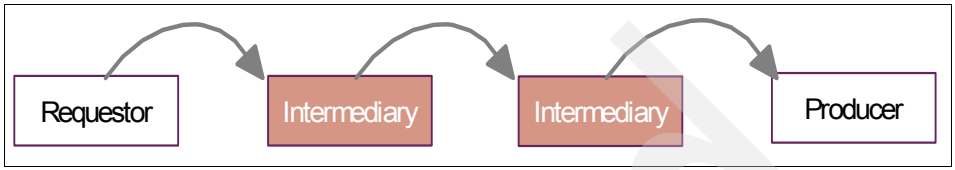


Figure 20-2 Intermediary flow

- *Actors* are intermediaries or endpoints (not browsers or users) that process SOAP messages under a specific URI identity.

20.1.3 Security standards

Here we discuss the WS-Security definition submitted to the OASIS organization, in order to provide an overview of the different security standards that should compose a robust Web services strategy.

Secure Sockets Layer (SSL)

SSL is the most popular way to encrypt communication between business partners over the Internet. Its behavior is fairly well understood. It is quite often implemented with a basic authentication and is the method often used in conjunction with SSL in the great majority of e-commerce sites.

SSL provides for transport-level protection. It is not aware of the specifics being transferred, it just creates a secure pipeline between two nodes and encrypts all traffic flowing through there.

Because most Web services flow over HTTP, SSL provides a straightforward way to provide confidentiality. It also includes a built-in communication integrity check. Connection layer authentication is achieved by the client always authenticating the serve, and optimally being authenticated by the server, through the exchange of digital certificates. SSL over HTTP is referred to as HTTPS.

If your Web services environment is simple (does not span multiple nodes), this approach could provide all the security you need. For more complex scenarios, however, it is not enough. There are three main shortcomings:

- SSL does not provide for end-to-end security if multiple nodes are involved in a message path.

  Because it does not span different nodes, an SSL context cannot be shared throughout the full path of the message. In a situation in which several intermediaries lie between the originator of the request and the provider of the Web service, there could be as many SSL handshakes as there are nodes

  Also, because there is an SSL termination activity at each node, each intermediary is able to see the unencrypted message and possibly perform modifications that might go undetected.

- SSL does not provide non-repudiation.

  Because it does not offer a mechanism that users can use to sign a message, it is not possible to guarantee that a message was indeed received by a consumer, or that the provider delivered it in the first place. Moreover, the non-application-accessible key used by SSL after its handshake is symmetric in nature.

  Asymmetric keys better facilitate non-repudiation over symmetric because there is a division of public and private keys. If the provider signs a message with its private key, the requestor can verify its validity by using the provider's public key, and vice versa.

- SSL encrypts transmitted messages in their entirety.

  It is quite common for modern day Web application scalability strategies to depend on content-based routing. For this to occur, the dispatcher needs to be able to look into the content of the message in order to make its routing decision.

  Because SSL encrypts the whole message, the dispatcher (a type of intermediary) is not able to route unless it is able to decrypt the message. some routing can occur by utilizing the SSL ID, but this does not represent the information that can be obtained from the message itself.

  Also, if we do decrypt, there might be some data that might need to be encrypted all the way to its destination (for example, credit card numbers), so this does not represent a good strategy.

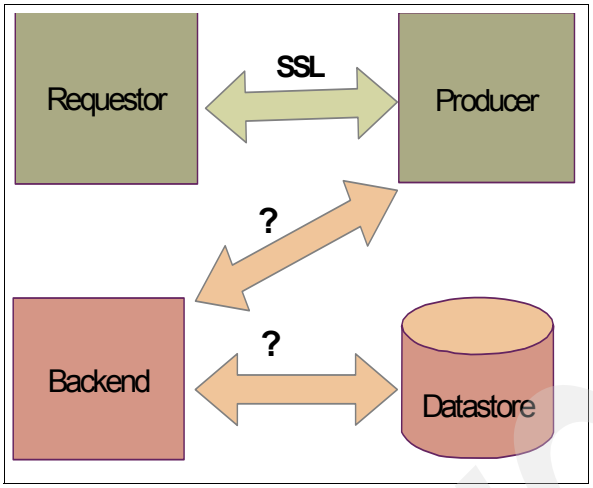


Figure 20-3 SSL and end-to-end multinode security

SSL is connection-oriented, not message-oriented, and we just described why a connection layer protocol will have trouble delegating identities. The solution to this problem is to apply the security to the SOAP message itself, as opposed to relying on the transport layer for support, and for this we need XML security.

All of this is not to say that SSL is not an appropriate solution, on the contrary, it is a very important option, used to secure the transport aspect of Web services throughout various industries. The thing to keep in mind is that SSL is a *part* of the security puzzle, not the solution itself.

XML security

XML's flexibility and maintainability is at the heart of its usefulness. One reason why it is easily manageable is because it describes its contents in clear text, but this also presents the problem that if not protected, the text can easily be copied or modified. The following section describes two solutions for providing stronger security for XML-based documents.

XML digital signature

As mentioned before, digitally signing a document allows for non-repudiation to exist during a message interchange. Technologies for signing documents have existed for some time now. What makes the XML digital signature standard different (it has achieved recommendation status with the W3C) is that it provides for a way to sign portions of an XML document.

This is very important for cases in which there are many intermediaries along the path taken by a request. These intermediaries could work with a portion of the

document and sign it when they are done, to make sure that if someone modifies their changes, it would be easily verifiable.

As Web services become more complex and possibly start to more commonly support workflow capabilities, the ability to offer non-repudiation will become more important. This standard also allows digital signatures to be represented in XML, thus fully integrating them within the XML world.

A typical XML digital signature flow can be divided into steps followed by the requestor (client side) and the provider (server side).

1. The client creates a message to be sent.
2. The client generates a hash value form message.
3. The client transforms the signature definition element (including hash value) to a canonical form.
4. The client uses its private key to create a digital signature of the material.
5. The client sends the message, signature, and certificate (with public key) to the server.
6. The server transforms the message to a canonical form.
7. The server uses the client public key to generate a digest of received material.
8. The server compares the generated digest with the received digest. If they match, the message passes the integrity test. If they do not match, it shows that the message has been tampered with somehow.

There might be some variations in the steps executed in particular scenarios, but the actions will typically involve running the data to be signed through some type of canonicalization process. Canonical algorithms provide a way to express an XML document in a way that transcends a specific encoding, structure or ordering, in order for the same document to be considered equivalent across XML parsers.

```
<Signature Id="SampleSignature" xmlns="http://www.w3.org/2000/09/xmidsig#>
   <Signedinfo>
      <CanonicalizationMethodAlgorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
      <SignatureMethod Algorithm="http://wwww.w3.org/2000/09/xmldsig#dsa-sha1"/>
      <Reference URI="http://wwww.w3.org/TR/2000/REC-xhtml-20000126/">
         <Transforms>
            <Transform Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
         </Transforms>
         <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <DigestValue>DsgFGDERHVsrsFwjaVDHddfgdfPGsweGrTDnkigeskjd=</DigestValue>
      </Reference>
   </SignedInfo>
   <SignatureValue> SEfgfFDGjLTgvYcJNNJTrfygGrGEtTyEw5RwerWOgf==</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <DSAKeyValue>
           <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
        </DSAKey/Value>
      </KeyValue>
    </KeyInfo>
</Signature>
```

Figure 20-4 XML digital signature sample

The signature sample in Figure 20-4 shows the various tags that compose the envelope:

- A Signature element defining an ID and a specific namespace
- A Reference element specifying a URI that points to the XML document being signed (not shown)
- A DigestValue element whose value is the digest obtained from the XML document
- A SignInfo element, which contains the data actually signed, together with information regarding the algorithm used, the canonization algorithm used, the digest, and so on

The canonicalization form of the SignInfo element is used as one of the inputs for the digital signature algorithm. Any changes to the original XML document would

result in a change in the canonicalization form, thus changing its resulting digest. This would invalidate the digital signature.

There are three main different types of signatures, each differentiating itself by the way it relates to the data to be signed:

- *Encompassing XML signature* defines a signature that contains the signed data within the tags defining the signature boundaries (see Figure 20-4 on page 680).
- *Encompassed XML signature* defines a signature that is contained within the signed document.
- *Detached XML signature* defines a signature that stands separately from the signed data.

Because several XML digital signatures can coexist in the same document, there can be many combinations of the types stated here in a single document.

Because a document can be signed several times, it is recommended that signers keep in mind that if the signed document is not visible (because of it being encoded or encrypted), and then non-repudiation is no longer guaranteed because the signer cannot be completely sure what exactly is being signed. In this type of situation, the signer could decide to decode or decrypting the text before signing it. However, careful thought should be given to such a strategy because the signing process could quite easily become convoluted.

Non-XML digital signature standards provide a version field in their definition, but this is not so with XML signatures. Different versions of given signatures are kept separately by way of XML namespaces. The sample in Figure 20-4 on page 680 uses the standard "`http://www.w3.org/2000/09/xmldsig#`" namespace. A new version would use a new namespace.

It should be noted that the XML digital signature standard does not offer a standard API. In Java, this is being addressed by JSR 105. Also, we have not discussed how the XML digital signature specification works with SOAP. This is actually described in a standard called WS-Security, managed by the OASIS organization. It defines the specific notation and schemas used to describe the needed security headers and body elements of a SOAP message. This type of meta description is also delivered for XML encryption. WS-Security is described in "WS-Security" on page 684.

XML encryption

The XML encryption standard allows for the encryption of specific parts of a document. It also allows various parts of the same document to receive different encryption treatments. There are many technologies that provide encryption

capabilities to a program. The difference is that they do not represent their encrypted data as XML.

These capabilities are important in complex Web services scenarios in which several actors might need to see some information in order to process the message, but do not need to see all the data contained in the message.

This standard, just as XML digital signature, has achieved recommendation status with the W3C.

```
<?xml  version='1.0'?>
<Payment xmlns='http://payment.com'>
    <CreditCard>
        <EncrytedData
Type='http://www.w3.org/2001/04/xmlenc#Element'xmlns='http://www.w3.org/2001/04/xmlenc#'>
            <CipherData>
                <CipherValue>A23B45C56</CipherValue>
            </CipherData>
        </EncrypterData>
     </CreditCard>
    <expiry-date>
        12/04
    </expiry-date>
</PaymentInformation>
```

Figure 20-5 XML encryption sample

The encryption sample Figure 20-5 shows a few of the tags that compose the envelope:

- EncryptedData element is the core element of the XML encryption standard. It replaces the element being encrypted and contains the actual encrypted data (unless the detached encryption strategy is followed (as described in the following section).
- CipherData element is the EncryptedData child element that contains the encrypted data (unless the detached encryption strategy is followed). It is a mandatory element.

The reference to a remote encrypted data (not shown in Figure 20-5) is delivered by the CipherReference element. Other elements perform various tasks, like the EncryptedKey element that is used to transport encryption keys, and the EncryptionProperties element that is used to deliver extra information regarding the generation of EncryptedKey or EncryptedData.

A typical XML encryption flow can be divided into steps followed by the requestor (client side) and the provider (server side):

1. The client chooses an encryption algorithm.
2. The client obtains the encryption key.
3. The client obtains the UTF-8 encoded data to encrypt.
4. The client performs encryption using the key and the encryption algorithm.
5. The server processes the document in order to extract each EncryptedData, CipherData, Key information, and so on.
6. The server locates the key (the key could be local or remote).
7. The server decrypts the data.
8. The server processes the XML elements.

It should be noted that the XML encryption standard is flexible regarding the roles a client (encryptor) and a server (decryptor) follow, understanding that some of the steps could be performed by either one.

Following the classification used with XML digital signatures, there are two main types of encryption strategies:

- *Encompassing encryption* consists of embedding the encrypted information inside the XML document.
- *Detached encryption* consists of referring to an external encrypted entity form within the XML document.
- *Encompassed encryption* means something different than in the encompassed digital signature approach. It does not make sense to encrypt all information about the encryption, including the message, keyring, and so on, because this would mean that in order to decrypt the message, you need to decrypt it in order to get the information necessary for the decryption. This is obviously not a feasible approach.

  Encompassed encryption refers to the approach of encrypting the data with a symmetric key and then encrypting that key with a public key.

XML encryption also does not offer a standard API. In Java this is being addressed by JSR 106.

The versioning strategy for XML encryption is the same as for digital signatures: namespaces. Instead of the standard "`http://www.w3.org/2001/04/xmlenc#`" shown in Figure 20-5 on page 682, a new version would use a new namespace.

WS-Security

This specification is supported in WebSphere Application Server for z/OS in a limited mode (given that it is still evolving) beginning with V5.02. Because WS-Security is the Web service security specification with the greatest following and support throughout the industry, it has become the de facto standard (even though it is still in *draft* mode). As a result, WS-Security is described in this section to provide an introduction to its main concepts and raise awareness of its components.

WS-Security provides a foundation set of security specifications that can be used when building secure Web services. These specifications, when completed, will define a robust, end-to-end security architecture. They provide platform-neutral interoperability and same-domain and cross-domain secure interchange.

This specification describes extensions to SOAP that deliver enhanced quality of protection by means of message integrity and confidentiality. This is implemented by allowing the addition of security tokens to SOAP messages.

With WebSphere Application Server for z/OS and OS/390 Version 5.1, support was added for the following authentication capabilities:

- Basicauth
- Identity assertion
- Signature
- Pluggable token (for example, Kerberos tickets and x.509 certificates)

These capabilities are configured at the application level through the webservices.xml file.

All this support is enabled by the use of WS-Security-defined SOAP headers.

WS-Security encourages decentralization, because, in order for two parties to exchange information securely, they only have to agree on the syntax and the underlying semantics. Other elements, such as the operating system and language of implementation, can be totally different.

WS-Security was designed as a flexible, composable architecture. It supports and integrates with several well-known technologies for message-level security. It works in conjunction with XML encryption to provide confidentiality and XML digital signature to ensure integrity. The extensibility is also showcased by its support of different security approaches such as Kerberos, X.509, and others through the definition of specific bindings.

SOAP provides the foundation from which this standard is built. On top of WS-Security, there are many other standards. They all work together to describe the Web services security strategy, as shown by Figure 20-6 on page 685. Note

WS-Authorization and WS-Privacy. These are specifications that were not yet released at the time of writing. This is a dynamic area; they should be released in the not so distant future.

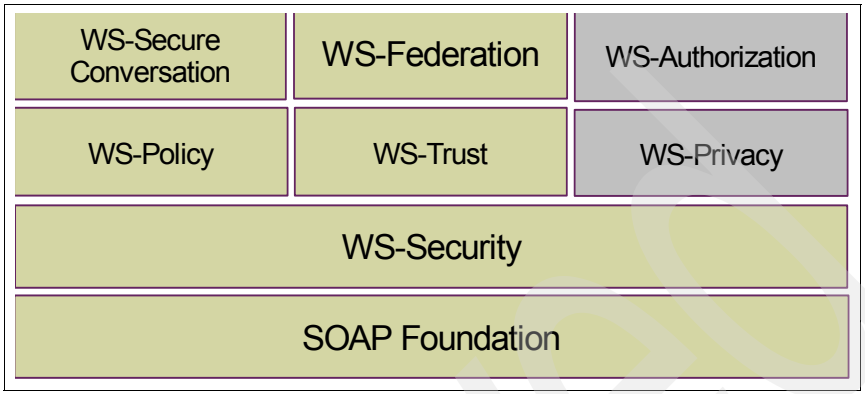


Figure 20-6 WS-Security protocol stack

The following sections provide descriptions of the main protocols that compose the WS-Security stack.

WS-Policy

This specification defines an abstract policy model. It defines how requesters and providers can specify policies that affect their capabilities and requirements when conducting exchanges, and how to attach those policies to SOAP messages.

It consists of four specifications:

- *WS-Policy Framework* describes a general purpose strategy for delivering the policies of a Web service. It serves as the base for policy-based features that can be extended by the other, related Web services specifications.
- *WS-Policy Attachments* defines how subjects are associated to policies and how policy providers bind to the policy framework.
- *WS-Policy Assertions* defines the common assertions that can be specified in a policy (for example, natural language and character encoding). It is strongly advised that policies and assertions be *signed* to prevent tampering.
- *WS-Security Policy Language* identifies WS-Security's policy assertions for use with WS-Policy.

WS-Trust

This specification defines how trust relationships are established, either directly or through intermediaries. It allows for support for impersonation and delegation

(which is a more traceable type of impersonation) of principals. This means that intermediaries can insert security tokens inside SOAP messages.

WS-Privacy

This planned specification will allow for the enforcement of predefined privacy policies using a combination of WS-Security, WS-Policy and WS-Trust. Privacy policies can require SOAP messages to contain claims that the requestor conforms to specific privacy policies.

WS-SecureConversation

This standard provides a mechanism to deliver end-to-end authentication in a Web services environment. It is built on the lower layer of standards to accomplish that task. Basically, the consumer establishes a security context with the provider after the authentication process. This is protected, like in SSL, by a mixture of symmetric and asymmetric algorithms. WS-SecureConversation provides an SSL-like quality of protection to the SOAP message layer.

WS-Federation

This specification describes how to manage trust relationships between service providers in a federated environment. This includes support for federated identities.

WS-Authorization

This planned specification will define how to handle authorization policies and exchanges. This is important because various security providers might have very different authorization implementations.

20.1.4 Other standards

The first two standards described here are sometimes present in Web services security solutions (especially involving Microsoft .NET integration). The last three are up-and-coming standards. They are mentioned here because, depending on the architecture chosen, they could become a central part of a security implementation.

Kerberos

Throughout this chapter we have been discussing methods for providing security mainly based on a PKI approach, that is, having a private and public set of keys available. This typically involves a x.509 certificate to transport the public key and such. It should be noted that utilizing Kerberos tokens has become more and more common, especially since Microsoft started supporting Kerberos Version 5 as the default network authentication protocol in their Windows 2000 operating system.

SPNEGO

IETF's RFC 2478 describes the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) standard. It is a security negotiation mechanism for the Generic Security Service Application Program Interface (GSS-API). This API provides a layer of abstraction over common security mechanisms (message privacy and integrity, authentication, and so on). This API is commonly implemented on top of Kerberos. JSR 72 is currently in the works to provide this functionality in the Java language.

Microsoft has added support for this protocol to its Internet Explorer Version 6 Web browser and its Internet Information Server (IIS). This allows for a Kerberos-based authentication strategy between IIS and Internet Explorer.

SAML

Secure Assertion Markup Language (SAML) is the leading standard for establishing XML-based security assertions. Assertions deliver a portable mechanism for the exchange of XML-based security information across a network. Assertions are similar to *claims* (which are defined in 20.1.2, "Security terminology" on page 675). A principal can have various assertions grouped as a profile.

Each of these assertions could be from different organizations. This allows for integrating disparate security domains in a loosely-coupled manner, as the assertion from one can be used in some other domain. This could be used, for example, to enable single sign-on (SSO) scenarios across multiple domains (for example, business partners' environments).

There are three types of SAML assertions: authentication, authorization and attribute (which helps assert a certain security outcome for a given attribute).

It should be noted that SAML does not perform authentication or authorization; it only provides mechanisms for supporting assertions about the security results of those operations.

XACML

XML Access Control Markup Language (XACML) delivers a mechanism for exchanging authentication policies without exchanging specifics about how they are implemented.

The specification describes some terms that define the components in the XACML exchanges. The following are the two most common:

- Policy Enforcement Points (PEP) serves as the component that enforces the defined policies.

- Policy Decision Point (PDP) serves as the component that evaluates the merits of a request against an existing policy and returns the result.

The other components deal with administrative functions: the Policy Administration Point (PAP) deals with policy creation, the Policy Information Point (PIP) deals with providing information regarding attribute values, and the Policy RetrivalPoint (PRP) deals with the retrieval of policy information for a particular request.

SAML and XACML complement each other in that XACML delivers the actual response for the assertions SAML is delivering. For example, a consumer could send a SAML authorization request to a destination. The destination then needs to find out if the consumer is able to execute certain Web service operations. The destination is considered a PEP, so it needs to contact a PDP in order to establish an outcome. That PDP more than likely will then contact some type of role database. After the PED receives the result, it returns either a SOAP fault, if the outcome is negative, or a SAML response, if the authorization was successful. The interaction between the PED and PDP is specified by XACML.

The use of XACML allows us to deliver a consolidated view of disparate ACL repositories. This facilitates the management, monitoring, and control of these entities in an enterprise.

The adoption of this standard in the industry has been somewhat slower than for SAML. This is partly due to it covering more complex terrain.

XKMS

XML Key Management Specification (XKMS) simplifies the integration of Public Key Infrastructure (PKI) with XML applications and Web services by shielding the programmer from the typically complex details of a specific PKI implementation. This delivers a reduction of the complexity of client code, helping ease its development.

XKMS provides a Web service view to the PKI world. In fact, its specification is divided into three services:

- Register Service is used for registering PKI key pairs.
- Locate Service is used to retrieve the registered key pairs.
- Validate Service provides the facilities to validate that registered keys are indeed valid.

Another key item that XKMS provides is centralized trust management. This is important because one of the typical setbacks experienced by large PKI implementations deals with the policy management mechanism, or the lack of such a mechanism. XKMS allows to set PKI policies at the enterprise level, thus

gaining some control back from the situation in which the user wants to set its own security.

20.1.5 Best practices

The following security recommendations can be used as a set of starting guidelines for a more robust Web services security implementation.

Keep in mind that no approach is ever fully secure and that given the dynamic nature of information technology, it is advisable to routinely analyze your environment for elements that might need revision.

Secure the RPCRouter servlet

Because all RPC and message requests go through the RPCRouter (even in the case of document-style, those messages are then routed to the MessageRouter servlet), the RPCRouter servlet should be secured. Because RPCRouter is a servlet, the same security offered to any other servlet by the Web container can be used. This entails assigning identities to roles and enabling WebSphere global security.

Web services as session beans for fine-grained security

All WebSphere Studio Application Developer operations in Apache SOAP share the same user ID, so if finer-grained authorization is needed, EJB containers provide an excellent way to control access at the method level. In order to accomplish this, develop the Web service implementation as a stateless session bean.

Establish precautions regarding WebSphere Studio Application Developer availability

It is quite easy to generate a client from only WebSphere Studio Application Developer. Because it is typically easily accessible, make sure you agree with what exposing WebSphere Studio Application Developer entails. WebSphere Studio Application Developer does not have to be exposed as freely if the application is in a controlled environment (for example, intranets). If WebSphere Studio Application Developer files are exposed, make sure you have the security in place to prevent misuse.

Use cryptography during intermediary processing

If data needs to be protected end to end, remember that intermediaries are able to inspect your data unless the data is encrypted (not just SSL-protected). This might not be fully possible, as some intermediaries might be required to do some modification on portions of the document by design (for example, a purchase

order approval application). Take advantage of the XML encryption and digital signature support for selective signing and encryption in order to ensure integrity, confidentiality, and non-repudiation.

Consider issuing time stamps with digital signatures

A signed message can be subject to a replay attack in which a signed message is recorded and re-sent in order to obtain the information accessed by the original exchanged. The chances of this occurring can be dramatically reduced if digital signatures are combined with nonces and time stamps. There are several ways to implement this strategy; one possible way is to encode such elements within the SOAP message as XML digital signature properties. When WS-Security becomes supported, the <Timestamp> element of the security header could be used to carry the generated time stamp.

Establish auditing mechanisms

Understanding how you are going to track Web services interactions is very important for problem tracing and possible intrusion detection, especially if your Web services are publicly published. There are many ways to implement this, especially if all you want is to track down HTTP/SOAP interactions. Check your topology's software to see what your out-of-the-box choices for logging are. All enterprise Web servers and application servers provide some type of advanced logging functionality. If you plan to do some auditing at the application layer, consider an asynchronous approach, which could be accomplished through the use of JMS or Java Sockets. Make sure you are auditing any intermediaries as well, if possible. Remember that logging can easily become a bottleneck if not implemented correctly.

Consider establishing a XML filtering mechanism

Most of the existing routing and security artifacts tend to work at the transport layer or below. It would be beneficial to have a solution that provided management in a layer that can take advantage of Web services specifics. A good example of such an offering is the Web services gateway. This is a WebSphere Web services application that provides a multichannel, code-free, one point of control of the different Web services deployed in your organization. It provides a filtering mechanism (to deliver regulation of a request before and after the invocation) together with a configurable logging mechanism.

Establish attacks precautions

Typical Web services are at their core HTTP-based applications. If they are publicly available, they are susceptible to the same type of danger as any other Web application: Denial of service, spoofing, Web server bug exploitation, and so on. All of these are real possibilities, so make sure you are aware of the dangers and exercise caution by having robust security policies and procedures.

Part 5

Appendixes

This part contains an appendix that offers additional information about security tasks. We also provide information about obtaining the Additional material available with this book in an appendix.

A

Setup and debugging guides

This appendix provides supplemental installation, troubleshooting, and setup information related to security tasks described in this redbook.

Configuring SSL between WebSphere Application Server and a CICS Transaction Gateway daemon

In this section, we describe how to configure SSL between WebSphere and a CICS Transaction Gateway daemon. We show how to create self-signed certificates for the CICS Transaction Gateway server and the WebSphere servant in order to illustrate the use of the newer versions of the gskkyman utility (part of the optional Cryptographic Services feature of z/OS), keytool, and the key management utility (iKeyman).

Note: In z/OS 1.4, the gskkyman utility is slightly changed. Refer to “Detailed steps for using gsskyman” on page 696 for instructions on how to use this utility.

Also, WebSphere Application Server for z/OS V5 contains a later version of the iKeyman utility; refer to “Using iKeyman on Windows” on page 705 for setup instructions for this utility.

Chapter 8 of the Redbook *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133 and *CICS Transaction Gateway z/OS Administration*, SG34-6191 provide details, so here we offer only a brief overview of the main steps.

To use System SSL between WebSphere Application Server for z/OS and the CICS Transaction Gateway daemon, follow these steps:

1. Prepare the System SSL libraries to be program-controlled.

   To determine whether this has already been done, check the status by changing to each of the following directories and issuing an **`ls -E`** command for each:

   – cd /usr/lpp/gskssl/lib

   – cd /usr/lpp/gskssl/bin

   Verify that the p bit is set in the second column. If it is not there, set it using the **`extattr`** command.

   Make sure that both your user ID (the one you will use to run gsskyman) and the RACF user ID of the CICS Transaction Gateway daemon have access to the CSK profile in class CSFSERV, otherwise you will receive an error like the following:

   ```
   ICH408I USER(STC   ) GROUP(SYS1  ) NAME(STARTED TASK ID) 393 CSFCKI
   CL(CSFSERV)
   INSUFFICIENT ACCESS AUTHORITY
   ```

```
FROM ** (G)
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

2. Use the gskkyman utility to create a key database (.kdb file); refer to “Detailed steps for using gsskyman” on page 696 for details.
3. Make sure that this .kdb file is also marked as program-controlled:

   ```
   extattr +p /your_dir/yourkeydatabase.kdb
   ```

4. Generate an end-user certificate for the CICS Transaction Gateway server and export the certificate to a file.
5. Configure your CICS Transaction Gateway daemon to support the SSL protocol.

   You can use the ctgcfg graphical tool or make changes to configuration files manually. You must tell the CICS Transaction Gateway to use the key database you have created by using ctgcfg or by setting the keyring and keyringpw properties in CTG.INI.

   You also need to set the protocol statement in CTG.INI to reference the key database (.kdb file) you have created; see Example A-1.

Example: A-1 Protocol settings for system SSL in CTG.INI

```
protocol@systemssl_ssl.handler=com.ibm.ctg.server.GskSslHandler
protocol@systemssl_ssl.parameters=port=8050;sotimeout=1000;\
connecttimeout=2000;idletimeout=600000;pingfrequency=60000;\
keyring=/your_dir/your_keydatabase.kdb;keyringpw=default;clientauth=on;
```

6. To use SSL *client* authentication, either use ctgcfg to select the box to enable client authentication, or add the clientauth=on option to the protocol property in CTG.INI, as shown in Example A-1.

   Note that, while you might want to consider using SSL client authentication in the CICS Transaction Gateway daemon to ensure the requests come from the correct WebSphere servant, you will probably *not* want to associate WebSphere’s client certificate with a RACF user ID using RACDCERT, because this would prevent the current Java identity from being flowed to CICS.

7. Use keytool from z/OS UNIX or the graphical iKeyman tool to create a JSSE keystore to hold the certificates that will be used for the WebSphere-CICS Transaction Gateway connection.
8. Using the WebSphere administrative console, set the ConnectionURL property of your ECI Resource Adaptor connection factory to ssl://<ip_address_of_CTGDaemon>

9. Set the PortNumber property of your ECI Resource Adaptor connection factory to match the port number that CICS Transaction Gateway is listening on for SSL requests, as shown in Example A-1 on page 695.
10. Use the keytool utility (which invoked iKeyman) to create a keystore for WebSphere (if one does not already exist), and import the CICS Transaction Gateway daemon's signer certificate.
11. If the CICS Transaction Gateway daemon will perform SSL client authentication, you must use keytool to create a certificate for WebSphere in JSSE keystore and export WebSphere's signer certificate to a file. Then import WebSphere's signer certificate into the CICS Transaction Gateway daemon's key database using gsskyman.

Detailed steps for using gsskyman

To use gsskyman, complete the following steps:

1. Start gsskyman and create a key database (.kdb file). In our case, create a directory called kdb under the CICS Transaction Gateway /bin directory.

   Tip: You should probably not put the kdb directory under the CICS Transaction Gateway install path because you might lose your customization when the CICS Transaction Gateway is upgraded. Instead, create a separate mount point and mount a small HFS data set to hold CICS Transaction Gateway daemon customization files.

2. Go into z/OS UNIX and change to the directory where you want to create a directory for your key database. Then make the directory and change to it:

```
SEC2@SC59:/u/sec2> cd /usr/lpp/ctg/bin
SEC2@SC59:/u/sec2> mkdir kdb
SEC2@SC59:/u/sec2> cd /usr/lpp/ctg/bin/kdb
```

3. Type `gsskyman` on the command line to start gsskyman.

   Note: This utility should be in the default system path. If it is not, you will need to add /usr/lpp/gskssl/bin to your path:

```
SEC2@SC59:/usr/lpp/ctg/bin/kdb> gskkyman
```

4. You now get the menu of gskkyman. The options you need to choose and text you need to enter are highlighted in bold in Figure A-1 on page 697.

```
Database Menu

  1 - Create new database
  2 - Open database
  3 - Change database password
  4 - Change database record length
  5 - Delete database

  0 - Exit program

Enter option number: 1
Enter key database name (press ENTER to return to menu): ctgserver.kdb
Enter database password (press ENTER to return to menu): *******
Re-enter database password: ********
Enter password expiration in days (press ENTER for no expiration): 365
Enter database record length (press ENTER to use 2500):

Key database /usr/lpp/ctg/bin/kdb/ctgserver.kdb created.

Press ENTER to continue.
```

Figure A-1 Database menu

Press Enter to get the next menu.

```
Key Management Menu

      Database: /usr/lpp/ctg/bin/kdb/ctgserver.kdb

  1 - Manage keys and certificates
  2 - Manage certificates
  3 - Manage certificate requests
  4 - Create new certificate request
  5 - Receive certificate issued for your request
  6 - Create a self-signed certificate
  7 - Import a certificate
  8 - Import a certificate and a private key
  9 - Show the default key
 10 - Store database password
 11 - Show database record length

  0 - Exit program

Enter option number (press ENTER to return to previous menu): 6
```

Figure A-2 Key management menu

5. Select option 6 to create a self-signed certificate. You get the Certificate Type menu.

```
 Certificate Type

   1 - CA certificate with 1024-bit RSA key
   2 - CA certificate with 2048-bit RSA key
   3 - CA certificate with 1024-bit DSA key
   4 - End user certificate with 1024-bit RSA key
   5 - End user certificate with 2048-bit RSA key
   6 - End user certificate with 1024-bit DSA key

Select certificate type (press ENTER to return to menu): 4
```

Figure A-3 Certificate type

Note: You need a self-signed end-user certificate, not a CA.

6. In response to the prompts shown in the next menu, enter the information appropriate for your organization.

```
Enter label (press ENTER to return to menu): ctgserver
Enter subject name for certificate
  Common name (required): ctgserver
  Organizational unit (optional): itso
  Organization (required): ibm
  City/Locality (optional): pok
  State/Province (optional): NY
  Country/Region (2 characters - required): US
Enter number of days certificate will be valid (default 365): 365

Enter number of days certificate will be valid (default 365):

Please wait .....

Certificate created.

Press ENTER to continue.
```

Figure A-4 Enter information appropriate for your organization

7. Press Enter; you will be returned to the Key Management Menu. Now, you need to set this as the default key for the kdb.

```
Key Management Menu

      Database: /usr/lpp/ctg/bin/kdb/ctgserver.kdb

  1 - Manage keys and certificates
  2 - Manage certificates
  3 - Manage certificate requests
  4 - Create new certificate request
  5 - Receive certificate issued for your request
  6 - Create a self-signed certificate
  7 - Import a certificate
  8 - Import a certificate and a private key
  9 - Show the default key
 10 - Store database password
 11 - Show database record length

  0 - Exit program


Enter option number (press ENTER to return to previous menu): 1
```

Figure A-5 Key management menu

8. You are prompted to choose the certificate.

```
Key and Certificate List

     Database: /usr/lpp/ctg/bin/kdb/ctgserver.kdb

  1 - ctgserver    <-- (this is the certificate we want)

  0 - Return to selection menu

Enter label number (ENTER to return to selection menu, p for previous
list): 1
```

Figure A-6 Key and certificate list

9. Now set this as the default key.

```
Key and Certificate Menu

       Label: ctgserver

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu): 3

Default key set.

Press ENTER to continue.
```

Figure A-7 Key and certificate menu

10. Press Enter and you return to the Key and Certificate Menu. Now, you should set the certificate as trusted.

```
 Key and Certificate Menu

       Label: ctgserver

   1 - Show certificate information
   2 - Show key information
   3 - Set key as default
   4 - Set certificate trust status
   5 - Copy certificate and key to another database
   6 - Export certificate to a file
   7 - Export certificate and key to a file
   8 - Delete certificate and key
   9 - Change label
  11 - Create a certificate renewal request

   0 - Exit program

Enter option number (press ENTER to return to previous menu):   4
Enter 1 if trusted, 0 if untrusted (press ENTER to return to menu): 1

Record updated.

Press ENTER to continue.
```

Figure A-8 Key and certificate menu

11. Press Enter and you return to the Key and Certificate Menu. Now, you should export the certificate for later importing into the WebSphere key store.

```
Key and Certificate Menu

      Label: ctgserver

  1 - Show certificate information
  2 - Show key information
  3 - Set key as default
  4 - Set certificate trust status
  5 - Copy certificate and key to another database
  6 - Export certificate to a file
  7 - Export certificate and key to a file
  8 - Delete certificate and key
  9 - Change label
 11 - Create a certificate renewal request

  0 - Exit program

Enter option number (press ENTER to return to previous menu): 6
```

Figure A-9 Export the certificate

12. You receive a menu that enables you to select the export format.

```
 1 - Binary ASN.1 DER
   2 - Base64 ASN.1 DER
   3 - Binary PKCS #7
   4 - Base64 PKCS #7

Select export format (press ENTER to return to menu): 1
```

Figure A-10 Select the export format

> **Note:** You choose Binary ASN.1 DER here to keep the certificate on the same platform. If you want to export the certificate for use with the graphical iKeyman tool on Windows, choose Base64 ASN.1 DER.

13. Now you are prompted to enter the file name for the exported certificate.

```
Enter export file name (press ENTER to return to menu):
 ===> ctgserver.arm
Certificate exported.

Press ENTER to continue.
 ===>
```

Figure A-11 The file name for the exported certificate

14. Press Enter three times to return to the main menu.

15. Then, use option 0 to exit.

Now, you need to create a JKS for WebSphere. Then, you will import the exported ctgserver certificate into the JKS using iKeyman. (You could import the ctgserver certificate into an existing JKS, if you prefer. We thought it safer to create a new key store.)

Using keytool to drive iKeyman

The CICS ECI resource adaptor uses the JSSE implementation of SSL. In order to use SSL over a connection from WebSphere Application Server for z/OS to a CICS Transaction Gateway daemon, the CICS ECI connection factory needs to know the location of the JSSE keystore that contains the certificate and public key of the CICS Transaction Gateway daemon. If you are not using a self-signed certificate for the CICS Transaction Gateway daemon you will also need to import the certificate and public key of the Certification Agency into that key store.

WebSphere already has some JSSE key stores, created during configuration of the node or base cluster. They can be found in /<websphere_config_root>/appserver/etc.

You could use one of these .jks files, but when you are testing, it might be safer to create a new one. If you make a mistake using one of the existing key stores, you could disable a WebSphere function and even prevent your server from initializing properly.

On z/OS, you can use the command line keytool utility, which is described in the *WebSphere Application Server for z/OS Information Center*. Here, we show how to use it only to create a database, import a self-signed certificate, and export a certificate.

Note: In order to run keytool, make sure that you have the <JAVA_HOME/bin in your PATH.

Creating the JSSE key store

Create and execute a shell script (we called it keystore.sh), as shown in Example A-2.

Example: A-2 Using keytool to create a JSSE keystore and certificate for WebSphere

```
#!/bin/sh
# keystore

 echo "Creating key store with self-signed certificate"

 #  Change to the directory where we will create the key store
 cd /WebSphere/BSOO/appserver/etc

 keytool -genkey -alias wasdO.ctg -keysize 1024 \
 -dname "cn=wtsc59.itso.ibm.com,o=ibm,ou=itso,l=pok,s=NY,c=US" \
 -keystore wasdOctg.jks -keypass password -storepass password -keyalg RSA
```

If the shell script is in a directory that is in your current PATH, you should be able to execute it by typing `keystore` on the z/OS UNIX command line.

Otherwise, change to the directory that holds the shell script and type: `./keystore` on the command line.

You can check the contents by running keytool in a shell script (we called it keylist.sh) with the -list option, as shown in Example A-3.

Example: A-3 Listing the contents of the JSSE key store

```
#!/bin/sh
# keylist

echo "List contents of key store"

cd /WebSphere/BSOO/appserver/etc

keytool -list -v \
-keystore wasdOctg.jks -storepass password
```

Now you can import the self-signed certificate of the CICS Transaction Gateway daemon. In our case, we used a shell script called keyimp.sh, as shown in Example A-4 on page 705.

Example: A-4 Import CICS Transaction Gateway daemon's certificate to JSSE key store

```
#!/bin/sh
# keyimp
echo "importing certs to keystores"

keytool -v -import -alias 'ctgserver' \
   -file /usr/lpp/ctg/bin/kdb/ctgserver.arm \
   -keystore /WebSphere/BSO0/appserver/etc/wasdOctg.jks \
   -storepass password -noprompt \
   -storetype JKS -provider com.ibm.crypto.provider.IBMJCE
```

Finally, export WebSphere's certificate for later importing into the CICS Transaction Gateway daemon's kdb. We created a shell script called keyexp.sh, as shown in Example A-5

Example: A-5 Exporting a certificate from a JSSE key store

```
#!/bin/sh
# keyexp
echo "exporting certs from keystores"

keytool -v -export -alias 'wasdO.ctg' \
   -file /WebSphere/BSO0/appserver/etc/wasdOctg.arm \
   -keystore /WebSphere/BSO0/appserver/etc/wasdOctg.jks \
   -storepass password -noprompt \
   -storetype JKS -provider com.ibm.crypto.provider.IBMJCE
```

Tip: The syntax is a bit tricky, so why not run keytool with no options from z/OS UNIX to get help information about the syntax. Copy the syntax for the operation you want to perform into the shell script you are creating, and edit the pasted help information to use the names and properties you want. This method ensures that your command contains all the options you need, and avoids typing errors on the option names.

Using iKeyman on Windows

You can use the graphical iKeyman utility to create and maintain certificates in a JSSE .jks key store rather than using the command line keytool.

Note: The IBM Redbook *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133 suggests that you can use iKeyman to maintain certificates in JSSE key stores (.jks files), but we found that the iKeyman provided with CICS Transaction Gateway V5.01 (cfwk.zip and gsk5cls.jar from /usr/lpp/ctg/classes) did not seem to support .jks files.

We installed iKeyman V6.0.0, which is supplied with WebSphere Application Server for z/OS and this version does support .jks files. With iKeyman, you can:

- Request and receive a digital certificate from a Certification Authority (CA)
- Generate self-signed certificates
- Add certificates to your KeyRing files
- Change your KeyRing password
- Set a private key as the default
- Delete a key
- Export a key by copying it to a file
- Import a key from an exported copy and add it to a KeyRing

Installing and running iKeyman on Windows

If you have installed Java SDK 1.3.1 or later on Windows and have access to a WebSphere V5 distributed system, you could obtain iKeyman from the WebSphere distributed <WebSphere_install_root>/bin directory.

We downloaded iKeyman and other Java .jar files it needs from JDK 1.3.1 on z/OS and the WebSphere Application Server for z/OS /lib directory. In this case, you still need to make sure that your workstation has the minimum level of Java support, that is, IBM Java SDK 1.3.1.

To set up iKeyman, create a directory to hold the utility (we created a directory called c:\iKeyMan), and then FTP the following .jar files in binary mode to the iKeyMan directory on your workstation. These .jars can be found in JDK 1.3.1 on z/OS in JAVA_HOME/lib/ext, which normally resolves to /usr/lpp/java/IBM/J1.3/lib/ext.

- ibmjceprovider.jar
- ibmjcefw.jar
- ibmpkcs.jar

The following .jar files are also required to run iKeyman, but we found that they were already installed in the /lib directory of the Java runtime on the workstation:

- local_policy.jar
- US_export_policy.jar

Then ftp the following .jar file in binary mode to the iKeyman directory on your workstation. This .jar can be found on z/OS in <WebSphereV5_install_root>/lib/security.

- gskikm.jar

Add the following .jar files to the CLASSPATH environment variable:

- ibmjceprovider.jar
- ibmjcefw.jar
- ibmpkcs.jar
- gskikm.jar

Now you need to update the java.security file for your Java runtime to register the IBMJCE and JSSE providers. Locate the install path for your Java runtime environment on the workstation. On our workstation we had installed the Sun Java 1.4.0 runtime and it was located in c:\Program Files\java\j2re1.4.0.

Change to <java_install_path>/lib/security directory and edit java.security.

Locate the lines that set the security providers and set the security.provider.2, 3, 4 lines as shown in Example A-6. We commented out existing lines.

Example: A-6 Registering security providers in java.security

```
#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.jsse.JSSEProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
security.provider.5=sun.security.jgss.SunProvider

#security.provider.2=com.sun.net.ssl.internal.ssl.Provider
#security.provider.3=com.sun.rsajca.Provider
#security.provider.4=com.sun.crypto.provider.SunJCE
```

Save these changes, and then open a new command prompt window.

If you want to work with a .jks file that already exists on the z/OS system, you will need to ftp it down to your workstation in binary and place it a suitable directory.

You are now ready to start iKeyman. Open a new command prompt window, and then invoke iKeyman with the following command:

```
java com.ibm.gsk.ikeyman.Ikeyman
```

If you get an error message during startup saying that you have to register providers, it means you still have to customize the java.security file properly or that your Java runtime is using a different java.security file from the one you customized. This could happen if you have more than one Java runtime installed on your machine.

If everything is configured properly, you should see the graphical interface as shown in Figure A-12 with all options active and not greyed out. If you want to work with an existing key store, click the open folder icon and browse to the directory where the .jks file is. Otherwise, create a new database.

When you create a new database, it is automatically loaded with the common signer certificates of the various certification agencies. Refer to the redbook *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133 for detailed information about the various tasks you need to perform when using iKeyman to create certificates rather than keytool.

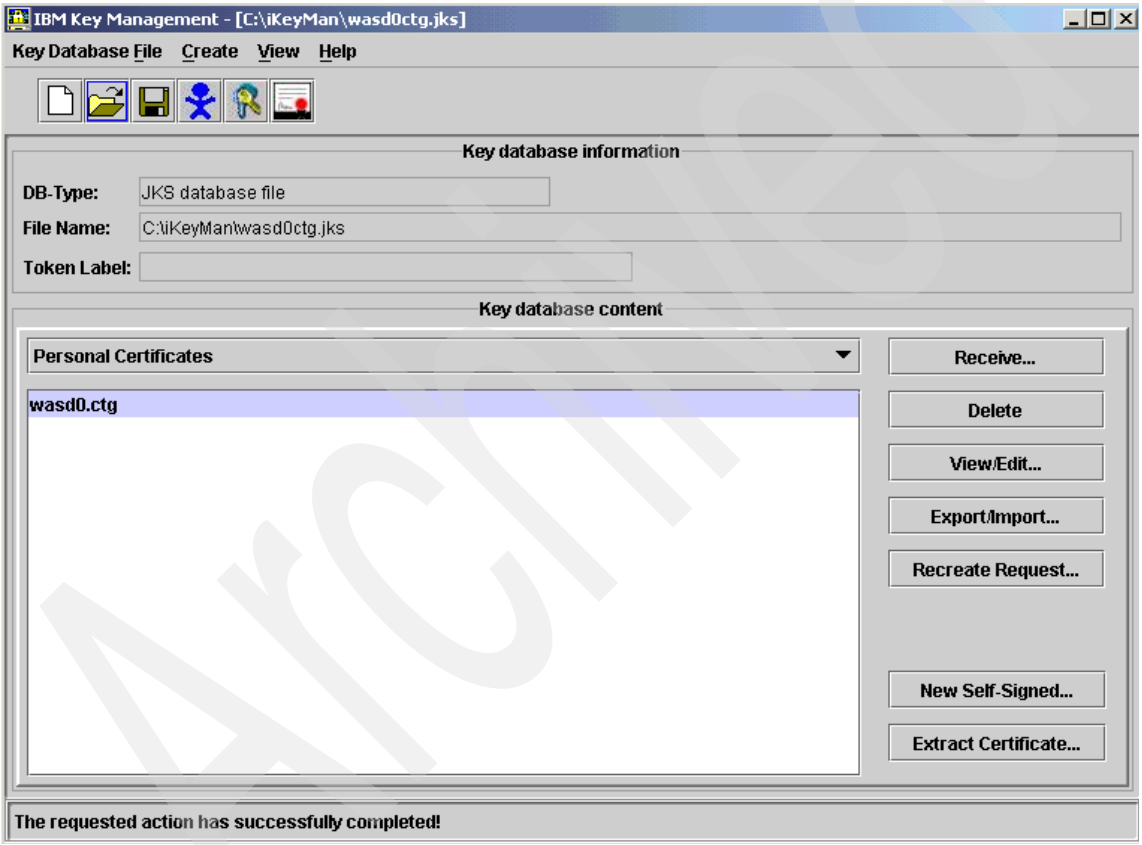


Figure A-12 iKeyman graphical interface

Importing WebSphere's certificate in CICS Transaction Gateway KDB

If you want to use client authentication in the CICS Transaction Gateway daemon, you will have to use keytool or iKeyman to create a client certificate for the WebSphere servant. You will then need to use gskkyman again to import

WebSphere's certificate into the CICS Transaction Gateway daemon's key database.

To import the certificate, start gskkyman as before and open the CICS Transaction Gateway daemon's key database.

```
SEC2@SC59:/u/sec2> cd /usr/lpp/ctg/bin/kdb
SEC2@SC59:/usr/lpp/ctg/bin/kdb> gskkyman

Database Menu

   1 - Create new database
   2 - Open database
   3 - Change database password
   4 - Change database record length
   5 - Delete database

   0 - Exit program

Enter option number: 2
Enter key database name (press ENTER to return to menu): ctgserver.kdb
Enter database password (press ENTER to return to menu): ctgserver
```

Figure A-13 CICS Transaction Gateway daemon's key database

You now get the main menu. Choose option 7, `Import a certificate`, and specify the full path to the exported certificate.

```
Key Management Menu

      Database: /usr/lpp/ctg/bin/kdb/ctgserver.kdb

  1 - Manage keys and certificates
  2 - Manage certificates
  3 - Manage certificate requests
  4 - Create new certificate request
  5 - Receive certificate issued for your request
  6 - Create a self-signed certificate
  7 - Import a certificate
  8 - Import a certificate and a private key
  9 - Show the default key
 10 - Store database password
 11 - Show database record length

  0 - Exit program

Enter option number (press ENTER to return to previous menu): 7
Enter import file name (press ENTER to return to menu):
/WebSphere/BS00/appserver/etc/wasd0ctg.arm
Enter label (press ENTER to return to menu): wasd0.ctg

Certificate imported.

Press ENTER to continue.
 ===>
```

Figure A-14 Main menu

Now that the JSSE key store and CICS Transaction Gateway daemon key database have been set up with the self-signed certificates, you should customize the connection factory using the WebSphere administrative console.

Creating a connection factory to support SSL

You define a connection factory together with Custom Properties, which include ConnectionURL, Portnumber, Servername, KeyRingClass, and KeyRingPass, as follows:

1. Open the WebSphere administrative console and navigate to **Resources**.
2. Click the CICS ECI resource adaptor you created earlier.
3. Click **J2C Connection Factories** → **New**.

4. Give the connection factory a name and a JNDI name. (For example, for SWIPECICS testing we used cicsremoteJCASSL for the connection factory name and /eis/cicsremoteJCASSL for the JNDI name.)

   Note: You will need to use WebSphere Studio Application Developer or the AAT to update or add a resource reference that maps to the JNDI name.

5. Click **OK**.
6. Click the connection factory you created, and then scroll down and click **Custom Properties**.

   The Servername property should have the VTAM APPLID of the CICS region you want to connect to.

   For a remote mode connection that uses SSL, the ConnectionURL property must be ssl, or https. We used ssl.
7. Set the PortNumber for the port that the CICS Transaction Gateway daemon is listening on for the System SSL protocol. This is coded in the CTG.INI file of the CICS Transaction Gateway daemon as shown in Example A-7.

Example: A-7 Protocol settings for System SSL in CTG.INI

```
protocol@systemssl_ssl.handler=com.ibm.ctg.server.GskSslHandler
protocol@systemssl_ssl.parameters=port=8050;sotimeout=1000;\
connecttimeout=2000;idletimeout=600000;pingfrequency=60000;\
keyring=/your_dir/your_keydatabase.kdb;keyringpw=default;clientauth=on;
```

8. Click **KeyRingClass** and enter the full path to the JSSE key store you created earlier. We set this to /WebSphere/BS00/appserver/etc/wasd0ctg.jks.
9. Click **KeyRingPass** and enter the password for the key store (-storepass option when you created the key store.)
10. Click **OK** and save the configuration.

Now you are ready to test the SSL connection.

Certificate generation hints for network deployment cells

This section describes certificate generation hints for network deployment cells.

Certificate creation

The setup process does not support the creation of more than one server certificate, which means from the private key ownership point of view it is not

possible to use different user IDs for different servers (DMN, DPLMGR, NA, or AS). In this case, the RACF jobs (BBODBRAK&BBOCR2FD) have to be modified and rerun, with caution of course.

Certificate expiration

Both WebSphere CA and the Server certificate will be created with a validity of one year. Because RACDCERT does not issue a warning before expiration, is seems to be more suitable to specify a expiration date with 10 or 20 years from now.

```
Examle: RACDCERT CERTAUTH-
          gencert -
          subjectsdn(ou('ibm itso') o('ibm') c('us')) -
          notafter(date(2020-12-31)) -
          keyusage(certsign) -
          withlabel('itso ca')
```

Java security provider

Java directory $JAVAHOME/lib/security contains the java.security file. In that file, the security provider is specified:

```
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

If someone is using the Java keytool HWKEYTOOL, it is necessary to put the IBM Provider (which contains the JCA4758 class) in front of the Sun provider entry. If using WebSphere Application Server V.5 with Java 2 security concurrently (Java 2 security is switched on with global security at least), deployment manager will abend with A03, even if both global and Java security are switched off again. The Sun provider has to be the only one or the first in the list of providers, as shown:

```
   Symptoms :
      ---- Begin backtrace for nested exception
       java.lang.ExceptionInInitializerError: java.lang.ClassCastException
          at sun.security.pkcs.PKCS7.getCertificate(PKCS7.java:587)
          at
sun.security.pkcs.SignerInfo.getCertificate(SignerInfo.java:216)
          at sun.security.pkcs.SignerInfo.verify(SignerInfo.java:342)
          . . . .
```

Troubleshooting ICSF Pass Phrase initialization problems

Several problems can arise during Pass Phrase Initialization. In this section we describe these issues and how to correct them.

- The error message `SSM Not Enabled` appears in the upper-right corner of the Pass Phrase Initialization panel. If this occurs, make sure that SSM(YES) is specified in the initialization options data set, and that the box for Special Secure Mode was selected on the Crypto tab of the Image or Reset profile. If they are, stop or cancel the ICSF proc and start it again.

  The problem is related to first-time ICSF startup and should not occur again on that partition.

- The messages `Register not empty` or `CKDS already initialized` appear in the upper-right corner of the Pass Phrase Initialization panel. This problem is more complicated, and is usually the result of skipping steps in the installation instructions, or mistyping something along the way. This occurs when Pass Phrase Initialization is started, but does not complete successfully for some reason. When this occurs the master key registers and CKDS/PKDS are left in a partly loaded state, which will then cause subsequent attempts at Pass Phrase Initialization to fail.

The solution is to fix the problem that caused Pass Phrase Initialization to fail in the first place, reset the master key registers, delete and reallocate the CKDS and PKDS, and try again. The steps are as follows:

1. Identify and resolve the problem that caused Pass Phrase Initialization to fail. Console log messages will give clues as to what the problem is. One reason is forgetting to include CSFDAUTH as a value in the AUTHPGM list and AUTHTSF list (See “Customize SYS1.PARMLIB” in the *ICSF System Programmer’s Guide*). Another reason is failing to enclose the CKDS name in quotes on the Pass Phrase Initialization panel. Without quotes, your TSO prefix will be included in the data set and the data set will not be found.
2. Reset the master key registers. Two sets of instructions are given below, because the ICSF menu options were rearranged and updated at about z/OS 1.2. Examine ICSF main menu option 7 to determine which instructions go with your ICSF panels.

   If your ICSF main menu option 7 is User Control Functions, you have the earlier menu. Follow these instructions:

   a. Disable PKA Services. From the ICSF main menu, enter option 7, User Control Functions. At the User Control Functions menu, enter option 4, Disable PKA Callable Services.

b. Go to Clear Master Key Entry (from the ICSF main menu, enter options 1, 1, 1).

c. Select your first KSU (0).

d. At the Clear Master Key Entry Panel, for Key Type, select **KMMK**. For Part, select **RESET**. Press Enter. Ignore any messages about `Restart Option Invalid or Ignored`.

e. At the Clear Master Key Entry Panel, for Key Type, select **SMK**. For Part, select **RESET**. Press Enter. Ignore any messages about `Restart Option Invalid or Ignored`.

f. At the Clear Master Key Entry Panel, for Key Type, select **NMK**. For Part, select RESET. Press Enter. Ignore any messages about `Restart Option Invalid or Ignored`.

g. Select your second KSU (1), if you have two, and repeat steps d, e, and f above.

If your ICSF main menu option 7 is TKE, you have the more recent menu. Follow these instructions:

a. Disable PKA Services. From the ICSF main menu, enter option 4, Administrative Control Functions. At the Administrative Control Functions panel, enter a D beside PKA Callable Services and press Enter.

b. From the ICSF main menu, enter option 1, COPROCESSOR MGMT.

c. On the ICSF Coprocessor Management page, type an `E` next to your first coprocessor (C0) and press Enter.

d. At the Clear Master Key Entry Panel, for Key Type, select **KMMK**. For Part, select **RESET**. Press Enter. Ignore any messages about `Restart Option Invalid or Ignored`.

e. At the Clear Master Key Entry Panel, for Key Type, select **SMK**. For Part, select **RESET**. Press Enter. Ignore any messages about `Restart Option Invalid or Ignored`.

f. At the Clear Master Key Entry Panel, for Key Type, select **NMK**. For Part, select **RESET**. Press Enter. Ignore any messages about `Restart Option Invalid or Ignored`.

g. Select your second coprocessor (C1), if you have two, and repeat steps d, e, and f above.

3. Stop ICSF.

4. Using IDCAMS, delete your CKDS and PKDS, and then allocate them again. Instructions are available in the *ICSF System Programmer's Guide*. Be sure to include both PURGE and ERASE on the IDCAMS DELETE command.

5. Start ICSF

   Skip back to the beginning of this section, initialize the CKDS and PKDS and load your master key, and try again.

LDAP activity logging

z/OS V1R4 introduces the ability to do activity logging within the z/OS LDAP server. The log file can contain information about operations handled by the server, messages generated by the server, and summary statistics. This information can be stored in either an MVS data set or a file in the HFS.

The log file location is determined by the value set in the LDAP configuration file. To ensure accuracy, enter only fully qualified file names in the logfile variable in the configuration file.

The following is an example of a typical operation log record followed by the ending log record:

```
|   Wed May 22 11:53:52 2002 Search: connid = 4, base = cn=monitor, filter
= (objectclass=*)
 |   Wed May 22 11:53:52 2002 End Search: connid = 4, base = cn=monitor,
filter = (objectclass=*), rc = 0
```

Summary records are all created during activity logging. These records are created on an hourly basis as long as log records are being collected, or when a modify command is processed that affects the log function. The summary log records contain information about the operations that the server has processed. The following is an example of the summary log records.

```
Wed May 22 11:45:25 2002 total operations started = 6
Wed May 22 11:45:25 2002 total operations completed = 6
Wed May 22 11:45:25 2002 total search entries sent = 2
Wed May 22 11:45:25 2002 total bytes sent = 897
Wed May 22 11:45:25 2002 total connections processed = 2
Wed May 22 11:45:25 2002 current connections = 2
Wed May 22 11:45:25 2002 connection high water mark = 2
```

For more information about LDAP activity logging, refer to *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923, available at:

```
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/glda2a21/CONENTS
?SHELF=&DT=20021029160905#1.9.6
```

Tracing System SSL

For detailed instructions about how to debug System SSL flow and handshakes, a good starting point might be Chapter 12 of *z/OS System Secure Sockets Layer Programming*, SC24-5901.

To enable SSSL tracing for WebSphere, set the following variables in was.env:

```
GSK_TRACE=0x3F
GSK_TRACE_FILE=/tmp/sssltrace.trc
```

Format by using:

/usr/lpp/gskssl/bin/gsktrace /tmp/sssltrace.trc >readablesssltrace.txt

B

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246086

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246086.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
SG246086.zip	SWIPE .ear files and other code fragments developed while writing this book

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material ZIP file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 721. Note that some of the documents referenced here might be available in softcopy only.

- *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- *IBM WebSphere Application Server V5.1 System Management and Configuration: WebSphere Handbook Series*, SG24-6195
- *IBM WebSphere V4.0 Advanced Edition Security*, SG24-6520
- *IBM WebSphere V5.1 Performance, Scalability, and High Availability: WebSphere Handbook Series*, SG24-6198
- *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401
- *Migrating Applications from WebSphere for z/OS V4 and V3.5 to V5*, SG24-7044
- *Migrating WebSphere Applications to z/OS*, SG24-6521
- *Revealed! Architecting e-business Access to CICS*, SG24-5466
- *Securing Web Access to CICS*, SG24-5756
- *Tivoli and WebSphere Application Server on z/OS,* SG24-7062
- *WebSphere for z/OS Connectivity Architectural Choices,* SG24-6365
- *WebSphere for z/OS V5 Connectivity Handbook,* SG24-7064
- *WebSphere Web Services Information Roadmap*, REDP-3854
- *z/OS WebSphere and J2EE Security Handbook*, SG24-6846

Other publications

These publications are also relevant as further information sources:

- *CICS Transaction Gateway Configuration Deploying the ECIDateTime J2EE Sample into WebSphere Application Server Version 5.0 for z/OS*, SC34-6306
- *CICS Transaction Gateway z/OS Administration*, SC34-6191
- *CICS Transaction Server for z/OS CICS Internet Guide*, SC34-6007
- *CICS Transaction Server for z/OS Java applications in CICS*, SC34-6000
- *CICS Transaction Server for z/OS: CICS RACF Security Guide*, SC34-6011
- *IBM Tivoli Access Manager Base Administration Guide, V5.1*, SC32-1360
- *IBM Tivoli Access Manager for e-business IBM WebSphere Application Server Integration Guide, V5*, SC32-1368
- *IBM Tivoli Access Manager for e-business Web Security Installation Guide Version 5.1*, SC32-1361
- *Implementing Enhanced Form Based Authentication with Servlet Filters in J2EE Applications under WebSphere Application Server v5.0.x and v5.1 for z/OS*, TD101255
- *IMS Version 8: Installation Volume 2: System Definition and Tailoring*, GC27-1298
- *SOAP for CICS feature of CICS Transaction Server for z/OS User's Guide*, SC34-6315
- *WebSphere Application Server for z/OS Version 5 Installation and Customization*, GA22-7910
- *z/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521
- *z/OS HTTP Server Planning, Installing, and Using Guide*, SC34-4826
- *z/OS Integrated Security Services LDAP Server Administration and Use*, SC24-5923
- *z/OS MVS Programming: Resource Recovery*, SA22-7616
- *z/OS MVS Setting Up a Sysplex*, SA22-7625
- *z/OS System Secure Sockets Layer Programming*, SC24-5901
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS UNIX System Services Planning*, GA22-7800
- *z/OS: Security Server RACF Security Administrator's Guide*, SA22-7683

- *Java 2 Platform Enterprise Edition Specification, v1.2*, Sun Microsystems, Inc., 1999
- *Java 2 Platform Enterprise Edition Specification, v1.3*, Sun Microsystems, Inc., 2001
- *Java Servlet Specification, v2.2*, Sun Microsystems, Inc., 1999
- *Enterprise JavaBeans Specification, v1.1,* Sun Microsystems, Inc., 1999
- "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617
- "MIME Part One: Format of Internet Message Bodies," RFC 2045

Online resources

These Web sites and URLs are also relevant as further information sources:

- *WebSphere Application Server Information Center*

  `http://www.ibm.com/software/webservers/appserv/infocenter.html`
- Sun Microsystems, Inc. Java Technology Web site

  `http://java.sun.com/`
- Request For Comments for Internet specifications

  `http://www.faqs.org/rfcs/`
- WebSphere Application Server V 4.0.1 for z/OS and OS/390: WebSphere HTTP Plug-in for z/OS APAR PQ68250, Service Level W401500, available at:

  `http://www.ibm.com/software/webservers/appserv/zos_os390/support/`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

`ibm.com/redbooks`

Help from IBM

IBM Support and downloads

`ibm.com/support`

IBM Global Services

`ibm.com/services`

Index

B

C

D

E

F

G

H

I

J

K

L

M

R

S

T

U

V

W

X

Z

WebSphere Application Server for z/OS V5 and J2EE 1.3 Security Handbook

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

WebSphere Application Server for z/OS V5 and J2EE 1.3 Security Handbook

Integration of the z/OS J2EE server into a heterogeneous landscape

Interoperability and cross-platform security enablement

J2EE security concepts and their implementation

What do you think of when someone mentions z/OS security? Probably of something that is trustworthy, or even impenetrable. Perhaps you also think of something that is a little complex and challenging to administer.

What comes to mind when someone mentions Internet security? Perhaps you think of prominent Web sites that have been maliciously “hacked” or credit card numbers that have been stolen.

Using working examples of code and configuration files, in this IBM Redbook, we explain how you can run your Web-enabled applications with as high a level of security as other z/OS applications and subsystems, even if those applications were written or originally deployed on another platform, by using the Java 2 Platform Enterprise Edition (J2EE) programming model and IBM WebSphere Application Server for z/OS and OS/390.

This redbook will help architects, application programmers, WebSphere and security administrators, and application and network architects to understand and use these products.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6086-01 ISBN 0738490679