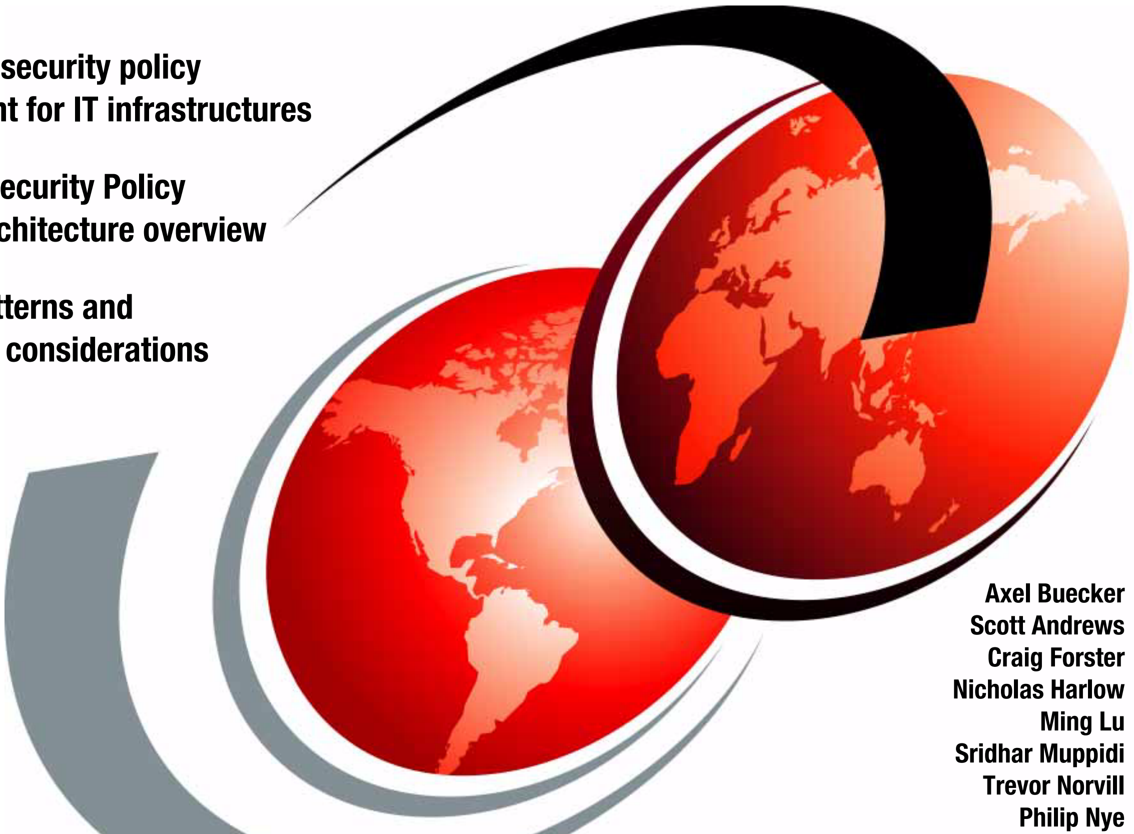
IBM® Tivoli®

IT Security Policy Management Usage Patterns Using IBM Tivoli Security Policy Manager

End-to-end security policy management for IT infrastructures

IBM Tivoli Security Policy Manager architecture overview

Solution patterns and deployment considerations



Axel Buecker
Scott Andrews
Craig Forster
Nicholas Harlow
Ming Lu
Sridhar Muppidi
Trevor Norvill
Philip Nye
Günter Waller
Eric T. White

Redbooks

ibm.com/redbooks

International Technical Support Organization

IT Security Policy Management Usage Patterns Using IBM Tivoli Security Policy Manager

October 2011

SG24-7880-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (October 2011)

This edition applies to Version 7.1 of IBM Tivoli Security Policy Manager.

Contents

Notices ix
Trademarks x

Preface xi
The team who wrote this book xii
Now you can become a published author, too! xv
Comments welcome xv
Stay connected to IBM Redbooks xvi

Part 1. Business context 1

Chapter 1. Business drivers and foundation for IT security policy management 3
1.1 Drivers that influence security 4
1.1.1 Business drivers that influence security 5
1.1.2 IT drivers that influence security 7
1.2 IBM Security Framework 10
1.2.1 Security Governance, Risk Management, and Compliance model . . 12
1.2.2 People and Identity domain 13
1.3 IBM Security Blueprint 14
1.4 SOA governance 17
1.4.1 SOA adoption: Impact on business and security 18
1.4.2 Relating SOA governance to other business drivers 18
1.5 Identity and access management governance 20
1.5.1 Critical data: Ensuring authorized access only when needed 20
1.5.2 Driving operational efficiency through automation 21
1.5.3 Enforcing consistent policy enforcement across the IT environment 21
1.6 Compliance management 22
1.6.1 Regulation and privacy concerns 22
1.6.2 Assessing compliance: The audit trail 23
1.6.3 Relating compliance management and governance 24
1.7 Data and information security 25
1.7.1 Risk of unauthorized access and data loss 25
1.7.2 Context-based information access 26
1.7.3 Data security in cloud and SOA environments 27
1.8 IT security policy management: A unifying solution 28
1.8.1 Addressing governance 29
1.8.2 Compliance and data security 30
1.8.3 Risk management and the cost containment 31

1.9 Introduction to IT security policy life cycle management 32
1.9.1 Policy authoring. 34
1.9.2 Transform 38
1.9.3 Enforcement 39
1.9.4 Monitor 39
1.10 Conclusion. 40

Chapter 2. Architecture patterns for externalizing security from applications and services 41
2.1 Intermediary approach 43
2.1.1 Customer example 44
2.1.2 Integrating policy at the intermediary level 44
2.2 Container level approach 47
2.2.1 Customer example 48
2.2.2 Integrating policy at the container level. 49
2.3 Database level approach. 51
2.3.1 Customer example 52
2.3.2 Integrating policy at the database level. 54
2.4 Application level approach 56
2.4.1 Customer example 56
2.4.2 Integrating policy at the application level 57
2.5 Conclusion. 59

Part 2. Implementing a policy life cycle management solution 61

Chapter 3. Tivoli Security Policy Manager overview and architecture . . . 63
3.1 Tivoli Security Policy Manager overview 64
3.1.1 Tivoli Security Policy Manager components 64
3.2 Tivoli Security Policy Manager architecture 69
3.2.1 Logical component architecture 72
3.2.2 Policy server architecture 75
3.2.3 Tivoli runtime security service architecture 78
3.2.4 Policy data model, repository, and exchange 83
3.2.5 Policy administration and classification. 84
3.2.6 Delegated administrative security 85
3.2.7 Auditing and reporting 87
3.3 Example deployment physical architecture 88
3.4 Conclusion. 91

Chapter 4. Integration with external systems 93
4.1 Identity management. 94
4.1.1 Integration with identity management 95
4.1.2 Integration with Tivoli Identity Manager. 96
4.1.3 Integration with other identity management systems 98

4.2 Access management 98
4.2.1 Integration with Tivoli Access Manager for e-business 100
4.2.2 Integration with other access management systems 103
4.3 Role management 103
4.4 User repositories 104
4.4.1 Integration with Tivoli Directory Server 104
4.4.2 Integration with Tivoli Directory Integrator. 105
4.4.3 Integration with other user repositories 109
4.5 Trust services 110
4.5.1 Integration with Tivoli Federated Identity Manager 110
4.6 Application repositories 111
4.6.1 Databases 111
4.6.2 User registries 112
4.6.3 Proprietary repositories 112
4.6.4 Java. 112
4.7 Classification management tools. 113
4.8 Compliance management 114
4.9 Rules engines 115
4.10 Conclusion. 116

Part 3. Usage patterns for IT security policy management 119

Chapter 5. Intermediary level integration 121
5.1 Concepts and benefits. 122
5.1.1 Scenario A: Established application environments 122
5.1.2 Scenario B: Services external to the enterprise 124
5.1.3 Scenario C: SOA message protection policies 125
5.1.4 Conclusion 127
5.2 Java Web Application Servers 127
5.2.1 Foundation for integration 128
5.2.2 Java Web Application Server integration and using the policy life cycle model 128
5.2.3 Conclusion 137
5.3 Web Application Firewalls 137
5.3.1 Foundation for integration 138
5.3.2 WebSphere DataPower SOA Appliance integration with Tivoli Security Policy Manager 138
5.3.3 IBM WebSphere DataPower integration and using the policy life cycle model 139
5.3.4 Conclusion 146
5.4 Enterprise Service Bus 147
5.4.1 Foundation for integration 148
5.4.2 WebSphere DataPower SOA Appliance integration with Tivoli Security

Policy Manager 149
5.4.3 Conclusion 156
5.5 Third-party intermediaries 156
5.6 Conclusion. 157

Chapter 6. Container level integration. 159
6.1 Concepts and benefits. 160
6.2 WebSphere Application Server 161
6.2.1 Foundation for integration 161
6.2.2 WebSphere integration using the policy life cycle model 164
6.2.3 Conclusion 169
6.3 Microsoft environment 169
6.3.1 Microsoft container integration 170
6.3.2 Integration with Tivoli Security Policy Manager. 171
6.3.3 Microsoft integration using the policy life cycle model 174
6.3.4 Conclusion 185
6.4 Conclusion. 185

Chapter 7. Database level integration 187
7.1 Concepts and benefits. 188
7.2 Database policy information point 189
7.2.1 Foundation for integration 190
7.2.2 Integration with Tivoli Security Policy Manager. 190
7.2.3 Database integration using the policy life cycle model 192
7.2.4 Conclusion 193
7.3 Database policy enforcement point 194
7.3.1 Foundation for integration 194
7.3.2 Integration with Tivoli Security Policy Manager. 195
7.3.3 Database integration using the policy life cycle model 195
7.3.4 Conclusion 202
7.4 Enterprise content management databases 202
7.4.1 Foundation for integration 203
7.4.2 Integration with Tivoli Security Policy Manager. 204
7.4.3 ECM integration using the policy life cycle model. 206
7.4.4 Conclusion 212

Chapter 8. Application level integration 213
8.1 Runtime security services interfaces. 214
8.1.1 Tivoli Security Policy Manager authorization API 214
8.1.2 JSP tag library. 223
8.1.3 Custom authorization solutions for external systems 227
8.1.4 Policy information point. 233
8.1.5 External rules 235
8.2 Policy management API 237

8.2.1 Plug-in structure 237
8.2.2 Data model 241
8.2.3 Plug-ins 243
8.3 Application integration 247
8.3.1 Integration with Java technology 247
8.3.2 Integration with WebSphere Portal 253
8.3.3 Integration with Microsoft technology 254
8.4 Conclusion. 265

Chapter 9. Deployment considerations. 267
9.1 Business considerations 268
9.1.1 Business use cases 268
9.2 Deployment considerations 270
9.2.1 Identifying stakeholders 271
9.2.2 Identifying policies, services, and data 272
9.2.3 Prioritizing services 273
9.2.4 Identifying operational requirements 273
9.3 Deployment architecture 274
9.3.1 Single data center deployment pattern 275
9.3.2 Remote office deployment pattern 276
9.3.3 Other considerations in determining deployment patterns 278
9.3.4 Operational considerations 279
9.4 Application integration considerations. 280
9.4.1 Integration patterns 280
9.4.2 Application policy design considerations. 283
9.4.3 Conclusion 288
9.5 Conclusion. 288

Related publications 291
IBM Redbooks 291
Other publications 291
Online resources 291
How to get Redbooks. 292
Help from IBM 292

Notices

Trademarks

alphaWorks®	IBM®	Redpaper™
DataPower®	ILOG®	Redbooks (logo) ®
DB2®	Lotus®	Tivoli®
Extreme Blue®	On Demand Community®	WebSphere®
FileNet®	Redbooks®	

Preface

In a growing number of organizations, policies are the key mechanism by which the capabilities and requirements of services are expressed and made available to other entities. The goals established and driven by the business need to be consistently implemented, managed and enforced by the service-oriented infrastructure; expressing these goals as policy and effectively managing this policy is fundamental to the success of any IT and application transformation.

First, a flexible policy management framework must be in place to achieve alignment with business goals and consistent security implementation. Second, common re-usable security services are foundational building blocks for SOA environments, providing the ability to secure data and applications. Consistent IT Security Services that can be used by different components of an SOA run time are required. Point solutions are not scalable, and cannot capture and express enterprise-wide policy to ensure consistency and compliance.

In this IBM® Redbooks® publication, we discuss an IBM Security policy management solution, which is composed of both policy management and enforcement using IT security services. We discuss how this standards-based unified policy management and enforcement solution can address authentication, identity propagation, and authorization requirements, and thereby help organizations demonstrate compliance, secure their services, and minimize the risk of data loss.

This book is a valuable resource for security officers, consultants, and architects who want to understand and implement a centralized security policy management and entitlement solution.

The team who wrote this book

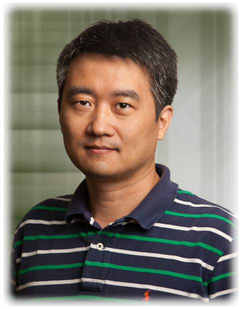
This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Axel Buecker is a Certified Consulting Software IT Specialist at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on areas of Software Security Architecture and Network Computing Technologies. He holds a degree in computer science from the University of Bremen, Germany. He has 25 years of experience in a variety of areas related to Workstation and Systems Management, Network Computing, and e-business Solutions. Before joining the ITSO in March 2000, Axel worked for IBM in Germany as a Senior IT Specialist in Software Security Architecture.

Scott Andrews is a Software Engineer in the Australian Development Lab at the Gold Coast, Australia. He is the Team Lead for a group that develops strategic integration solutions within the IBM Security portfolio for Independent Software Vendor (ISV) applications, such as Microsoft SharePoint. He holds a Bachelor of Information Technology degree from Bond University, Australia. Scott joined IBM in April 2002 and is a University Ambassador and active member of the IBM On Demand Community® helping to promote IBM to university graduates and within the community.

Craig Forster is an Advisory Software Engineer with IBM Tivoli® Software Security in Austin, Texas. He joined IBM in 2005 after graduating from the University of Queensland with a Bachelor of Engineering (Software) and Bachelor of Science (Computer Science) degrees. His areas of expertise include XACML, SOA security, J2EE security, and authorization best practices.

Nicholas Harlow is a Product Manager in the IBM Tivoli Security software organization. He manages a portfolio of host and content security products. He began his career in Tivoli Security as a software engineer in 2003. He holds a degree in Computer Science from Stanford University and an MBA from INSEAD. Nicholas has over ten years of experience building high-quality software in both large and small organizations and bridging the gap between the business and technical aspects of organizational management. Before joining IBM Tivoli Security in 2003, he was an IBM Extreme Blue® intern and a university student, during which time he was involved in several early-stage startup ventures.



Ming Lu, Ph.D., is a Senior Managing Consultant in the IBM Software Services for Tivoli (ISST) security practice team. He is in the enterprise security architect role and the team lead of the US East team. Ming has over 15 years of experiences in the field of information security, software engineering, and system management. He has helped many customers successfully design and deploy IBM security solutions. He has also been involved in security solution design for IBM WebSphere®, IBM Lotus®, and SOA projects. Before joining ISST in 2006, Ming worked in the IBM Tivoli Austin lab for seven years as a Senior Security Architect. He holds a Ph.D. degree in Computer Science from Tsinghua University, Beijing, China.

Sridhar Muppidi, Ph.D., is the Chief Architect for IBM Security Solutions. As a Senior Technical Staff Member, he drives security architecture and design activities across IBM products, platforms, and solutions. In his career at IBM, his responsibilities have included SOA and Cloud Security Architecture, Identity and Access Management, Policy Management, and solutions that are based on these areas. He was the lead architect for IBM Security Policy Management solutions, and led the product architecture and design activities, as well as collaboration across IBM for a unified security policy management solution. His current responsibilities also include providing secure cross brand security solutions to enterprises, leading work groups in security and privacy, and representing IBM in open standards activities.

Trevor Norvill is a senior accredited IT Specialist working for IBM software group. He is based in the Gold Coast, Australia. He has worked in Tivoli Security technical pre-sales, post-sales lab services, and product development roles. During his nine years at IBM, he has gained extensive experience helping IBM clients design, deploy, and customize Tivoli Security solutions. His certifications include advanced deployment professional in IBM Tivoli Security management solutions and IT Infrastructure Library (ITIL) V3. He holds a degree in Computer Systems Engineering with honors from the University of Queensland.

Philip Nye is an IT Specialist working on the Tivoli Security Software Advanced Technology (SWAT) team, based out of the Australian Development Lab on the Gold Coast. Philip works with Tivoli Security products, including Tivoli Security Policy Manager, Tivoli Identity Manager, Tivoli Access Manager for e-business, Tivoli Federated Identity Manager, and Tivoli Security Information and Event Manager in pre-sales, post-sales engagements and delivering enhanced product enablement. His holds a Bachelor of Engineering in Software Engineering degree and a Bachelor of Business Management degree from the University of Queensland.

Günter Waller is a Certified IT Specialist with IBM Germany. He has 12 years of experience in IT security plus additional 21 years in IT and networking. He holds a degree in Mathematics from Johann Wolfgang Goethe-Universität Frankfurt am Main, Germany. His areas of expertise include Identity and Access Management, Federated Identity Management, and Security Policy Management. He has co-authored two IBM Redbooks publications in the past (1996 and 2000). In 2008, he co-authored a publication about SOA Policy Management in Germany.

Eric T. White is a Sr. Managing Consultant in the US GBS Security and Privacy Practice with over 25 years of experience in the IT field, including Data Security & Privacy, SOA Security Architecture, Process Reengineering, and Systems Management. Eric is a globally recognized subject matter expert, thought leader, and trusted advisor for highly complex security and privacy initiatives among industry leaders within the banking, financial, energy & utilities, healthcare, insurance, retail, distribution, and communications sectors. Most recently, Eric has lead the security and privacy activities for multiple highly complex SOA Application Infrastructure modernization programs in the retail and energy/utilities sectors.

Thanks to the following people for their contributions to this project:

Wade Wallace
International Technical Support Organization, Austin Center

Richard Cohen, Jeffrey DeMent, Deepak Gangadhar, Lachlan Hillman, Bill Hines, Albee Jhoney, Carsten Lorenz, Timothy Moore, Martin Schmidt, Eric Schulz, Ravi Srinivasan, Drew Walters, Nicholas Whelan, Eric Wood, Krishna Yellepeddy
IBM

Now you can become a published author, too!

Comments welcome

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`

- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

Part 1

Business context

In this part, we take a closer look at the current business challenges of managing the increasing complexity for IT security policy life cycle management. We introduce product agnostic architecture patterns for externalizing security that can help to build an IT security life cycle management model.

1

Business drivers and foundation for IT security policy management

As IT infrastructure and business processes have grown in complexity with the continuing evolution of business practices and technology, so has the cost and risk of managing the security of sensitive data and governing organizations that rely on these processes and technologies.

In the first part of this chapter, we explore some of the concerns that characterize security requirements of, and threats to, business and information technology (IT) systems. We identify a number of the business drivers that illustrate these concerns, including managing risk and cost, and compliance to business policies and external regulations, showing how they can be translated into frameworks to enable enterprise security.

To help you with your security challenges, IBM has created a bridge to address the communication gap between the business and the technical perspectives of security to enable simplification of thought and process. The IBM Security Framework can help you translate the business view, and the IBM Security Blueprint describes the technology landscape view. In concert, they can help bring together the experiences we gained from working with many clients to build a comprehensive solution view.

In the second part of this chapter, we look closer at comprehensive IT security policy management and how it has emerged as a disciplined approach that allows organizations to satisfy four critical business requirements:

- SOA governance
- Identity and access management governance
- Compliance management
- Data and information security

We provide an overview of each of these drivers and illustrate how IT security policy management can help address the business challenges they pose. This discussion of IT security policy management will introduce the concept of *policy life cycle management*, a dynamic process of policy authoring, enforcing, transforming, and monitoring that results in more transparent, efficient, and cost-effective governance of the organization.

1.1 Drivers that influence security

Most of today's projects are driven by both business and IT drivers, although we can probably agree that business drivers are almost always the initiating factor. Let us take a closer look at these influencing factors:

- Business drivers: Business drivers measure value, risk, and economic costs that influence their approach to IT security. Value drivers determine the worth of assets of the system to the business and of the business itself. Risk drivers involve compliance, corporate structure, corporate image, and the risk tolerance of the company. Economic drivers determine productivity impact, competitive advantage, and system cost.
- IT drivers: IT drivers represent operational constraints in the general IT environment. For example, the complexity of a system, including its environment, that is exposed to internal and external threats presents risks that the organization must address.

Business drivers also represent issues and consequences of significance to the stakeholders of the managed business system. This set of drivers might vary from industry to industry, from organization to organization in the same industry, and even between different business applications in an organization.

IT drivers represent technical considerations that affect the trustworthiness of the IT environment and likely the managed business systems as a whole. IT drivers are universal and must be considered within the context of the business drivers in all efforts. The combination of business and IT drivers represents the key initiatives for security management.

1.1.1 Business drivers that influence security

Business drivers represent a relationship between the IT organization and the rest of the business. They refer to business values that must be supported by the IT security infrastructure.

Correct and reliable operation

Correct and reliable operation is the degree to which the business must be accurate and consistent in its operation. Correct operation means that the operations perform the proper response or function with no errors. Reliable means that the same result occurs all the time. Any IT system must consistently provide stakeholders with the expected results.

Security events and incidents might impact the correct and reliable operation of these business processes. It might also affect the underlying IT infrastructure or upstream and downstream business processes. The consequences of a defective service (incorrect or varying results over time) might be significant to the consumer of the service, and therefore to the provider of the service.

Service-level agreements

This driver applies to circumstances where security threats and threat agents can impact an organization's ability to conduct business. Service-level agreements (SLAs) incorporate acceptable conditions of operation within an organization. SLAs might vary from business system to business system or application to application. Availability of systems, data, and processes is a condition commonly referenced within SLAs.

IT asset value

From a business perspective, the IT asset value is directly related to the value of the business transactions that it supports. These assets might be tangible or intangible. For an e-retailer, these are tangible assets. For a financial services company, the asset might be the client information or other data used in transactions of the system.

Protection of the business asset value or brand image

This driver captures the firm's desire to protect its image. The loss of good will from a security incident or attack has a direct consequence to the business. Therefore, the security measures are likely to be proportional to the consequence. When the desire to avoid negative publicity increases upon encountering a security breach, the stipulation for this driver becomes stronger.

Legal and regulatory compliance

Legal and regulatory compliance refers to the externally imposed conditions on the transactions in the business system and the company, which includes the rules and policies imposed by regulatory and government agencies. Civil, criminal liability, or regulatory penalty from a security incident or attack have a negative impact on the business. Therefore, the amount of regulation and steps to ensure compliance should be factored in this driver, which includes privacy issues, the ability to prove the transaction initiator, and proving compliance.

An implemented log management system can tell who did what, where, and when. Log management, therefore, is an integral part of an IT security compliance management system. For the retention period of the logs, it is ensured that the necessary information is available and can be analyzed or interpreted to a level that can help management to better investigate security incidents or comply with external regulation or laws. Compliance is a key business driver today, and log management should be a part of every IT security compliance management solution. But it can also be implemented alone as an initial step towards a larger IT security compliance initiative. As already mentioned, many international standards and regulatory controls require logging to be enabled and implemented. Also, these logs must be analyzed periodically and stored for a specific period of time, depending on the particular standard or regulatory control.

Contractual obligation

Security measures for an IT system are likely to be proportional to the consequences encountered when the business encounters contractual liability from a security attack. Depending on the structure and terms of the contract, the consequence might lead to financial loss or liability. For example, when security incidents are encountered, the business might be unable to fulfill its contractual obligations of providing goods or services.

Financial loss and liability

Direct or indirect financial loss is a consequence to the business as a result of a security incident. Direct loss might include theft of an asset, theft of a service, or fraud. Indirect loss might include loss based on civil or criminal court ruling, loss of good will, or re-prioritized budget allocation. This driver identifies the fact that security measures for an IT system are likely to be in proportion to these consequences.

Critical infrastructure

This driver applies where security threats or threat agents can have a major impact on services or resources that are common to, or shared among, a community of businesses, the population at large, or both. Examples include telecommunications, electrical power, transportation systems, computing, and so on. The loss of critical infrastructure by its provider might have a ripple effect, causing secondary losses and driving security decisions for those affected. An important part of risk analysis is identifying critical infrastructure.

Safety and survival

This driver applies where security threats and threat agents can have a major impact on aspects of human life, government function, and socio-economic systems. Examples of processes to be considered for safety and survival impact include continuity of critical infrastructure, medical system, life support, or other high-impact or time-dependent process.

1.1.2 IT drivers that influence security

IT drivers make up the second group of key security initiatives. These are considered universal drivers that must be considered in every modern IT solution in a manner commensurate with the risks and consequences of a related failure or incident.

Internal threats and threat agents

Security-related failures and incidents are caused by threats or threat agents found within the physical and logical boundaries of the organization or enterprise that operates and controls the IT system. These threats and threat agents might be associated with technology or people.

An example of an internal threat is a poorly designed system that does not have the appropriate controls. An example of an internal threat agent is a person who would use his ability to access the IT system or influence business or management processes to carry out a malicious activity.

External threats and threat agents

Security-related failures and incidents are caused by threats or threat agents found outside the physical and logical boundaries of the organization or enterprise that operates and controls the IT system. These threats and threat agents are also associated with technology or people. They seek to either penetrate the logical or physical boundary to become internal threats or threat agents, or to influence business or management processes from outside the logical or physical boundary.

Examples of external threats are single points of failure for one or more business or management processes that are outside the enterprise boundary, such as a power system grid or a network connection, or a computer virus or worm that penetrates the physical or logical network boundary. An example of an external threat agent is a hacker, or someone who has gained the ability to act as an insider, using personal electronic credentials or identifying information.

IT service management commitments

This driver identifies the fact that failure to manage the operation of the IT system might result in security exposures to the business. This driver can be divided into two categories: IT service delivery and IT service support.

- Service delivery commitments

  The failure of the IT system to meet its metrics for managing itself can be viewed as a security exposure to both business or management processes.

  An example of security exposure for service delivery is when IT operations processes cannot respond to critical events in a timely manner. Another is when IT resilience processes cannot recover from a denial of service attack in a timely manner, resulting in a loss of capacity or response time for business processes.

- Service support commitments

  The failure of the business or IT management system to meet its service-level agreements can be viewed as a security exposure to business or management processes.

  An example of security exposure for service support is a situation in which the customer relationship processes do not add, modify, or remove users from access control lists in a timely manner.

IT environment complexity

The complexity of the IT environment might contribute to the security or insecurity of the IT system. The IT environment reflects the infrastructure on which the business system will be placed.

For example, any IT environment that is connected to the intranet or extranet is exposed to internal or external threats or threat agents and requires specific security responses. A stand-alone facility for our system represents the lowest complexity. A hosting facility with other systems and other firms represents a more complex environment. An environment with a larger number of systems, varied network access paths, or a complex architecture, is a complex IT environment.

Business environment complexity

Because most businesses rely on IT, most business environments are an interconnected set of businesses, each with its own complex IT environment, business processes, and IT management processes. This complexity might contribute to the security or insecurity of the IT system.

Audit and traceability

This driver identifies the need for the IT system to support an audit of information contained within the system, whether it is associated with management data or business data.

IT vulnerabilities: Configuration

Configuration vulnerabilities are potentially present in every IT system, providing an opening to a potential attack based on the system and how it is designed and set up.

IT vulnerabilities: Flaws

Software flaws potentially exist in every IT system. These flaws represent vulnerabilities that were not detected and are not evident in the design documents. As such, they are an unexpected deviation from what was designed. An example is a defect in an operating system or application that is discovered after implementation.

IT vulnerabilities: Exploits

The basic design of software in any IT system might be exploited by threats or threat agents as a part of an attack on the IT system, the business, or the management processes, which might include the use of a function within a system in a way to compromise the system or underlying data. Although certain people might define an exploit as both the flaw and the method, we treat them separately because an exploit might involve using normal functions as designed in an unusual manner to attack the system. The exploits can also be viewed as the openings or avenues that an attacker can use.

Now it is time for us to introduce the IBM Security Framework, which focuses on the *what*, not the *how*. It can help you translate your requirements into coarse-grained business solutions, not into specific IT components or IT services.

1.2 IBM Security Framework

Today's business leaders are expected to manage risk in their areas of responsibility in the same way that CFOs manage risks in their domains. Security risks and the potential impact on IT need to be communicated to executive peers in business terms. Additionally, they need to align IT security controls with their business processes, monitor and quantify IT risk in business terms, and dynamically drive business-level insight at the executive level. Finally, business leaders need to manage risk and orchestrate security operations in a way that enforces compliance and optimizes business results.

As an organization secures its business processes, a business-driven approach needs to become the guiding influence for ensuring that all the different security domains work together in a holistic and synergistic manner, in alignment with the overarching business objectives. Otherwise, the organization's risk stance becomes vulnerable due to misalignment of priorities between IT and the business strategy. Using a standards-based approach to map business drivers to IT security domains is often difficult and is often an afterthought.

IBM created a comprehensive IT security framework (Figure 1-1) that can help ensure that every necessary IT security domain is properly addressed when using a holistic approach to business-driven security.

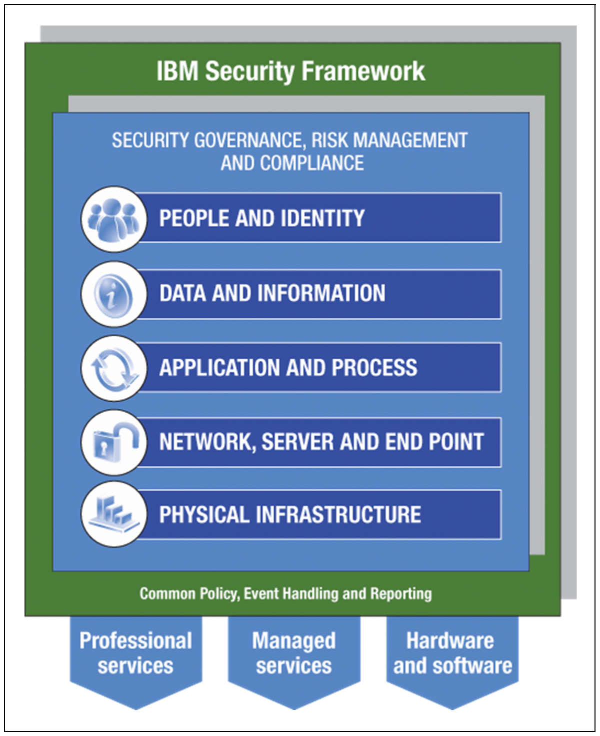


Figure 1-1 The IBM Security Framework

IBM provides the full breadth and depth of solutions and services that can enable organizations to take this business-driven, secure by design approach to security in alignment with the IBM Security Framework. Comprehensive professional services, managed services, and hardware and software offerings are available from IBM to support your efforts in addressing the different security domains covered by the IBM Security Framework.

1.2.1 Security Governance, Risk Management, and Compliance model

Every organization needs to define and communicate the principles and policies that guide the business strategy and business operation. In addition, every organization must evaluate its business and operational risks, and develop an enterprise security plan to serve as a benchmark for the execution and validation of the security management activities that are appropriate for their organization.

These principles and policies, the enterprise security plan, and the surrounding quality improvement process represent the enterprise Security Governance, Risk Management and Compliance model. Specifically, the requirements and the compliance criteria for the five security domains are:

- People and Identity

  This domain covers aspects about how to ensure that the correct people have access to the correct assets at the correct time.
- Data and Information

  This domain covers aspects about how to protect critical data in transit or at rest across the organization.
- Application and Process

  This domain covers aspects about how to ensure application and business services security.
- Network, Server, and Endpoint (IT infrastructure)

  This domain covers aspects about how to stay ahead of emerging threats across IT system components.
- Physical Infrastructure

  This domain covers aspects about how to use the capability for digital controls to secure events, on people or things, in the physical space.

In the following section, we take a closer look at the People and Identity domain. We focus on this domain because it is the driving factor for implementing a security policy management solution. If you want to learn more about the other IBM Security Framework domains refer to the IBM Redpaper™ publication *Introducing the IBM Security Framework and IBM Security Blueprint to Realize Business-Driven Security*, REDP-4528.

1.2.2 People and Identity domain

Organizations need to protect the assets and services that serve the business and support the business operation. One aspect of protection is provided by *access control*. The ability to provide effective access control services is based on the ability to manage people and identity as defined by the enterprise's security governance, risk, and compliance model.

The Security Governance, Risk Management, and Compliance model provides guidance about how identities are managed and how access control is conducted. Organizations register people and map them to identities. The relationships between people and organization are expressed in terms of role, rights, business policies, and rules. The ability to register people and describe their relationship with the enterprise is a key security enabler for the remaining security domains:

- Data and Information
- Applications and Process
- Network, Server, and Endpoint (IT infrastructure)
- Physical Infrastructure

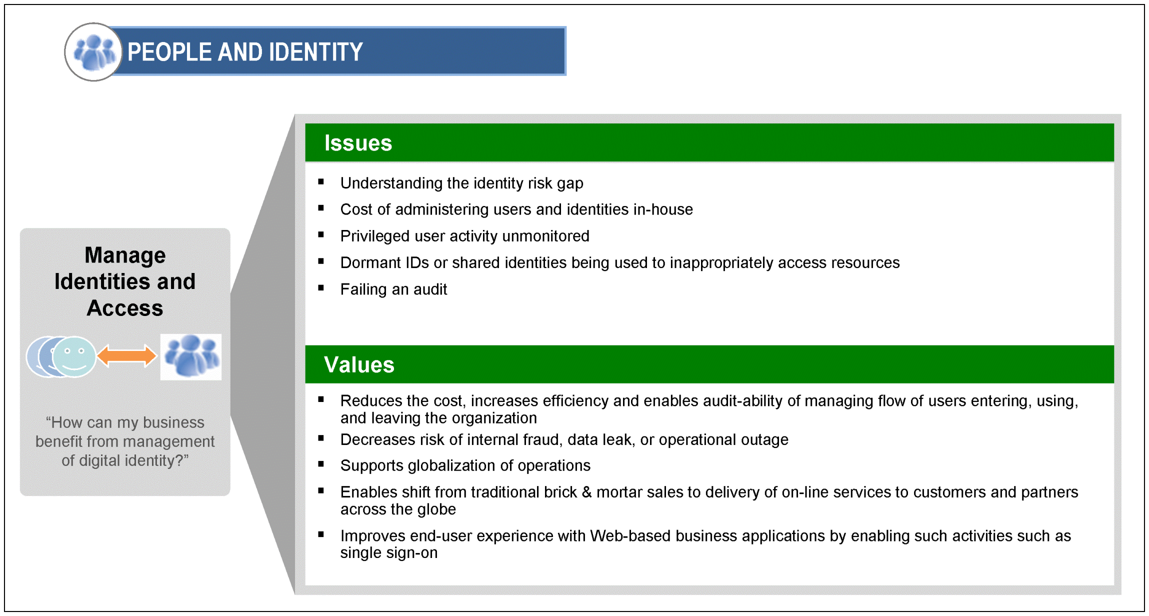
Operationally, people acting in authorized roles in an organization or as part of an extended relationship are granted access to infrastructure, data, information, and services. At the same time, people acting in unauthorized roles are denied access to infrastructure, data, information, and services if they are acting outside of the business policies and agreements.

Within an identity system, people can be issued a *credential*. A credential can take any of several forms, including a physical identity card or logical token or user identifier. The *trustworthiness* or *strength* of the credential is an important aspect of business policy and risk management. The ability to effectively manage the life cycle of identity, that is, the creation, removal, and role changes for dynamic populations of workforce, customer, or user communities, is extremely important. For example, the life cycle of identities and credentials can be influenced by business cycles, employment cycles, customer relationship, agreement, business, or calendar events, and so on.

Identity systems need to be integrated with appropriate sets of access controls. Identity systems need to manage user roles, rights, and privileges across the IT infrastructure that might contain multiple technology architectures, or multiple identity and access control systems will be required to ensure that users have access to the correct assets and services.

Compliance for identity and access is often externally motivated compliance. For example, legislated privacy and evidence recording is a significant driver for implementation of comprehensive user provisioning and identity-related record keeping.

Figure 1-2 shows a summary and additional aspects to be addressed within the People and Identity domain.



PEOPLE AND IDENTITY

Manage Identities and Access

"How can my business benefit from management of digital identity?"

Issues

- Understanding the identity risk gap
- Cost of administering users and identities in-house
- Privileged user activity unmonitored
- Dormant IDs or shared identities being used to inappropriately access resources
- Failing an audit

Values

- Reduces the cost, increases efficiency and enables audit-ability of managing flow of users entering, using, and leaving the organization
- Decreases risk of internal fraud, data leak, or operational outage
- Supports globalization of operations
- Enables shift from traditional brick & mortar sales to delivery of on-line services to customers and partners across the globe
- Improves end-user experience with Web-based business applications by enabling such activities such as single sign-on

Figure 1-2 People and Identity domain

After having addressed and mapped the IT security domain, People and Identity, into your business solutions, it is time to look at the component-oriented view of IT security in the IT Security Blueprint.

1.3 IBM Security Blueprint

The IBM Security Framework divides the area of business-oriented IT security into five domains. The next step is to break these down into further detail to work toward a common set of core security capabilities needed to help your organization securely achieve its business goals. These core security capabilities are called the *IBM Security Blueprint*.

The IBM Security Blueprint uses a product-agnostic and solution-agnostic approach to categorize and define security capabilities and services that are required to answer the business concerns in the IBM Security Framework.

The IBM Security Blueprint was created after researching many customer-related scenarios, focusing on how to build IT solutions. The intention of the blueprint is to support and assist in designing and deploying security solutions in your organization.

Building a specific solution requires a specific architecture, design, and implementation. The IBM Security Blueprint can help you evaluate these areas, but does not replace them. Using the IBM Security Blueprint in this way can provide a solid approach to considering the security capabilities in a particular architecture or solution.

IBM has chosen to use a high-level, service-oriented perspective for the blueprint, based on the IBM service-oriented architecture (SOA) approach. Services use and refine other services (for example, policy and access control components affect almost every other infrastructure component.) To better position and understand the IBM Security Blueprint, see Figure 1-3.

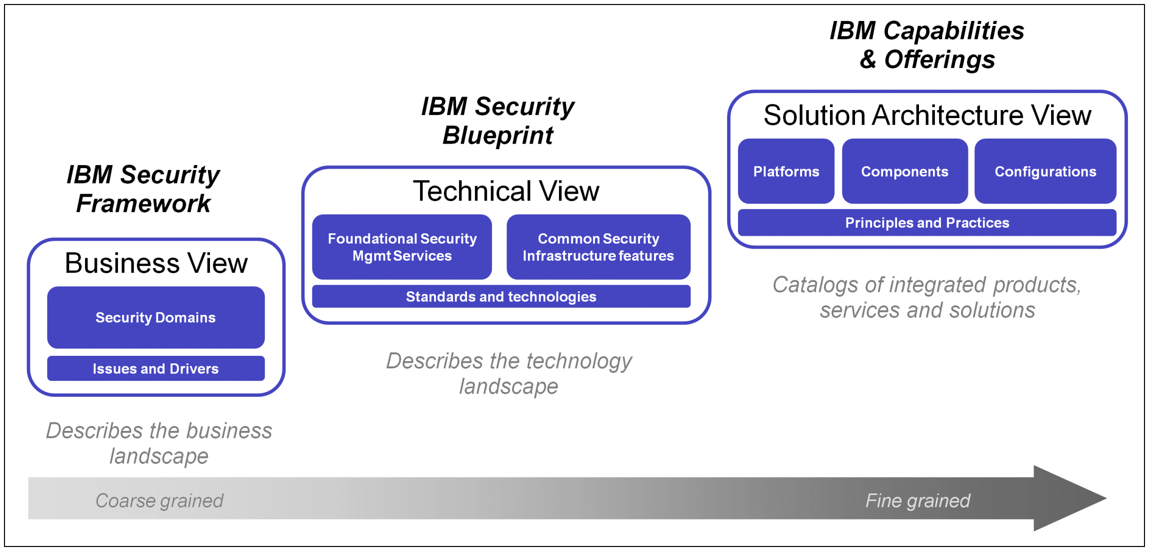

IBM Capabilities & Offerings
Solution Architecture View
Platforms
Components
Configurations
Principles and Practices
Catalogs of integrated products, services and solutions
IBM Security Blueprint
Technical View
Foundational Security Mgmt Services
Common Security Infrastructure features
Standards and technologies
Describes the technology landscape
IBM Security Framework
Business View
Security Domains
Issues and Drivers
Describes the business landscape
Coarse grained
Fine grained

Figure 1-3 IBM Security Blueprint positioning

The left portion of Figure 1-3 represents the IBM Security Framework, which describes and defines the security domains from a business perspective.

The middle portion in Figure 1-3 represents the IBM Security Blueprint, which describes the IT security management and IT security infrastructure capabilities needed in an organization. As discussed earlier, the IBM Security Blueprint describes these capabilities in product and vendor-neutral terms.

The right portion of Figure 1-3 on page 15 represents the solution architecture views, which describe specific deployment guidance particular to a given IT environment. Solution architecture views provide details about specific products, solutions, and their interactions.

Figure 1-4 highlights the components and subcomponents of the IBM Security Blueprint that have to be examined for every solution in the People and Identity security domain. Besides the Foundational Security Management, the IBM Security Blueprint enables you to determine the Security Services and Infrastructure components by reviewing the component catalogs for these Foundational Security Management services. Each of these components can then be assessed by determining whether each particular infrastructure component is required to make a Foundational Security Management service functional so that it can address the issues or provide a prospected value associated with the particular business security domain, in this case, People and Identity.

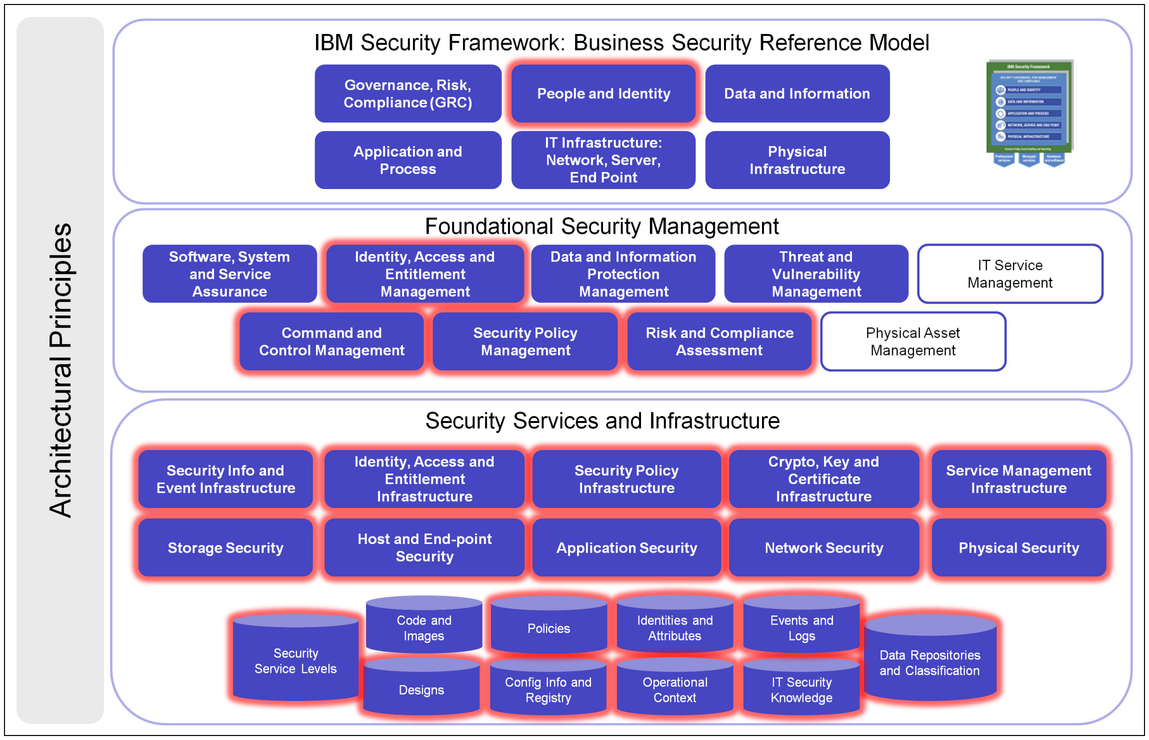

Architectural Principles
IBM Security Framework: Business Security Reference Model
Governance, Risk, Compliance (GRC)
People and Identity
Data and Information
Application and Process
IT Infrastructure: Network, Server, End Point
Physical Infrastructure
Foundational Security Management
Software, System and Service Assurance
Identity, Access and Entitlement Management
Data and Information Protection Management
Threat and Vulnerability Management
IT Service Management
Command and Control Management
Security Policy Management
Risk and Compliance Assessment
Physical Asset Management
Security Services and Infrastructure
Security Info and Event Infrastructure
Identity, Access and Entitlement Infrastructure
Security Policy Infrastructure
Crypto, Key and Certificate Infrastructure
Service Management Infrastructure
Storage Security
Host and End-point Security
Application Security
Network Security
Physical Security
Security Service Levels
Code and Images
Designs
Policies
Config Info and Registry
Identities and Attributes
Operational Context
Events and Logs
IT Security Knowledge
Data Repositories and Classification

Figure 1-4 IBM Security Blueprint components for the People and Identity solution pattern

We can see in Figure 1-4 on page 16 that almost all infrastructure components may be required for a People and Identity security solutions apart from Code and Images. The reason why those components are not included is that they are mostly covered by other domains of the IBM Security Framework.

If you want to learn more about the Foundational Security Management as well as the Security Services and Infrastructure sub-components, refer to *Introducing the IBM Security Framework and IBM Security Blueprint to Realize Business-Driven Security*, REDP-4528.

In the next section, we examine the comprehensive IT security policy management and how it has emerged as a disciplined approach that allows organizations to satisfy four critical business requirements:

- SOA governance
- Identity and access management governance
- Compliance management
- Data and information security

1.4 SOA governance

Service-oriented architecture (SOA) is a conceptual framework that casts repeatable business tasks as *services*, which can be composed in different ways to implement complex business processes as a series of linked services. Sets of related services can then be organized into *composite applications* that support specific business processes. These applications and services can be exposed across organizations to enable internal collaboration within the organization as well as external collaboration with partners, customers, and other stakeholders.

Starting in 2006, SOA emerged in response to the realization among senior managers that continued innovation depends on effective integration between business and technology in the organization. The flexible SOA approach to this integration allows organizations to align business and IT objectives, using existing IT investments while continuing to add new technology, allowing for adaptation and minimal disruption of existing systems.

For more information about this topic, refer to Appendix A, “Introduction to service-oriented architecture”, in *Understanding SOA Security Design and Implementation*, SG24-7310.

In the remainder of this section, we explore the impact of SOA adoption on business and security and the relationship between SOA governance and the other business drivers.

1.4.1 SOA adoption: Impact on business and security

Organizations began to adopt SOA in their existing IT infrastructures as a result of the proliferation of independent business applications, each of which had its own proprietary way to connect to other applications. As the number of these services grows, connecting each new service to the others became a more labor-intensive and inefficient task. This tight-coupling of services reduced the flexibility of the organization to react to changing market conditions.

SOA emphasizes a loose-coupling between services, which increased the flexibility that enterprise architects would have in reconfiguring the business processes implemented through their IT investments, as well as promoting reuse of this basic business logic. If organizations retain proprietary, hardcoded security mechanisms, it decreases the flexibility and component reusability that are the primary advantages of SOA. To adopt SOA, while maintaining a manageable business security system, organizations encountered new security challenges.

These challenges include:

- Selecting and enforcing appropriate security constraints across organizational boundaries.
- Determining the relevant IT security policy for a composite application that might have completely divergent policies governing its constituent services.
- Enforcing security, performance, and level of service requirements to complete transactions across organizations in real time, which can be impeded by a security process that is too heavy-weight.

1.4.2 Relating SOA governance to other business drivers

Although the four business drivers are distinct in their definition, they are interrelated in a way that can be complex, each one affecting the others in different ways.

For example, governance of identity and access management in a SOA environment requires specific technical considerations. In an environment with tightly-coupled services, each service may handle the identity of its clients independently and differently from the other services. Decoupling identity and access control from services is critical to handling identity consistently for all services, reducing complexity while maintaining the flexibility of loosely-coupled services.

After identity and access management are handled consistently across services, it becomes easier to satisfy the next critical requirement: providing secure access to service transactions to both internal and external users. Providing secure access entails managing the authorization entitlements of the system's users to ensure that only authorized entities gain access to critical data and operations.

Furthermore, consistent handling of security across the SOA environment allows business and IT managers to define a consistent set of rules for how messages are passed between services, specifying, for example, the level of encryption or inclusion of digital signatures in service requests to protect the confidentiality or integrity of messages flowing through the system.

Protecting the data in the system includes data at rest in storage repositories. Customer data, such as financial information, healthcare details, or other privacy-sensitive information, must be protected from improper disclosure and corruption, either accidental or malicious. Failure to do so can cause embarrassment to the organization, loss of customers and revenue, and in some cases litigation for negligence, especially when the data is subject to governmental regulations such as HIPAA[1], Sarbanes-Oxley[2], and FISMA[3], to name a few.

Effective governance and compliance management is critical for businesses because they enable senior managers to manage business risk by protecting the data and information vital to the organization's operations. In addition, robust SOA governance increases both the security and transparency of a system, reducing the cost to comply with regulations, and providing auditable transaction records.

[1] HIPAA: The Health Insurance and Portability and Accountability Act was passed by the United States Congress in 1996. For more information, refer to the following address: `http://www.hhs.gov/ocr/privacy/`.

[2] The Sarbanes-Oxley Act was passed in 2002 in response to widespread financial misdeeds in the corporate community. For more information, refer to the following address: `http://www.soxlaw.com/`

[3] FISMA: The Federal Information Security Management Act promotes the development and enforcement of key information security standards. For more information, refer to the following address: `http://csrc.nist.gov/groups/SMA/fisma/index.html`

1.5 Identity and access management governance

As personally identifiable information has increasingly become subject to government regulations, the risk to organizations of failing to comply has increased as well. Identity and access management solutions provide a way organizations can try to manage this risk. In this section, we explore the importance of identity and access management to securing and managing the organization’s sensitive data. The topics covered include improving visibility into the organization’s IT environment and ensuring data access to the right entities when needed, improving operational efficiency through identity and access management automation, and the importance of consistent enforcement of identity and access management policies.

1.5.1 Critical data: Ensuring authorized access only when needed

Governance of identity and access management both within and across organizational boundaries entails the ability to verify both the identity and entitlements of the users and services attempting to access resources in the IT environment. A system must be able to grant or deny access based on the entitlements of these users. Additionally, system administrators must be able to grant or revoke new entitlements to users and groups and propagate those changes throughout the entire system.

Furthermore, organizations need to be able to manage access control not only on the application level, but also at the sub-application level with entitlements on individual operations within an application. These entitlements specify who is allowed access, what they are allowed to do, and under which contextual circumstances. For example, an online banking application might provide the following operations:

- check_balance(account_id)
- transfer_amount(src_account_id, dest_account_id, amount)

For simplicity, assume that there are only two groups of users: customers and bank staff. Bank staff should reasonably be allowed to check the balance of any account ID, while customers should only be allowed to check the balance on their own account. Similarly, bank staff should be able to transfer money between accounts in such a way that the transaction is logged so that it can be reflected in the account’s statements and audited later if necessary. Conversely, customers should only be able to transfer funds from their own accounts to destination accounts with the same auditability of the transaction. These requirements constitute the business rules that should be used to govern this simple banking application.

Translating these business rules into an actionable form is an essential challenge to overcome. With the proliferation of both client entities and protectable resources within most IT environments, overlaying business rules while managing the identity records and access entitlements can quickly become a cumbersome and complicated task.

1.5.2 Driving operational efficiency through automation

Intelligent identity and access management solutions can ease the burdens of assigning roles and entitlements by automating the discovery, pre-processing, and collation of identity data. These data include business roles, entitlements, and other attributes, giving system administrators a reasonable basis from which to build an identity and access control system.

To extend the previous banking application example, suppose the bank's managers have invested for the first time in an identity and access management system. It should be able to scan the user registry and automatically assign users to the appropriate roles, with the correct entitlements.

As the business requirements evolve, so must the business rules that make up the system of identity and access management governance. When the business rules change, so must the roles, entitlements, and other identity attributes. The identity and access management system can reduce the manual intervention required to complete these changes by automating the life cycle of roles and entitlements.

1.5.3 Enforcing consistent policy enforcement across the IT environment

A policy-based approach is the most direct way to conceptualize a group of business rules into an actionable directive. Collecting business rules for identity and access management and translating them into an enforceable form results in the production of a comprehensive IT security policy. This policy governs the IT resources of the organization and determines who is allowed access to each resource and under which contextual circumstances. In order for this policy to be an effective tool of governance, it must be enforced consistently across the IT environment.

Many IT environments are heterogeneous, with systems running existing applications on old hardware integrated with modern systems and new application types. As discussed earlier, service-oriented architectures can help to loosely couple these systems and facilitate communication between them.

However, it is important to be able to enforce IT security policy consistently, regardless of the technology endpoint involved. For example, the same identity and access management policy used to govern a composite web service application on a distributed application server platform should also be applied to an existing application running on the mainframe.

Consistent policy enforcement across the application and IT environment can reduce the complexity of identity and access management governance, as well as the cost of compliance with privacy and security regulations.

In addition, consistent policy enforcement can help to drive business workflow and automate business processes, resulting in less human intervention, greater efficiency, and reduced cost to manage the organization.

1.6 Compliance management

Organizations must be able to prove they are in compliance, for example, with regulations that govern the use of sensitive information. Compliance management can help to reduce the adverse consequences to an organization of failing to meet the requirements of these regulations. This section examines the importance of compliance management for application, process, data, and information security, discussing the impact of regulation and privacy concerns, assessing compliance requirements, and the relationship between compliance and governance.

1.6.1 Regulation and privacy concerns

In recent years, high profile corporate financial scandals and the increasing migration of personal information to online systems have ushered in a new era of government regulation of information. Such regulations include the Sarbanes-Oxley (SOX) Act, which introduced strict new financial reporting requirements. In the healthcare sector, the Healthcare Information Portability and Accountability Act (HIPAA) places strict constraints on how patient data is stored, transmitted, and shared. In the financial sector, the Basel II Framework promotes basic standards for banks worldwide.[4]

[4] Basel II Framework: Devised by the Bank for International Settlements to set minimum capital adequacy requirements. For more information, refer to the following address: http://www.bis.org/publ/bcbsca.htm

With the emergence of social networking tools, unprecedented amounts of personal information has been uploaded to networks such as Facebook[5], LinkedIn[6], and Twitter[7]. For individuals, these online networks constitute an efficient way to communicate with friends, family, and professional associates. However, the risk to privacy can result in the inadvertent disclosure of personal information, which can cause damage to lives and careers, in some cases.

For organizations, these networks represent a new avenue for the accidental release of sensitive corporate data to unintended audiences. The demand for a high degree of privacy control and business policies to prevent data leaks places new governance requirements on organizations.

Failure to comply with government regulations and to meet the privacy needs of users can result in fines, litigation, and loss of revenue and customers. This can present a significant business risk; managers who do not handle this risk effectively do so at the peril of their careers and the organizations they manage.

1.6.2 Assessing compliance: The audit trail

Compliance depends on being able to verify that the organization is adhering to the regulations and following its own policies as well. Adherence to the rules can be verified only if the business functions implemented in the IT environment leave transparent audit records that can be independently verified later by an auditor.

Auditability of services ensures accountability for actions taken within the organization's systems, and it provides two primary benefits: verifiable evidence of compliance and a source of event data should problems arise.

Just as identity and access management benefits greatly from a policy-based approach, so does compliance management. Asserting in a policy which operations require an audit record helps the business manager to translate the compliance requirements of government regulations into an enforceable business rule.

[5] Facebook is a popular social networking website. For more information, refer to the following address: http://www.facebook.com

[6] LinkedIn is the leading professional networking website. For more information, refer to the following address: http://www.linkedin.com

[7] Twitter is a prominent microblogging service. For more information, refer to the following address: http://www.twitter.com

Ideally, this audit data is collected in a central repository, from which managers can generate reports, both to gather business intelligence and to demonstrate compliance. Effective compliance management requires the ability to assert rules, record the enforcement of those rules in audit records, and use the audit data to demonstrate proper enforcement. However, compliance management entails not only looking to past audit records to verify compliance, but also monitoring the system's enforcement points in real time for possible infractions of the business rules.These management, enforcement, and reporting tasks can be disaggregated and delegated to appropriate roles within the organization, leading to greater flexibility and the possibility for greater alignment between the compliance management IT systems and the business roles of the organization.

1.6.3 Relating compliance management and governance

Compliance is inextricably related to governance and organizational control. In a complex IT environment, an application may be composed of both internal and external services, with the distinction not visible to the user. Nonetheless, these intraorganizational connections may impose both complex identity and access management and regulatory compliance requirements. A responsible manager will need to cope with these requirements, while providing a usable and compelling service to users. Compliance and organizational control are two related concepts that are important to understand.

Note: Being compliant versus being in control: If you have ever been audited (or audited someone), you probably know that there is a difference between being:

- *In compliance*: All your systems and processes are operated and delivered according to the security policies and standards (and you have evidence for that).
- *In control*: You know what is in compliance and what is not, you know why, and you have a plan of action.

Now, what is more important? Being *in control* is. Because you could be in compliance by accident. Further, if you are compliant, but not in control, chances are high that you will not stay compliant for long.

If you are in control, you will end up being compliant eventually. Or at least you will have it on record why you are not compliant.

And if you are not compliant and not in control, gaining control should be your primary goal.

Managing this complexity requires a normative statement as to how each component of the application should be accessed, what information can be shared across the organizational boundary, by whom, and what records of the transaction should be retained. Using a policy-based approach, this normative statement can be translated into a comprehensive IT security policy that can be centrally managed, distributed, enforced, and audited across a heterogeneous environment. Approaching compliance in this manner results in disciplined governance and vice versa.

Effective compliance management can result in better governance, reduced technological complexity, and lower cost of compliance.

1.7 Data and information security

Effective governance of SOA, identity and access, and compliance allows the organization to manage IT business services, adherence to business policies and government regulations, and access to data. However, none of these explicitly manage the security of the data itself. This section explores the importance of data and information security, discussing the risk organizations face if their critical data is improperly secured, the need for context-based information access, and data security in a cloud-based and SOA-based environments.

1.7.1 Risk of unauthorized access and data loss

Sensitive personal and business data, such as health and finance records, need to be protected. Newspapers are awash with embarrassing admissions of the improper exposure of credit card information, social security numbers, and other sensitive information from organizations that failed to protect sensitive data adequately. An important part of security governance is ensuring the security of data both in transit and at rest.

Data protection involves protecting the confidentiality and integrity of information where appropriate. Organizations can protect data confidentiality using encryption and integrity using digital signatures or message authentication codes. IT security policy can be used to specify the strength of encryption or signatures and which cryptographic algorithms should be used. Failure to do so can result in the loss of valuable proprietary information, litigation, and loss revenue, among other things.

Unauthorized access to an organization's data can result in the accidental exposure of confidential proprietary information to competitors. Such accidents can destroy an organization's competitive advantage or reveal weaknesses that would otherwise have remained concealed. Similar, accidental disclosure of sensitive customer information can obligate the organization to launch expensive mitigation efforts, make embarrassing disclosures in the media, and, in some cases, defend against litigation.

1.7.2 Context-based information access

In addition to protecting the confidentiality and integrity of information using cryptographic technology, an organization may also want to grant access to certain information only in certain contexts.

For example, if an organization were engaged in sensitive discussions regarding a potential merger or acquisition, the organization might want to grant only those few individuals directly involved with conducting the negotiations and the due diligence access to that information. If either of the organizations in this hypothetical scenario were publicly traded corporations, premature disclosure to the public of such discussions could adversely affect the stock price of either organization, which might alter the negotiating terms or derail the deal entirely.

Similarly, if an unidentified person inside the organization were to gain access to that information without its public disclosure, she could possibly use that information to make advantageous securities trades, otherwise known as insider trading, a serious offense punishable by incarceration. If the relevant regulatory agency, such as the Securities and Exchange Commission, determined that the organization has been negligent in allowing the disclosure of this information, it could levy significant fines or other sanctions against the organization.

We can see from this example that the ability not only to encrypt and digitally sign sensitive data but also to grant access only on a need-to-know basis can be a significant advantage to organizations by helping to limit the risk of accidental information exposure.

Similarly, an organization might want to grant access to certain data based on the user's role. For example, in a medical facility, patient information is highly confidential.To access an individual patient's record through the electronic health record system, the organization mighty specify that only that patient's physicians are authorized to read that person's file, while lab technicians are allowed to add new lab results to the file, they may not change previously recorded results or read anything else in the record.

The capability to specify fine-grained access entitlements provides another way for an organization to try to ensure the security of its sensitive information, by limiting access only to certain individuals in certain contexts.

1.7.3 Data security in cloud and SOA environments

In our discussion of SOA, we have briefly examined the trend in IT organizations toward disaggregating tightly coupled applications into reusable service components that can be reorganized when needed into new composite applications and even shared across organizational boundaries. Another trend is the widespread adoption of cloud computing.

Note: The term cloud computing has been used to refer to many different things. In this context, we use it to refer to the delivery of technological capabilities as a service hosted on servers outside the organization and delivered through the Internet. These capabilities can range from software to hardware infrastructure, all delivered as a service. A familiar analog to this model of delivering technology is the electrical utility: A customer subscribes to the electricity provider and pays for usage of the service. The service continues as long as the customer continues to pay. Cloud computing services benefit an organization by providing access to software technology or additional hardware infrastructure without having to purchase, install, maintain, and support these goods; they are all included in the price of the service.

Both SOA and cloud computing present opportunities for the organization to enhance its flexibility, ability to collaborate across organizational boundaries, and operational efficiency. However, they present new challenges with respect to data security as well.

When using cloud-based software services, an organization's data often resides on servers in a data center, over which the organization has no control. Similarly in a SOA environment, an organization's data might be subject to access by parties external to the organization.

In the first situation, the organization must rely on the cloud provider to ensure the security of its sensitive data and protect against unauthorized access by other organizations and employees of the cloud service provider. In the latter situation, the organization itself must provide robust security to protect its data from unauthorized access from beyond the organizational boundary.

For a concrete cloud computing example, suppose a relatively small organization that hosts a service online experiences an unanticipated spike in traffic and that they lack the financial means to immediately purchase extra server and storage capacity. The organization decides instead to rent new capacity from a cloud infrastructure provider. The organization's managers are rightly concerned that if their sensitive data resides on machines they do not physically control, they will significantly increase the risk of data leakage. Even though their cloud provider hosts the data, the organization itself is still responsible for ensuring the data's security.

In this situation, the cloud provider can help to ensure the security of its customers' data by providing fine-grained context-based access control and cryptography services, configurable for its different customers' needs.

Similarly, in a SOA environment, organizations might want to provide external users access to internal services. In this case, the service provider organization would need to protect its own information using context-based access control and cryptography to ensure that users of the service gain access only to data they are authorized to see.

In either scenario, the organization must take steps to ensure the security of its data. Context-based access control and cryptography can play an important foundational role in the security solution for these scenarios.

1.8 IT security policy management: A unifying solution

We have examined three interrelated business drivers: identity and access management, compliance management, and data and information security. Identity and access management can allow IT architects to control and audit who accesses the organization's sensitive information. This access control and auditability aids the organization's managers to assert compliance with relevant regulations and enhances the security of sensitive information. This enhanced data security further reinforces the organization's ability to comply with regulations and protect sensitive or confidential information from accidental disclosure. Management of compliance, identity, and access control requires a mechanism to specify the rules, according to which the system can act to grant or deny access to data. These rules represent the intentions of the organization's managers in service of the business goals and can be expressed and managed as a policy that describes the capabilities and requirements of services in a growing number of organizations. Business goals need to be implemented and managed consistently across the organization. Policies can be used to capture business goals and enforced across a heterogeneous environment. A flexible policy management framework must be in place to align business goals with IT assets and the implementation of security. Just as communication between services is externalized from the services themselves in a service-oriented architecture, so should the IT security policy definition and enforcement be external to the applications and services in the environment. This topic is covered in greater depth in *IBM Tivoli Security Policy Manager*, REDP-4483.

Policy management solutions can provide a snapshot of the rules used to determine how the organization's data is used. Managers can use this snapshot as a benchmark from which to measure different aspects of governance, such as transparency, processes, and stakeholder interests. Subsequently, we discuss how organizations are using the control over their sensitive data gained through the application of security policy management to satisfy their governance and compliance needs.

1.8.1 Addressing governance

A managed IT security policy unambiguously defines the rules by which data is stored, transmitted, and exchanged. Furthermore, it places constraints on how services can be composed and how users can interact with them. In effect, this policy not only protects the resources of the system, but also codifies the governing principles of the business.

With respect to identity and access management, IT security policy defines how data can be accessed and by whom. As services and resources in a system proliferate, the ability to manage this policy and apply it correctly to each resource becomes a more complex challenge. To provide the flexibility businesses need, IT security policy requires the ability to specify fine-grained entitlements down to the level of operations within applications. Resolving the policies that govern two distinct parts of the application to provide the correct effective policy for each user is critical and difficult when many sub-policies are in place for a user role or set of services. Automating this function allows business managers to focus their attention in more valuable ways.

Similarly, as organizations move towards service-oriented IT architectures, these security policies can define the rules governing how the reusable services in the environment can interact with each other. Having a systematic approach that allows policy to be versioned and adjusted gives the organization's managers more control in aligning the behavior of IT resources with desired outcomes for the organization. Because the effective policy is a snapshot of the rules governing the organization, changes to the policy can be linked to desired improvements in the performance of the organization and measured. In this way, policy management can help to provide a systematic way to manage and enhance governance.

For example, an organization can focus on creating the high-level rules to govern itself, from which they can author and manage policy using a central administration point. The central administration point provides a common way to manage policy across the entire IT infrastructure, a distinct benefit in highly heterogeneous environments. The central policy management structure can automatically handle distribution to the different enforcement points in the system, keeping policy versions consistent throughout the environment, reducing the manual human intervention that would otherwise be required to complete this task.

In addition, sophisticated policy management solutions should facilitate the authoring of IT security policies by automatically discovering and organizing identity attributes, inferring role information, and estimating appropriate entitlements from existing user and service registries.

1.8.2 Compliance and data security

As discussed previously, policy management can provide a snapshot of the rules used to govern the business and an auditable trail to assert compliance with these rules. These rules can try to accomplish a range of goals from implementing government regulations to enforcing special data security requirements. Regardless of the goals, these rules can be translated into an appropriate policy. Subsequently, this policy can be tracked, managed, and enforced. In this way, policy management solutions can be a valuable part of an organization’s efforts to comply with regulation and ensure the security of its vital information.

Automating the integration of identity and access management with IT security policy management in a centrally administered solution can reduce the cost and complexity of compliance with internal and external regulations. In addition, providing a mechanism for consistently enforcing IT security policy across the IT environment further reduces the cost and risk of failure to comply. IT security policy management systems can also facilitate the monitoring of policy enforcement, collecting of audit records, and reporting of audit results to stakeholders. Monitoring, auditing, and reporting help the organization ensure and assert compliance. Even without specific regulations in effect, they allow business managers to understand how critical data is flowing through the system and can serve as an early alert in case of problems, such as unauthorized access attempts and possible data leaks, among others.

In the case of data leaks in particular, a comprehensive IT security policy management system allows administrators to specify the precise level of protection different data in the system should have. This has two significant benefits: to prevent unauthorized access and corruption of data and provide consistent enforcement and management of data across the organization.

1.8.3 Risk management and the cost containment

The global economic downturn that began in 2008 and persists today has highlighted the importance of:

- Risk management
- Cost containment

IT security policy management can be a critical tool in managing the business risk posed by insecure data and IT systems and reducing the cost of operations. As previously discussed, security policy represents the rules that govern how the organization's data can be used. An unambiguous security policy can be automatically and consistently enforced across the organization, reducing risk in a number of ways. Risk and expected costs are proportional to each other.

For example, if the policy has been created to facilitate compliance with regulations and enforced consistently, the business risk of failing to comply will be reduced. Failing to comply with regulations can result in fines, sanctions, litigation, and in extreme cases incarceration of the organization's officers, a expensive outcome. By reducing the risk of this event, the expected cost is also reduced.

In another example, if the policy captures a strict set of data security rules, the risk of unauthorized data disclosure will be reduced. Unauthorized data access can result in the exposure of sensitive business and customer data, which can cause loss of customers and diminished competitiveness in the market, resulting in loss of revenue. In extreme cases, this situation can force an organization into bankruptcy. Reducing the risk of unauthorized data access can reduce overall business risk, reducing the cost of capital to the organization and, therefore, the cost of operations.

In an ideal world, security considerations would play an important part in the design of any IT solution from the beginning of a project. In real life, however, this has not always been the case. For various reasons, security measures remain non-functional requirements and hence often an afterthought. Time constraints or lack of appropriate skills sometimes force organizations to proceed with a solution that has only basic security mechanisms in place. But, after this solution is running in a production environment, the need to revisit and improve security becomes paramount to manage risk, either because of real security incidents or as a result of audits and reviews carried out by an independent entity. In such cases, organizations see the value and even realize an additional benefit of the externalization approach: It allows organizations to easily add and change security measures to a given application as business needs progress.

The austerity imposed by the drastic economic shift has made the reduction of cost another important driver of net income in a climate where revenue growth has slowed across the economy. In addition, in the wake of financial scandals and high-profile data breaches in recent years, businesses must now comply with a new raft of regulations. IT security policy management can help reduce the cost of compliance. These savings can provide managers with the opportunity to demonstrate financial discipline in the face of adverse economic conditions.

1.9 Introduction to IT security policy life cycle management

Organizations, the economy, and governmental regulations are all dynamic, changing in response to shifts in the business cycle, real economic events, and political conditions. Therefore, business policy must be dynamic as well, changing in response to new conditions. As business policy changes, the effective security policy governing the organization must also evolve. Security policy management solutions should provide a way to examine this evolution of policy in a holistic way and provide intelligent tools for managing changes. Among these tools is a life cycle approach to policy that allows for creation, implementation, and feedback from a policy, which can then be used to create the next more effective version of the policy. This section presents an introduction to a life cycle approach that conceptualizes policy as a dynamic entity, just like the organizations it helps to govern.

The aim of IT security policy management is to define and manage an end to end view of a security policy by modeling and transforming business processes into operational roles and entitlements at an operational level to enable context based, data-centric enforcement. To effectively manage this process, a holistic approach that manages security policies throughout the full life cycle of applications is required. All aspects of a security policy life cycle should be continuously monitored, evaluated, and updated in an iterative approach that meets the dynamic needs of business.

Figure 1-5 shows that a full policy life cycle should be considered at all levels of business and technology across multiple domains within an organization.

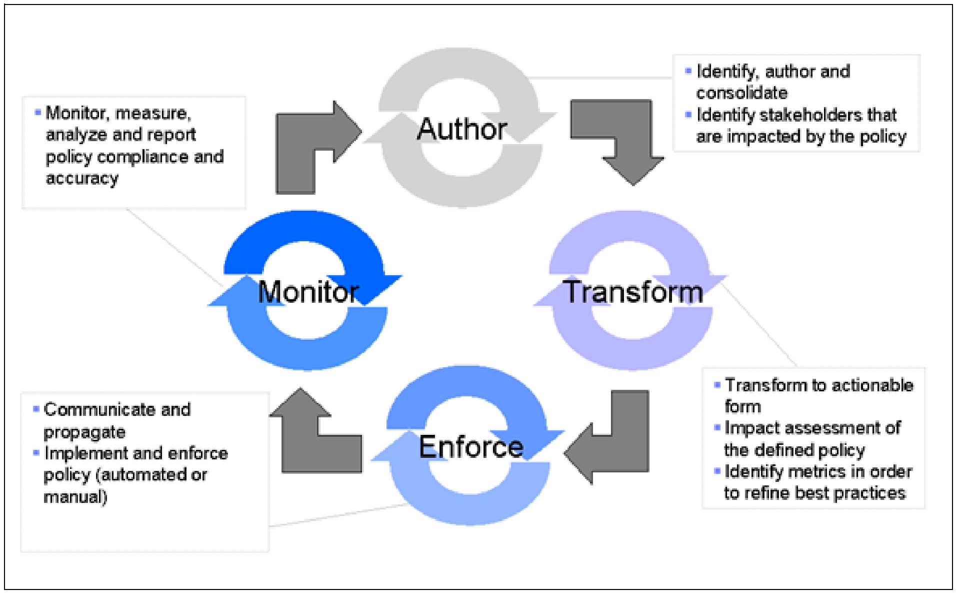

Monitor, measure, analyze and report policy compliance and accuracy
Author
Identify, author and consolidate
Identify stakeholders that are impacted by the policy
Monitor
Transform
Transform to actionable form
Impact assessment of the defined policy
Identify metrics in order to refine best practices
Enforce
Communicate and propagate
Implement and enforce policy (automated or manual)

Figure 1-5 The policy management life cycle

Using the life cycle model: We are using the policy life cycle model throughout the discussion of our usage patterns, business scenarios, and so on. Follow this life cycle model whenever you work on your IT security policies.

The policy management life cycle can be decomposed into four key phases that will be discussed in this section:

- Policy authoring
- Policy transformation
- Policy enforcement
- Monitoring

Several key tasks should be considered in all phases of policy life cycle management as part of an overall identity governance strategy:

- Communication with executive management

  Executive consultation and commitment should be obtained to sponsor any security policy management project. Policy management should be implemented using an iterative approach to incrementally refine an effective security policy infrastructure. An executive sponsor should be consulted at each stage of the project to ensure that objectives are continually prioritized in line with business needs and that investment is meeting the required objectives.

- Audit, compliance, and life cycle feedback

  A key goal of implementing policy management is allowing an organization to meet both internal security policy reporting and mandatory external identity governance requirements. By auditing all phases of the policy management life cycle, detailed data can be collected about aspects of policy creation, deployment, and enforcement to improve visibility of key enterprise resources in a holistic approach. Auditing all phases of a policy management process also allows continuous evaluation of policy effectiveness so that an iterative approach can be adopted for policy refinement.

- Change control

  Change control is an important aspect of all phases of policy life cycle management. A well defined, auditable, and secure change control process should be implemented to support compliance goals. Change management needs to be considered across all aspects of a policy management solution.

1.9.1 Policy authoring

Policy authoring is the process of translating business orientated security requirements into policies consumable by IT operations. It aims to address a key gap in the deployment of IT security measures between business process owners and IT professionals. Business process owners formulate high-level security requirements that IT professionals use to architect, implement, and manage security measures that map business requirements to operational IT solutions.

An executive sponsor should be consulted at each stage of the project to ensure that objectives are continually prioritized in line with business needs and that investment is meeting the required objectives. This is particular true when considering the authoring of policies, as it provides an opportunity to interact with executive management in a structured manner. Although there are no strict rules on where a security policy management undertaking should start, policy authoring is often the most logical starting point.

Security policy authoring involves analyzing existing and proposed IT environments to discover and classify metadata that captures key security information for identities, services, and contextual information. This understanding is necessary to effectively model a policy and simulate its effect on an IT environment prior to deployment. The authoring task consists of three phases:

- Importing resources and metadata
- Modeling and policy simulation
- Policy authoring

The outcome of policy authoring should be a policy that can be transformed and consumed by a policy enforcement point, monitored and evaluated for compliance, and fed back into the policy management life cycle. Change control is an important aspect of all phases of policy life cycle management. A well defined, auditable, and secure change control process should be implemented during policy authoring to track entitlement modification through a policy change.

Importing resources and metadata

to effectively model business policies, business processes should be defined in abstract IT terms. Before policy modeling can be considered, a process of discovering security related metadata should be implemented. Metadata representing identities, services, and contexts from heterogeneous systems should to be mined and centrally collected to prepare it for policy modeling.

The first step is to analyze details of services, application roles, entitlements, and data-level access requirements to extract metadata about those entities in a centralized repository. Appropriate tooling and processes should be used that allow for the collection of metadata for:

- Roles

  A logical first step to approach the challenge of securing an IT infrastructure is for organizations to model business and IT roles. Role management is a process that enables discovery, creation, and ongoing change control of an enterprise role structure governing user access to resources. After an enterprise role structure is defined, a role based access control (RBAC) system can provide a user-to-role and entitlement-to-role assignment that offers quick compliance, risk, and governance value to executive management. Role management provides the starting point for applying fine-grained, data centric, and context sensitive policy management.

  Tooling is required to support the role management process within an organization. If an organization has an existing role management infrastructure, it can be used to discover and import metadata about roles into a security policy management solution.

- Services

  Services represent a mechanism for exposing key business operations to consumers. Information about services can be mined from an existing infrastructure if appropriate tooling is available and open standards are used to define services. Metadata representing key aspects of services should collected so that services can be defined and classified for the purpose of policy modeling and simulation.

- Data

  Role based access controls systems have limitations in the granularity of data they use. It may be necessary to mine and classify data about the IT environment for use in policy modeling. An analysis of data from heterogeneous sources should be completed and compared against business processes to establish classes of data required for authorization. This involves a process of mining and centralizing metadata. The output of this process should be a collection of metadata representing key aspects of the IT environment that can be classified and modeled in policy modeling and simulation.

- Other resource taxonomies: Generic resource definitions should be analyzed so that metadata can be mined and centralized for classification and used in policy modeling and simulation.

Modeling and policy simulation

Policy modeling is the process of using centralized data to effectively model a business process. The aim is to capture business policies and define entitlements in terms of classified groupings of metadata representing key aspects of roles, resources, application data, and contextual constraints. The following aspects should be considered:

- Classifying information

  Business policies need to be articulated using an abstraction of the underlying IT data representation to convey enough meaning to IT operators so that detailed policies can be authored. Resources, data, and services should be grouped under logical labels that represent common sets of authorization groupings. As an example, a physician seeking to view a patient’s medical history may request access to the their medical records. The IT representation of a medical record may be represented by data distributed across heterogeneous system. Consider that a medical record is composed of data from two tables in a database system. This information can be logically considered as a single data set and should be classified as patient medical data.

- Model policies

  Policies should be modeled using classified data. They should contain a series of rules that permit or deny defined actions for subjects. Each rule maps classified resources to subjects and adds contextual conditions. For example, a rule may specify that access should be granted to a physician if they are granted the appropriate physician role and the data they are accessing is considered patient medical data. An additional constraint can be imposed to allow access only within business hours. Subjects in all other roles or without the required data-level entitlements or context should be denied.

Following a process of classifying resources, data, subjects, and environmental constraints to produce policies allows for common sets of authorization groupings to rationalize and promote re-use of logical groupings where possible. This approach allows the number of managed policy objects to be significantly reduced, facilitating more effective policy management.

As multiple policies can contain multiple rules, policies can become complex in large deployments. It is desirable to validate a policy before moving to implementation where the cost and complexity of fixing errors is much higher. A policy simulation phase should be considered to analyze and simulate policy models so that errors are not propagated to the implementation phase. The modeling tool should provide a mechanism to evaluate a model policy for correct behavior.

Policy authoring

Policy authoring is the process of taking a modeled policy template and authoring a completed policy that can be transformed and operationally deployed. IT operations take a modeled policy template and add details to abstract business definitions. The following tasks should be considered as part of policy authoring:

- Author policies

  Based on policy templates, authors complete policies to include all the details required for a configurable policy. Policies are defined in terms of roles and rule parameters. Rule parameters define a series of conditions that are evaluated to make an authorization decision.

- Implement custom extensions

  Any custom extensions needed to support an implemented policy should be planned, designed, and built at this stage.

- Change control

  Change control should be implemented to track all authored policies and custom solutions. These policies and solutions become the baseline for entitlement governance by collect policy artifacts.

1.9.2 Transform

After policies are modeled and authored using well defined open standards, policies need to be transformed into a format that can be consumed by enforcement agents. Transformation can be decomposed into two components:

- Policy configuration
- Policy distribution

Configuration

Policy configuration is the process of transforming an abstract policy to add binding information so that policies can be communicated to decision points in terms of real IT infrastructure.

Policy authoring is performed at a *policy administration point* (PAP). PAPs do not perform any policy decisions or enforcement tasks. The output of policy authoring are policies defined in a canonical format that uses open standards for interoperability, which allows a policy to be distributed to multiple decision points by adding configuration information specific to each decision point. A distributed policy consists of two parts, a canonical policy and configuration information that binds the canonical policy to the underpinning IT infrastructure that supports the implementation. Policy decisions are performed by a *policy decision point* (PDP). There can be one or more PDPs to which a policy can be deployed. Transformation must to be performed for each PDP.

Distribution

Configured policies need to be distributed to decision points in a controlled and auditable manner. After a policy is transformed to bind authored policy roles and rule parameters to real IT resources for each PDP, a process is required to distribute policies. Each time a policy or the associated policy configuration is changed, the following items should be considered:

- The distribution of the new policy needs to be approved by stakeholders defined by a business workflow.
- Change control should ensure that a policy change is audited.
- The policy should be published to the PDP on a schedule determined by the business.

1.9.3 Enforcement

After policies are modeled, authored, transformed, and deployed, policies need to be enforced. Enforcement is implemented through a *policy enforcement point* (PEP). A PEP allows a local resource manager to call out to a PDP for an authorization decision. Each PEP has its own resource specific mechanism to enforce the required policy. A canonical policy and binding information configured turning transformation is used to implement authorization decisions.

Using a centralized policy decision model allows automation of policy management, which is typically handled by local administrators. However, strong integration is required with enforcement points to make an automated policy management solution effective. The integration points must support the required coverage of distribution targets and provide the flexibility security services require.

A proper enforcement design should contain the following information:

- Enforcement agents required
- Systems requiring deployment of policy enforcement agents
- Custom enforcement point solutions

A change control process should be implemented that tracks and stores the following information:

- Enforcement design

  An enforcement design document should be kept under change control, as the design may change over time.

- Custom enforcement solutions

  Custom enforcement solutions should be kept under change control, as those solutions may evolve with changing technology requirements.

- Enforcement point deployment status

  Active enforcement points should be tracked under change control so reports can be generated and this data can be used as feedback to the policy life cycle.

1.9.4 Monitor

A security policy is authored, transformed, and enforced throughout the life cycle of an application. All changes to this infrastructure must be closely monitored for compliance and continuous feedback to the policy life cycle. A holistic monitoring solution needs to be implemented to provide the ability to collect, analyze, and report to key stakeholders.

Three classes of monitoring can be considered:

- Policy runtime audit data

  Most if not all systems in the IT environment are capable of collecting data specific to its function. As policy management involves using distributed data to author and enforce policies, an approach is required to define what audit data is relevant to gain visibility into system access behavior. Monitoring access to sensitive data may no longer be achieved by collecting a single log file or audit record. Audit data and records from disparate sources needs to collected, synthesized, and presented to authorized stakeholders. This information can be useful for vulnerability analysis to detect and react to threats in a rapidly change business environment.

- Policy configuration audit data

  Artifacts from policy authoring, modeling, transformation, distribution, and enforcement should be tracked through change control as policies are implemented and refined, which allows stakeholders to monitor and analyze the policy as it moves through the entire life cycle. Synthesized reports based on policy artifacts may be useful in designing report for compliance purposes.

- Reporting

  A key function of IT systems to produce reports on various aspects of operation for management. A policy management solution should provide standard reports and provide the ability to customize reports.

1.10 Conclusion

After introducing a business-driven security approach with the IBM Security Framework, we presented the four business drivers for security policy management:

- SOA governance
- Identity and access management governance
- Compliance management
- Data security risk management

We explored each of these topics, the challenges they represent to the organization, and how security policy management can be used to effectively address them. Finally, we discussed an IT security policy life cycle management approach.

2

Architecture patterns for externalizing security from applications and services

In Chapter 1, “Business drivers and foundation for IT security policy management” on page 3, we discussed the emerging business challenges of today’s changing economy. We observed that IT security plays an important and increasing role in this field. Given that IT security is part of the business, we propose an approach that makes it part of the solution as well. We perceive policy as being the mechanism that provides the link between the business drivers and possible IT based solutions. The business view of policies is a high level, overall view by nature, and it defines general objectives. To match this perspective on the IT side, a centralized policy management approach is needed that can abstract from the specific and implementation dependant enforcement options and provide a scope that goes beyond the scope of a single entity.

Let us assume, for example, bank A has acquired insurance company B to be able to provide a broader scope of financial services. The IT systems are consolidated, so bank employees are now internal users of the insurance applications. Regulations require that personally identifiable information be provided on a need-to-know basis only, which means in particular that they must not be combined with other information about the same person.

Let us assume a mutual customer of both A and B applies for a bank credit. The bank clerk who has to decide about this credit application needs to access certain data about the customer's insurance policies, such as type of insurance and due payments, while information about any claims made to the insurance must not be shown. The insurance application must determine which information needs to be provided based on the context of the inquiry.

The most specific, and therefore least desirable IT enforcement option, is the enforcement through individual, custom coded applications. This approach bears the highest cost for both development and maintenance and it is the least agile and the most error prone approach. In our approach we combine the alternatives, which avoid policy enforcement through custom coding under the term *externalizing of IT security policies*.

The centralized approach that we are describing in this book can help bridge the gap between business level security requirements, mainly formulated in a non-technical language, and the technical rules. Those technical rules are used to manage and enforce security policies across the organization's IT and application environments by using industry standards such as *eXtensible Access Control Markup Language*[1] *(XACML), WS-SecurityPolicy*[2], *Automated Compliance Expert Markup Language*[3] *(ACEML)*, and so on.

In this chapter, we describe patterns for externalizing security policy enforcement that we have observed based on a variety of customer requirements and deployments. In Chapter 1, "Business drivers and foundation for IT security policy management" on page 3, we provided business reasons regarding the need to externalize security policies from applications. In this chapter, we discuss the more technical reasons that will drive a particular architecture pattern. Technical parameters and constraints also determine the policy enforcement options that are available.

Depending on the existing IT and application environment in an organization, a combination of more than one pattern may be appropriate. None of the patterns excludes any of the others. Having a choice here allows you to be as non-intrusive as possible when adding security measures to an existing application environment. IT components that are already present, such as a proxy, an XML firewall, an Enterprise Service Bus, or application containers, may

[1] More information about XACML can be obtained at the following OASIS website: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`

[2] The latest information about the OASIS standard WS-SecurityPolicy can be found at the following address: `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html`

[3] More information about the Automated Compliance Expert (ACE) workgroup from the OpenGroup can be found at the following address: `https://www.opengroup.org/projects/security/ace/`

be used for policy enforcement by integrating them with the common and centralized security services.

Let us now take a closer look at the different architecture patterns for security policy enforcements:

- Intermediary approach
- Container level approach
- Database level approach
- Application level approach

2.1 Intermediary approach

Organizations today are continually seeking new ways to deliver applications and services efficiently and cost effectively. Many are already using approaches such as service-oriented architecture (SOA) implementations, SOAP based web services, and RESTful web services to provide access to on-premise and off-premise applications, while increasing numbers of these organizations are exploring techniques such as cloud computing, software as a service (SaaS), and smart grid as a delivery platform. These alternatives to traditional IT infrastructures can help reduce IT and application development costs, increase opportunities for collaboration, and drive business growth. At the same time, however, they can create new vulnerabilities, exposing access to applications and services beyond traditional organizational boundaries. Organizations therefore require more than traditional IT security to manage and protect access to these applications and services.

As a result of the transformation of IT and application environments, a large number of such web services (SOAP and REST) are being deployed to support business needs. These web services need to be protected not only from vulnerabilities from externalizing them, but also from the loss of intellectual capital and company specific information. Typically, this protection needs to be achieved without any modification to the service providers. One approach is to intercept these services before the request reaches the service provider to enforce security policy to protect the service. Components such as web services gateways, XML firewalls, web proxies, an enterprise service bus, and others are examples of *intermediaries*, where a service can be intercepted and an appropriate security policy can be enforced. This approach is called the *intermediary approach* and architecturally, provides a good fit within SOA environments.

Let us review the need for an intermediary based approach through the following utility services example.

2.1.1 Customer example

A utility company with several million customers is in the process of deploying smart meters on their customers' premises for more efficient and lower-cost operations, because the smart meters can send customer usage data directly to the utility company's SOA-based IT environment, rather than requiring their employees to physically go out and read the meters. For scalability reasons, the meters send their data to concentrators, which in turn use web service interfaces to transmit this data. Those web services must be secured within the SOA environment.

The key to securing access to both on- and off-premise applications and services in this scenario is to build on the existing identity and access management infrastructure and use it to support a scalable and highly available solution for secure access to transformed resources. A centralized approach for security policy is a good way to maintain overall governance and control. Each additional service offering can then be implemented by simply adding more web services and protecting each service per business needs.

Most of the communication involved in this example has financial implications, so the implementation needs to be tamper resistant, meaning that it requires data integrity. In addition, the data flows over the public network, which is not protected against eavesdropping by nature. Therefore, confidentiality based on data encryption is also important. After the smart meters are in place, it becomes easy to increase the frequency of the read and write operations to enable more business operations, which requires high availability of the entire IT environment.

Another important consideration might not be quite as obvious: The amount and nature of the data that gets created in a full-scale smart meter environment is likely to give away personal information about the customers, such as habitual behavior patterns, periods of absence, and cultural attitudes, if it is correlated intelligently. So there may be a need for policies governing the data life cycle, duration of storage, anonymization, and auditing.

2.1.2 Integrating policy at the intermediary level

Let us now take a look at the steps required to establish the intermediary level security policy enforcement approach. The most common implementations of this approach involve SOAP based or RESTful web services; we use SOAP while discussing the intermediary pattern.

Service discovery and service registry integration

A web service typically comes with a formalized, standards-based service description. This description presents a perfect starting point for adding policy management information. However, in application environments with a large number of services, this description might not scale well. To better use the benefits of a formal service description language, organizations typically deploy *service registries* that can assist with finding specific services. These registries can also store policies and metadata, making them available to govern the services.

A security policy management solution can use the registry and repository in two ways:

- It can use the service registry to discover a service and import not only its description but also any metadata that may be associated with the service.
- The policies authored by the security policy management solution can be published back to the registry so that they can be used by other components within the IT environment, which includes web service clients for guidance as to how requests must be built, as well as policy enforcement entities for instructions about the policies they have to enforce.

Service security policy authoring

Industry standards such as *WS-SecurityPolicy* and *XACML* provide a mechanism to define security policies for message protection, and context level authorization, respectively. These standards based policies can be associated with the web services that need to be protected.

There is, however, a gap that needs to be bridged between the business policies and their representation in IT terminology. At the current state of technology, this is not an automated task. The tools that are available today can assist, but not replace, the human assessment as to how a business requirement formulated in a non-technical language needs to be represented in a formal way. However, the fact that the formal policy languages are being used facilitates the process greatly. It helps in particular to develop a common understanding between the business owner and the enterprise architect. This is also where the need for policy life cycle management as an iterative process becomes important.

Policy configuration and distribution

The formal policies are still an abstraction to a certain extent. The process of converting them into a set of automatically enforceable rules is commonly referred to as *policy configuration*, which usually means binding an abstract policy element, such as a role, to a concrete object, such as a user ID or a group, in a specific registry that can be accessed and enforced at run time.

The defined policies and the associated configuration need to be distributed to various components within an organization so that they can then be enforced. Distribution targets include, but are not limited to, the service registries. In general, policy decision points need to have access to the policies, but whether these PDPs are distributed to them directly or by means of a service registry, is specific to the implementation. In either case, it is important to have a mechanism in place that can assure that the current version of the policy gets used at any point in time.

Enforcement at the intermediary

Typically, all service requests pass through an intermediate entity (that cannot be bypassed) before reaching the service provider. There are different examples of such intermediaries; they may have been introduced for other purposes and now need to be enhanced for security policy enforcement (XML firewalls, an enterprise service bus (ESB), business process management servers, or application containers), or they may have been introduced specifically to serve as a policy enforcement point.

Depending on the specific needs (available interception points, performance implications, persistent storage options, and so on) and also depending on the type of policy that needs to be enforced (message protection versus access control), different combinations of policy decision making and policy enforcement are possible. Both tasks can be co-located in a single entity, or they can be separate from each other, in which case the enforcer will have to reach out to the decision point.

Security of runtime information

Complex fine-grained entitlements often require additional data for the decision making process that go beyond the basic *who does what on which target*. This is where the effect of the policy configuration step sets in. Any additional resources, such as directory servers or databases whose contents can influence the authorization decision, must be secured with the same due diligence as the core application environment itself.

Message security, conversely, requires a strong authentication token such as a *Security Assertion Markup Language (SAML) token* and a reliable verification mechanism that must be in place.

Auditing

In this intermediary pattern, we have touched on a variety of good reasons for comprehensive auditing and logging. Let us recap some of them here.

- In an environment that consists of numerous on-premise and off-premise devices it is paramount to have an early indication of any irregularities. If we look at the example of smart meters, it is unlikely that such a device will ever send a false identification unless it has been tampered with, there is a stealthy attack, or there is a technical problem.
- We observed that policy authoring is an iterative process involving at least two different groups of people with different skills and backgrounds, which might result in errors when crafting the policies. Audit logs can serve as a means to identify erroneous policies and correct them.
- We also observed that the data gathered by the business services have the potential to be used for profiling. Audit logs can be used to identify that attempts to perform this action have actually taken place.

More information about the intermediary approach can be found in Chapter 5, “Intermediary level integration” on page 121.

2.2 Container level approach

To share information with a wide variety of users and to facilitate valuable partner and customer interactions, organizations are creating increasingly collaborative environments with shared touch points, using IBM WebSphere Portal, Microsoft SharePoint, and custom applications built in Java and Microsoft .NET development environments. These shared touch points have brought about an enhanced user experience that needs to be controlled based on business needs without compromising any company specific information.

As organizations are building these portals and applications, they like to externalize the security policies that influence the behavior of the application or portal. Typically, these policies are based on business needs of the organization and can often change when business drivers or processes change. One way to help facilitate this situation is for applications and portals to externalize security enforcement to the application server container and focus only on the business logic. The container can then be integrated with a security policy management solution so that it can enforce the security policies defined by the business and influence the behavior of the applications and portals.

This approach to externalize application security from the application server container and have the container influence the behavior of the application by using a security policy management solution is called the *container level approach*. This architecture pattern is typically used when developing new applications or portals or for application re-engineering. Examples of application containers that can be integrated with a security policy management solution include J2EE container, SharePoint, ASP.NET, and so on.

Let us review the need for a container based approach through the following financial services example.

2.2.1 Customer example

A large financial institution is offering some of their services through online portals. This action proved successful, so more and more customers take advantage of these online channels. The institution now wants to add more services to its portals, including the most critical ones that expose a variety of mission-critical applications and services. These services involve the exchange of critical data such as credit card numbers, social security numbers, and other personally identifiable information (PII). Now the organization faces the challenge of ensuring in a productive way that user access remains secure. They must also secure access to critical data from inside the organization. Corporate security policies dictate that this access be managed on a need-to-know basis to minimize risk of both intentional and unintentional data loss or breach of data security.

Hardcoding security controls into these applications can generate redundant logic, while running the risk of introducing inconsistency among applications. Once in place, such controls are difficult to change, and they contribute to increased operational costs and security risks for the organization.

Here are some of the specific challenges that this organization and the financial services industry in general are facing:

- Being among the earliest IT pioneers, they have a long history of core applications that were written for an entirely different environment. Portals and service veneers are used to provide interfaces that serve today’s needs.
- New business initiatives and rapid growth have caused the establishment of a variety of application platforms using different technologies that are inconsistent in many ways.
- A number of mergers and acquisitions has added even more to the inconsistencies in IT. As a result, governance requirements and the need for better cost efficiency mandate a consolidation. Conversely, a smooth migration path is needed to not disrupt the daily business.

- The trend towards one-stop financial services has forced cooperation among formerly separate types of operations within the organization, such as banks, insurance companies, and building associations.
- The financial services industry worldwide is governed by a number of government regulations and standards. Compliance is not an option; it is mandatory.
- The observed overwhelming trend towards online banking and other self-service offerings available to customers has raised the bar for security controls. Sensitive financial data combined with PII is sent across the public Internet and demands the best possible protection. Incidents of fraud can damage reputation overnight.

From the above challenges these needs emerge: Applications should not have to code their own security mechanisms, a centralized approach is required for security policy management, it has to cover both external and internal communication, and a set of common security services are needed that can be used across all platforms and that can be easily adapted to change.

Given the fact that the core business of our financial services organization depends 100% on the availability of its underlying IT systems, high availability and resilience are paramount. This situation has two immediate consequences for security services, including the security policy management:

- The security services themselves must have failover mechanisms in place.
- Special attention must be paid to the prevention of denial-of-service attacks against the security services.

2.2.2 Integrating policy at the container level

We now take a look at the steps required to establish the container level security policy enforcement approach. The most common implementations of this approach involve portals and application servers, most of them either based on Java or .NET. A security policy management solution should therefore support at least these two.

Interfacing with the container

There is not a single common interface available for security policy management across all application container implementations. A policy management solution must therefore provide a flexible, extendable, and modular architecture to be able to support the consolidation challenges that we have described for this pattern. It should support existing standards for obtaining external policy decisions, such as *Java Authentication Contract for Containers (JACC)*, a well-established industry standard within the J2EE environment, by implementing a JACC provider. Standard solutions for widely deployed platforms such as SharePoint are desirable. An API to enable custom-made interfaces on other container platforms will serve as a last resort if nothing else is available.

Discovering application resources

The possibilities for automatic (or easy) discovery of application resources depend heavily on the available interfaces a specific platform provides. For example, SharePoint provides a web service that can be used to discover SharePoint resources. J2EE platforms have the potential to enable easy discovery of resources such as ESBs, web URL patterns such as JSPs, and roles through use of administrative interfaces. Unlike the web services environment with its service description language, there is no universal description language for a container in general. A good approach can be to normalize to an intermediate descriptive format similar to what exists for web services, which can help facilitate the resource import from different sources.

Policy administration

All aspects of policy administration depend on what the container platform can offer and what is standardized. In the case of J2EE, there is existing standards work that can be used. For example, a J2EE profile for the generic XACML policy description language provides mapping of J2EE resources. This is a good starting point for policy authoring, whereas role mapping needs to integrate with the respective container runtime environment and its integration with user authentication mechanisms. A consistent support for security policy administration across multiple J2EE based container platforms appears to be both desirable and achievable. A good security policy management solution should provide capabilities to support other containers, such as SharePoint policy administration, and be open to further extensions.

Policy enforcement at the container

This is the core aspect of this pattern. The fact that the applications are unaware of the security policy decisions helps to keep them simpler and easier to maintain. Developers can focus on the business logic and do not need to become security experts. Business agility is increased as changes and new applications are implemented quicker. Consistency of security policies across distributed environments more easily achieved and verified, both for existing and new applications. In the case of portals, the portlets take the position of the applications, so policy enforcement is externalized from them as well.

Auditing

In a highly regulated industry, auditing is one of the most important aspects of security policy management. Being able to place this function at the container level makes it easy to build a consistent, all-encompassing auditing and reporting solution that includes both the policy management itself and the business transactions, provided there is standard auditing support within the deployed IT security policy management solution. A need for additional log and event management capabilities may still arise due to compliance requirements and government regulations.

More information about the container level approach can be found in Chapter 6, “Container level integration” on page 159.

2.3 Database level approach

In many of today’s collaborative business environments, organizations may have to share data with a variety of partners, employees, and others outside the organization, most of whom need access only to a particular segment of the data available to them in the environment in which they share information. This data may have to be accessed from a number of applications and channels. Because it is not always possible to modify the applications to enforce data-level security, it may be necessary to enforce controls at the database level to ensure that the right information is being disseminated to the right requestor for the right reason or in the right context. The database and the data that it holds become a natural point where one can apply and enforce security policies, particularly access entitlements.

This approach, where the security controls are implemented on the databases instead of modifying the applications, is called the *database level approach*. Architecturally, this is a good fit when consistent access policies to data are required regardless of the applications and channels that access the data.

Let us review the database based approach through the following healthcare organization example.

2.3.1 Customer example

A healthcare organization is constantly receiving data, such as medical records from healthcare providers, and claims and other benefits-related information from insurance providers. At the same time, the company is sending out data to third parties, such as companies to whom billing or other services are outsourced. In such a scenario, data security requires that information be shared strictly on a need-to-know basis so that, for example, healthcare providers have access only to medical records, and not claims or billing information, insurers see only data related to claims and benefits, and providers of outsourced services are limited to views of data related to the services they provide.

Both medical and financial data represent particularly sensitive types of information that, when combined with *Personally Identifiable Information (PII)*, can cause great damage to the person to which they relate, if they get into the wrong hands. In such a case, the healthcare provider would be held liable for the damage or would face respective charges of liability due to lack of due diligence. So it is obvious in this scenario how business needs directly relate to the successful implementation of security measures.

A number of past incidents of data leakage that had wide press coverage have shown that there is a less direct, but potentially far more severe, risk to the business due to this issue: loss of reputation and resulting decline of future revenue caused by customers walking away. In many countries, it is not possible anymore to try to avoid this reputation damage by simply refusing to comment on the incident because regulations require that data loss must be reported.

The following, non-exclusive list of data related and business driven requirements have been identified based on data security needs:

- Grant access to data on a need-to-know basis.
- Make sure that access is granted consistently across all applications and all data stores.
- Pay special attention to privileged users such as system administrators.
- Monitor sensitivity of data as time evolves.
- Be prepared for rapidly growing data volume.
- Consider different requirements for data at rest, in motion, and in use.

Need-to-know principles

A user should only have access to those data elements that are necessary to fulfill the respective duties based on their role. So, for example, a physician in a hospital needs to see the entire medical record of a patient to avoid mistreatment, but should not be able to see credit card information. A person in charge of billing, conversely, needs access to the insurance data, to be able to distinguish which treatment is covered by the insurance, and which treatment is not covered and must be billed to the patient directly.

Consistent fine-grained access control

Regardless of the number of applications, business imposed rules and policies apply to the entire organization. All applications operating on the same data must follow the same rules. When applications need to be changed or new applications are introduced, it should take little effort to establish consistency of data access entitlements.

Privileged user management

System administrators often need privileges far beyond what their duties require, simply because the system platform does not provide the level of granularity for access control that the business needs dictate. Specifically, this means that a system administrator can often access sensitive business or personally identifiable data as a side effect of their administration privileges. It is a best practice to reduce these privileges by adding an additional enforcement layer that has much finer granularity both in determining a user's role as well as the significance of the data element being accessed. Administrators benefit because they are no longer a prime suspect in case of a breach because the organization can apply the same principles related to business data to administrators as well as any other user.

Changing sensitivity of data

As time passes, certain information elements may lose their former importance or gain additional importance. Data that is critical one day is less consequential, perhaps not even relevant, at another time. For example, a temporary medical condition may be critical only until the patient has fully recovered. After that, it may not matter any more that the patient ever suffered from this problem. Financial transactions may be kept secret until a certain point in time, after which they become public knowledge. Policies may be needed to address conditions like this, for example, a specified date can trigger an automatic change in the status of a data element.

Growing data volume

Data growth is exploding today, with data volumes doubling every 18 to 24 months according to some analysts. The implications for data security in the data center are vast, as organizations store increasing amounts of content there. More storage is required to support such massive data growth, and organizations must make sure that the addition of more data stores takes place in accordance to the existing access policies at any given point in time. There must not be a single instant at which this principle is violated.

Data in different states

So far we have discussed how data entitlements are handled in the context of business applications that carry out well defined, intended business processes. For the sake of completeness, regulations require policies to be put in place to ensure confidentiality and integrity of data at all times, including when they are in transit (and thus prone to unauthorized interception) or at rest (and thus potentially subject to intrusion attacks or information loss).

2.3.2 Integrating policy at the database level

We now take a look at the steps required to establish the database level security policy enforcement approach.

Database resource discovery and classification

To be able to formulate policies related to data, the context of that data must first be known to the policy management solution. A discovery mechanism is required to provide information about relevant data, including metadata such as schema and table structures, further classification of the data based on business requirements, state of the data, and so on. Data classification is a task of its own and can be carried out separately from policy management. It provides important input to policy management, especially in providing important business contexts to information. Classification applies to both structured (database) as well as non-structured data (residing in the file systems).

Database security policy authoring

After the resource hierarchy has been identified, policies can then be defined and associated with data elements or classification of data elements. Most commercially available databases already provide a mechanism for enforcing both coarse and fine levels of authorization policies; but they are specific to the target database. Hence, policy authoring should provide an abstraction so that you do not need to understand the details of all the database implementations. There may be more than one policy required to accomplish the database specific actions.

Policy configuration and distribution

Because the policies are defined with a level of abstraction, configuration is needed to map the parameters of the policy into enforceable entities on the database where these policies would be enforced. The policies and configuration information for databases may then be transformed into database commands and distributed to databases where they will be enforced.

Enforcement at the database

Because a database server receives actions to perform natively, enforcement requires merely to execute the actions, but additional conditions may have to be accommodated for a proper context. The achievable granularity depends on the combination of options available both on the management side and the specific enforcement points. When calls by different applications invoke the same type of operation on the database, consistent policy enforcement can be achieved and eventually demonstrate compliance to the business. This situation also includes control over actions carried out by privileged users with administrator rights who have opportunities to bypass the business applications and access data directly by using administrative tools.

Application integration

When using the fine grained access capabilities in databases, you need the context of the user in addition to the application or system ID that is typically used to connect to them. Some databases support this notion and provide the ability to propagate identity, in which case there may not be any application changes required. In other cases, where automatic propagation of identity is not possible, applications may have to provide it explicitly during connection time.

Auditing

Accountability of both the business as a whole and the user as an individual is essential. Auditing of policy management solution and enforcement is the key approach. Given the fact that the pattern of database level enforcement presents itself as a combined effort between the security policy management component and the database, it is likely that the same situation applies to auditing. This situation implies that a correlation of different audit logs with different formats is likely to be needed. Specialized log and event management solutions provide both the technical and the business side of this requirement. They can normalize and abstract from the specific log formats and they can assess compliance with business driven or law imposed rules and regulations such as the *Health Insurance Portability and Accountability Act (HIPAA)*. Log management also helps to prove completeness and integrity of the logs, thus making it a business supporting feature rather than just a technical component.

More information about the database level approach can be found in Chapter 7, “Database level integration” on page 187.

2.4 Application level approach

There will be scenarios where the previous approaches may not prove sufficient, either because of specialized application needs or specific authorization requirements. In some cases, the organization's needs might not align with container based approaches, even if they are supported for their application platform. The business will still need to influence the behavior of applications through external policies, and explicitly leverage an IT security policy management system. This approach is called the *application level approach*. IT security policy management and decision making is still being externalized, so all the advantages of centralization still apply. The only functions that need to be implemented beyond policy enforcement are the creation of the policy decision request, the invocation of the decision point interface, and acting accordingly on the returned decision.

As mentioned before, none of the patterns are meant to be exclusive; they can be combined with each other for specific situations and business needs. In our current discussion, the application level approach in particular is most likely to be combined with other approaches, such as the intermediary or container level externalization pattern.

Let us review the application integration approach through the following services organization example.

2.4.1 Customer example

A highly specialized IT services organization provides premium services to tax counselor firms that are generally small or medium sized. Tax laws are complex in most countries and undergo frequent changes, so the supporting software is equally complex; it must not produce false results (because this would break the law), which means it has to be thoroughly tested to the extreme, and modifications due to legislation changes have to be in place precisely on the effective date. The target market for these types of services consists of a limited number of tax counselors. Their most important asset by far is the quality of their software based service, that is, the quality of the software itself.

Let us take a look at some of the security implications in this industry:

- Counselors' offices are likely to have several employees. These employees have individually assigned customers and they must not have access to other customers' sensitive tax data. So the IT services provided require a two stage authentication process, one level for the counselor's organization as a whole, and one for the individual employee with his dedicated customers. Entitlement policies must take the authentication level into consideration.
- The counselors' employees are working with the firm's local IT environment, so they need to log in locally first. Identity federation is a user-friendly and secure integration approach with the premium service provider. IT security policy management should integrate with federated identities.
- As stated above, tax data is extremely sensitive and requires thorough protection at all stages based on the context of the request.
- Auditing takes an important role not only to prove compliance of the provided applications and services with legal obligations, but can also serve as a monitoring tool for attempted attacks, especially insider attacks and privileged user attacks. Trustworthiness is essential in this industry. A single breach can destroy the services organization's reputation beyond repair.
- Tax regulations change frequently over time. Tax data sometimes needs to be processed some time after the fact, possibly even years later. In such a case, the respective laws and regulations that date back to that point in time must be applied. The respective versions of the software and possibly respective versions of the corresponding IT security policy must be applied. An adequate version control regime for IT security policy must be in place to support this situation.
- High availability is often a requirement driven by business needs. Tax obligations generally have a deadline attached, which, if not met, has serious implications, such as fines and hefty interest being added to the amount due. Therefore, the premium services must not fail, as covered by a service level agreement. IT security policy management, being an essential component of the solution, needs to provide that same service level.

2.4.2 Integrating policy at the application level

We now take a look at the steps required to establish the application level security policy enforcement approach. More than in the other patterns, this pattern should be used as an example only, because this pattern is not as standardized as the other patterns described previously.

Application resource discovery

With a custom-written application, it is less likely that the relevant resources can be captured by a standardized method, so there is a need for a custom made provider of resource metadata. Ideally, this provider would be integrated with the software development environment of choice to extract the resources directly from the source code. We already observed, with the container level pattern, that it is desirable for the IT security policy management to have an intermediate descriptive format. This situation equally applies here, especially when the two patterns are combined in a solution.

Application security policy modeling and authoring

If the application design and development takes place at the same time as the creation of the related IT security policies, both could be undertaken in a synchronized fashion and benefit from each other. High level application design could then be accompanied by policy modeling, which may include simulation of policies as a way of early testing at a high level. This situation might reduce the cost of application development substantially, because errors cost less the earlier they are detected during the development cycle.

In a similar way, policy authoring could be done during the software coding and testing phase, and again both activities would benefit from each other.

Policy configuration and distribution

A strict separation between policy decision making and policy enforcement characterizes the application level integration pattern. We have seen that new applications in particular benefit from this approach. Conversely, there may be some extra migration effort required for existing applications. There are situations where a nonstandard user repository with nonstandard interfaces is being used for complex granular authorization. Integrating such technologies with a new, standards-based IT security policy management solution can be achieved by using custom policy information points. Standards such as XACML provide enough flexibility to map to existing complex policies.

Policy enforcement

From the sample scenario that we have discussed so far, it can be assumed that there will be intermediaries playing a role in the connection between the service provider organization and the counselors' offices. A centralized overall IT security policy management solution can benefit from this architecture by using an optimized placement of the respective enforcement functions. For example, message security is often best placed at the intermediary, both from an architectural and a performance perspective; the same may apply to coarse-grained access control. Fine-grained authorization and entitlements, however, typically make more sense in the application context.

Although the effort of implementing an enforcement point adds to the amount of development, the interfaces used by the application to communicate with the policy decision point play an important role with respect to the resulting maintenance cost. A published industry standard interface is desirable, as it can be expected to remain stable and it avoids vendor lock-in.

More information about the application level approach can be found in Chapter 8, “Application level integration” on page 213.

2.5 Conclusion

In this chapter, we have observed different approaches about how to address the task of IT security policy management, a task that plays an important role in the overall context of business driven IT governance. We have seen that there is no single approach that covers all cases, but there are recurring patterns. We have classified those patterns into four categories, being well aware of the fact that a real-life situation is a mix of several of these categories. A centralized architecture seems to be the preferred approach.

Typically, there will be one primary pattern. We provided some generic customer examples to demonstrate how business demands tend to lead to its selection. Technical considerations determine the full picture, which include platform preferences, available skills, cost, availability of standards, availability of roadmap information, and so on.

In the remaining parts of this book, we continue to discuss the different patterns and approaches, and we introduce IBM Tivoli Security Policy Manager, which is a product that provides a wide range of solutions to the challenges that business requirements impose on IT security. We see that IBM Tivoli Security Policy Manager addresses these needs by applying the IT security policy life cycle to the observed architecture patterns in a flexible and agile fashion. We demonstrate this situation by showing examples for both the breadth and depth of the solution.

Part 2

Implementing a policy life cycle management solution

We begin this part by introducing the logical component architecture of the IBM Tivoli Security Policy Manager product, which can help you implement your own IT security policy life cycle management. We then discuss important integration aspects with external systems and conclude this part by looking at different deployment considerations.

3

Tivoli Security Policy Manager overview and architecture

In this chapter, we introduce Tivoli Security Policy Manager (TSPM) followed by an in-depth discussion of the architecture of its key components, including policy server and runtime security services.

The following topics are discussed:

- Tivoli Security Policy Manager overview
- Tivoli Security Policy Manager architecture
- Example deployment physical architecture

3.1 Tivoli Security Policy Manager overview

IBM Tivoli Security Policy Manager provides centralized application and data entitlement management and SOA security policy management to enable access control for data, applications, and services. Tivoli Security Policy Manager provides capabilities to help improve compliance, and to drive identity based operational governance across the organization.

Tivoli Security Policy Manager is a standards-based application and data security solution. Along with other security systems and components, such as auditing and reporting system, it provides life cycle management for IT security policies, including authoring, transforming, enforcing, and monitoring.

Tivoli Security Policy Manager enables application owners and administrators to externalize security and to simplify the management of complex authorization policies for new and existing applications, including custom applications, and databases. The benefits include:

- The ability to respond quickly to business changes through centralized management of application roles, entitlements, and data-level access control.
- Improved compliance and security management with roles-, rules-, attributes-, and context-based access control.
- Reduced manual, inconsistent, and costly administration of security policies at each policy enforcement point.
- Operational governance with the ability to delegate and audit all changes to policies.

3.1.1 Tivoli Security Policy Manager components

As shown in Figure 3-1 on page 65, at a high level, the product has the following components:

- The *policy administration point* (PAP) is represented by the Tivoli Security Policy Manager policy server and console.
- The *policy decision point* (PDP) is represented by the Tivoli Security Policy Manager Runtime Security Service (RTSS).
- An example of a Tivoli Security Policy Manager *policy enforcement point* (PEP) is a plug-in for WebSphere Application Server to enforce container level security.

- Tivoli Security Policy Manager provides a plug-in framework to integrate with *policy information points* (PIPs), such as user registries, repositories, and databases. Some PIP plug-ins are supplied by the product, and organizations can develop their own PIP plug-ins to meet business requirements.
- A *Policy Design Tool* (PDT) is a modeling tool that can help users to model and analyze high-level security requirements in a business context and create security policy templates in a standard format for use within the IT environment. The security policy templates created by the PDT can be imported into Tivoli Security Policy Manager.

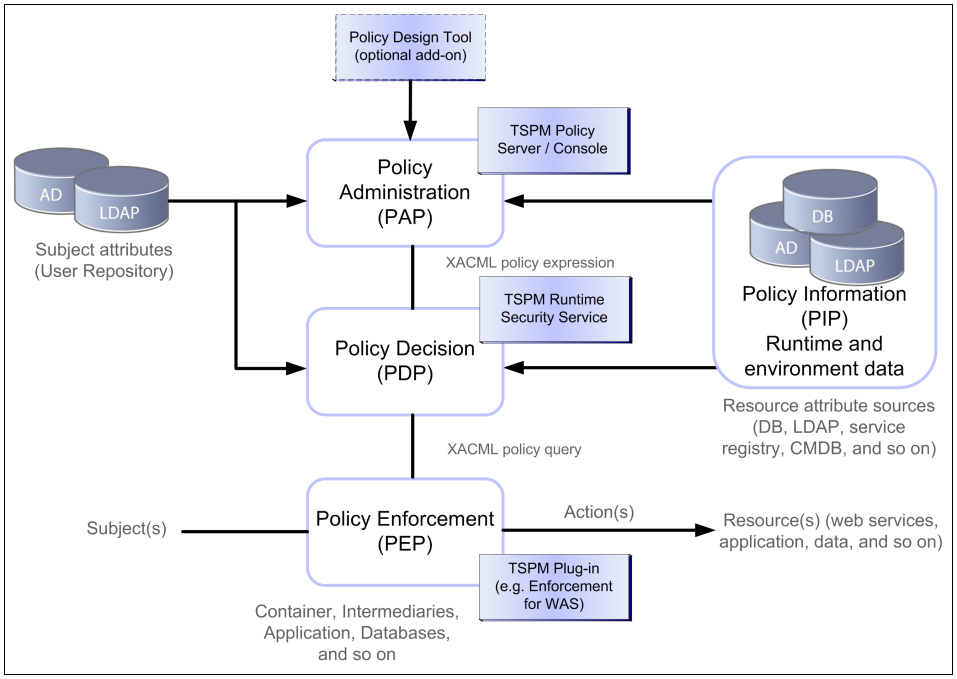

Policy Design Tool
(optional add-on)
TSPM Policy Server / Console
Policy Administration (PAP)
AD
LDAP
Subject attributes (User Repository)
XACML policy expression
TSPM Runtime Security Service
Policy Decision (PDP)
DB
AD
LDAP
Policy Information (PIP) Runtime and environment data
Resource attribute sources (DB, LDAP, service registry, CMDB, and so on)
XACML policy query
Subject(s)
Policy Enforcement (PEP)
Action(s)
Resource(s) (web services, application, data, and so on)
TSPM Plug-in (e.g. Enforcement for WAS)
Container, Intermediaries, Application, Databases, and so on

Figure 3-1 Tivoli Security Policy Manager end-to-end policy management

Policy administration point (centralized administration)

The PAP is delivered as the Tivoli Security Policy Manager policy server with the following features:

- Resource identification and discovery

  A Tivoli Security Policy Manager policy server helps you to discover existing resources and associated metadata using existing repositories and registries in an IT environment.

- Resource classification

  A Tivoli Security Policy Manager policy server enables you to classify resources into groups, for the purpose of applying policy to a given group of resources, in ways that align with the business requirements and provide ease of management.

- Policy authoring

  A Tivoli Security Policy Manager policy server enables authoring of different security policies, such as message protection and authorization policies to protect and govern services. Authoring is based on the type of policies using an intuitive user interface to cater to both application owners and IT operations. Policies can be defined based on the different contexts, such as identity, service or resource, environment, and business.

- Policy transformation

  A Tivoli Security Policy Manager policy server enables automatic conversion of the policy to the format that is required by a given target system, which is XACML or WS-SecurityPolicy. In addition, multiple policies can be combined to derive effective policies both from inheritance and direct association.

- Secure policy distribution

  A Tivoli Security Policy Manager policy server enables distribution of policy to stand-alone PDPs (such as Tivoli Security Policy Manager runtime security services) as well as to policy decision points that are embedded inside policy enforcement points and resource managers. Tivoli Security Policy Manager uses a protocol based on WS-Notification and WS-MetadataExchange, secured by XML protection measures, such as digital signatures, to ensure the integrity of the transferred policy. You can configure it to send updates over SSL for another layer of protection and confidentiality. The policies are secured both at the message level as well as by enforcing authentication and authorization so that only valid entities can get the policies.

- Delegated administration

  This feature allows you to delegate administrative tasks to individuals based on roles and responsibilities. Roles that are shipped with Tivoli Security Policy Manager include Application Admin, Application Owner, Policy Operations, Policy Author, Auditor, IT Admin, and Role Admin. In addition, you can define new roles based on business requirements.

- This feature provides auditing and logging for serviceability, reporting, and so on, with standard resports.

- Change management and version control to keep track of policy changes, versioning of policies, and so on.

Policy decision point (Security as a Service)

The PDP is represented by the RTSS.

- Tivoli Security Policy Manager provides the capabilities to handle security functions such as authentication (WS-Trust), authorization (XACML[1]), and audit (Common Base Event[2]) in standards based services. These services are delivered by the Tivoli runtime security service and can be used by enforcement points to enforce policies that are centrally defined and distributed.
- The Tivoli runtime security service provides not only token and identity mediation, but also multiple levels of authorization enforcement with different granularity based on standards based interfaces.
- For performance, scalability, and security reasons, you can deploy Tivoli runtime security service in either local or remote mode. Thus, you can evaluate policies locally if needed.

> **Additional information:** For more information about Tivoli runtime security service in local or remote mode, refer to 3.2.3, “Tivoli runtime security service architecture” on page 78.

Policy enforcement point

Where needed, Tivoli Security Policy Manager provides enforcement points that fall into the following patterns:

- Intermediary based enforcement

  Intermediaries such as a web services gateway (for example, IBM WebSphere DataPower®) can intercept a request and enforce policies before providing access to the applications. The intermediaries can function either both as PDP and PEP, or just as PEP using an external PDP (such as Tivoli runtime security services). Message protection policies can be enforced at message intermediaries, which are message handlers that can support the WS-SecurityPolicy standard.

- Container based enforcement

  The container is instrumented with a Tivoli Security Policy Manager plug-in (for example, a plug-in for WebSphere Application Server) to enforce policies without having to modify the applications.

[1] More information about the OASIS eXtensible Access Control Markup Language can be found at the following address: `http://www.oasis-open.org/committees/xacml`

[2] More information about the Common Base Event can be found at the following address: `http://www.ibm.com/developerworks/library/specification/ws-cbe/`

- Database based enforcement

  Relational database management systems (RDBMS) have their own policy enforcement mechanism. Tivoli Security Policy Manager can transform and distribute the security policy into the native policy of the RDBMS (such as IBM DB2® and Oracle Database system). The database system functions both as PDP and PEP to enforce the data access policy.

- Application based enforcement

  Applications can use either the Java API based on JACC or web services (SOAP or REST) based on XACML to retrieve authorization decisions from Tivoli runtime security services and to enforce them.

Integration with policy information points (PIPs)

Tivoli Security Policy Manager supports the use of information from existing identity management systems, identity and attribute repositories, application databases, and rules engines when evaluating authorization policy.

The context of the policy information can be based on identity, service, environment, or business.

Policy Design Tool

A Policy Design Tool (PDT) is provided by IBM alphaWorks®[3] to help architects model, analyze, and create policy templates and to check for Separation of Duties violations prior to using those templates within the application environment.

Architects can use this development tool to model and analyze the high-level security requirements in a business context and to create IT security policy templates in a standard format for use within the IT environment. Resources can be classified according to their business function using multiple taxonomies. Users can be assigned roles that capture the job functions that they are performing. Thus, the security template policies are captured with business-oriented resource classes and roles for ease of review with the business process owners. Policy design tool also provides the ability to import detailed IT security policy information that previously resided on different platforms and provides the ability to assess a complete view of the IT security policy landscape. A number of analysis functions also help ensure that policies are consistent on the business level.

[3] The IBM alphaWorks resources for the Policy Design Tool can be located at the following address: `http://www.alphaworks.ibm.com/tech/policydesigntool`

After policies are captured, modeled with roles, and verified using the Policy Design Tool, you can export them as policy templates for use by Tivoli Security Policy Manager for application and data entitlement, and SOA security operational policy management. Tivoli Security Policy Manager can use these templates to author and refine application entitlements (roles-, rules-, attributes-, and context-based) and to transform and enforce access control across heterogeneous IT environment.

Additional features

In addition to these key features, Tivoli Security Policy Manager also integrates with IBM products for governance and provisioning. Web services can be discovered from WebSphere Service Registry and Repository (WSSR) and the authored policies can be published back to the registry for governance.

Tivoli Security Policy Manager integrates with a number of IBM products and therefore safeguards the existing investment in IBM products, for example:

- The capability of SOA governance of WebSphere Service Registry and Repository
- The flexibility and speed of SOA security provided by DataPower SOA appliances
- The life cycle management of identity and accounts provided by Tivoli Identity Manager
- Tivoli Access Manager and Tivoli Federated Identity Manager

Other assets in the IT environment can also be used. Services can be discovered from other registries that support UDDI, and integration with standard user registries, such as LDAP and Active Directory, is supported as well. Tivoli Security Policy Manager integrates with the IBM Tivoli Common Reporting framework to provide unified reports on application roles and entitlements.

3.2 Tivoli Security Policy Manager architecture

In this section, we discuss the Tivoli Security Policy Manager architecture in detail, including components such as policy server, runtime security service, and other architectural aspects of the product, such as management API, data model, administrative security, and others.

To understand how the security capabilities of Tivoli Security Policy Manager can be mapped to the IBM Security Blueprint, refer to Figure 3-2 on page 71. This diagram shows the functional components of the People and Identity solution pattern, and the highlighted elements indicate the functional components that can be fulfilled, or implemented, using Tivoli Security Policy Manager. This functional highlighting is applicable for the infrastructure service components as well.

If we determine the desired functionality of a solution using the People and Identity solution pattern, the mapping shown in Figure 3-2 on page 71 can be used as a quick reference of the functional security management aspects of Tivoli Security Policy Manager. This reference can help us determine which functions of a solution can be covered by selecting this product.

Looking at the Risk and Compliance Assessment related sub-components, Tivoli Security Policy Manager provides the capability to log policy violations, and therefore can help with Risk Identification and Compliance Controlling. One general use case for policies is to control risks by implementing organizational guidelines, so you can use Tivoli Security Policy Manager to help with Risk Controlling. Finally, as policies are important from a business and legal perspective, and a policy can be considered as a specific type of business record, Tivoli Security Policy Manager can be used to help with Records Management. Furthermore, Tivoli Security Policy Manager puts the mechanisms in place to enforce access to assets, and any effort to bypass those mechanisms will be logged.

If you want to learn more about the Foundational Security Management and the Security Services and Infrastructure sub-components, refer to *Introducing the IBM Security Framework and IBM Security Blueprint to Realize Business-Driven Security*, REDP-4528.

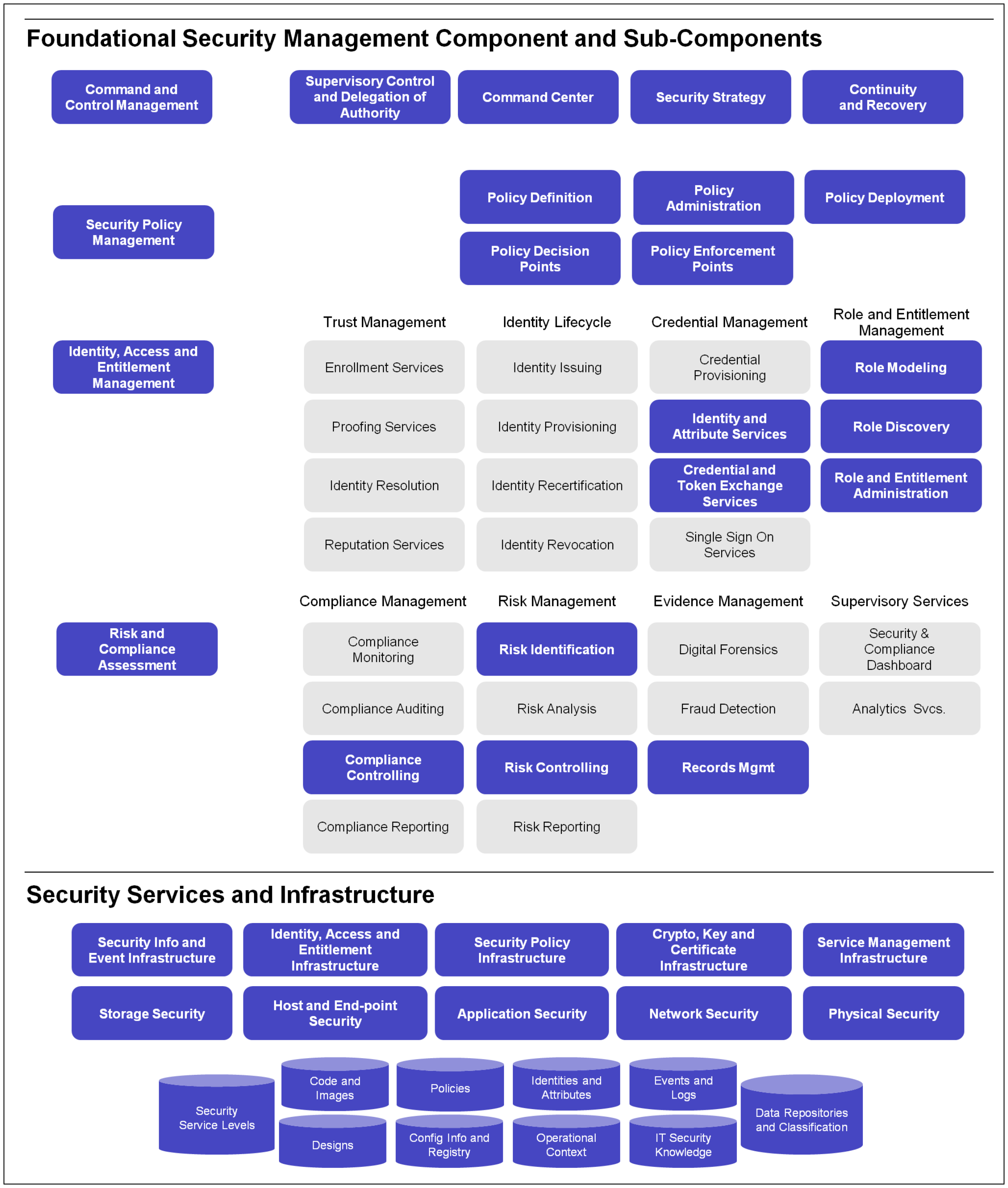


Figure 3-2 Mapping of Tivoli Security Policy Manager to the IBM Security Blueprint

3.2.1 Logical component architecture

Tivoli Security Policy Manager architecture includes three core components and one optional add-on component:

- The policy server and its console (for PAP)
- The runtime security services (for remote or local-mode PDP)
- The policy enforcement plug-ins (PEP) for specific application deployments
- The optional add-on component Policy Design Tool (PDT)

We discussed the capabilities of each of these components in 3.1.1, “Tivoli Security Policy Manager components” on page 64.

The architecture of Tivoli Security Policy Manager is entirely standards-based and scalable. The policy server is built on an Eclipse-based Open Services Gateway initiative (OSGi) plug-in architecture and is extensible to support new resources, new policies, new discovery and distribution mechanisms, new PDPs, and PIPs. The runtime security service includes an interface to integrate with existing identity management systems, existing identity and attribute repositories, rules engines, and application and business repositories.

The Policy Design Tool is built on the Eclipse Rich Client Platform (RCP). It offers a number of perspectives and views for modeling, analyzing, and debugging access control policies. Its user interface is easy to use for business users because it minimizes security-specific terminology and intuitively shows the effects of policies. The tool also contains an embedded XACML PDP so that the user can simulate access requests and debug the decision process that an operational PDP will carry out in a real deployment.

Using this Policy Design Tool, architects can model and analyze the high-level security requirements, assign users to roles, and create template access control policies for resources, which include web services and application data in relational databases. After review with business owners, these templates can be exported into Tivoli Security Policy Manager to author and refine both message protection and application entitlements (roles-, rules-, attributes-, and context-based), transform the policies, and enforce them across the IT environment. The policy templates are derived from the views of the business owners; they can be further refined by IT system owners when applied to specific IT environments.

At a high level, as illustrated in Figure 3-3, a Tivoli Security Policy Manager policy server can import service definitions and metadata from repositories, such as service registries, and can also work with existing identity and access management systems. Policies can be authored by either the Policy Design Tool (PDT) and imported into Tivoli Security Policy Manager as templates or they can be created directly from the Tivoli Security Policy Manager console. These security policies are transformed and securely distributed to various resources or to a RTSS so that the resources can enforce the policies either directly or using Tivoli runtime security service. Tivoli runtime security service can integrate with existing identity and resource attribute sources, rules engines, and so on, at run time. Finally, as we describe in 3.1.1, “Tivoli Security Policy Manager components” on page 64, the Tivoli Security Policy Manager enforcement points are provided in situations where the resource itself does not have the capability to interpret standards-based security policies.

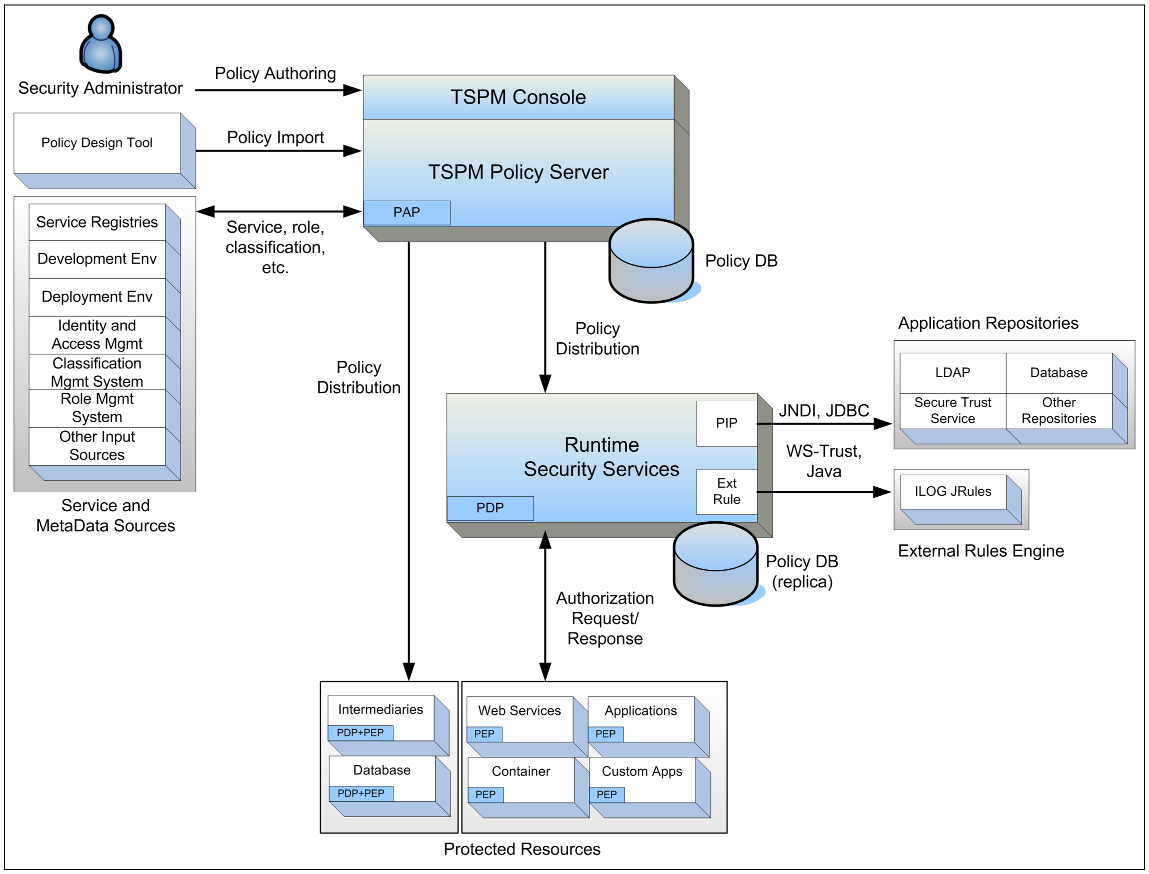


Figure 3-3 High-level architecture of Tivoli Security Policy Manager

The core of Tivoli Security Policy Manager consists of a policy administration framework and user interface, a policy repository to store policy data and metadata, and a security and delegation framework. Plug-ins of several different types, which exist on top of this framework, enable additional capabilities of Tivoli Security Policy Manager.

Tivoli Security Policy Manager uses the Java persistence object layer to work with the underlying database. It supports persistence with major relational database management systems (RDBMS) on the market today.

Figure 3-4 shows the various logical components of Tivoli Security Policy Manager.

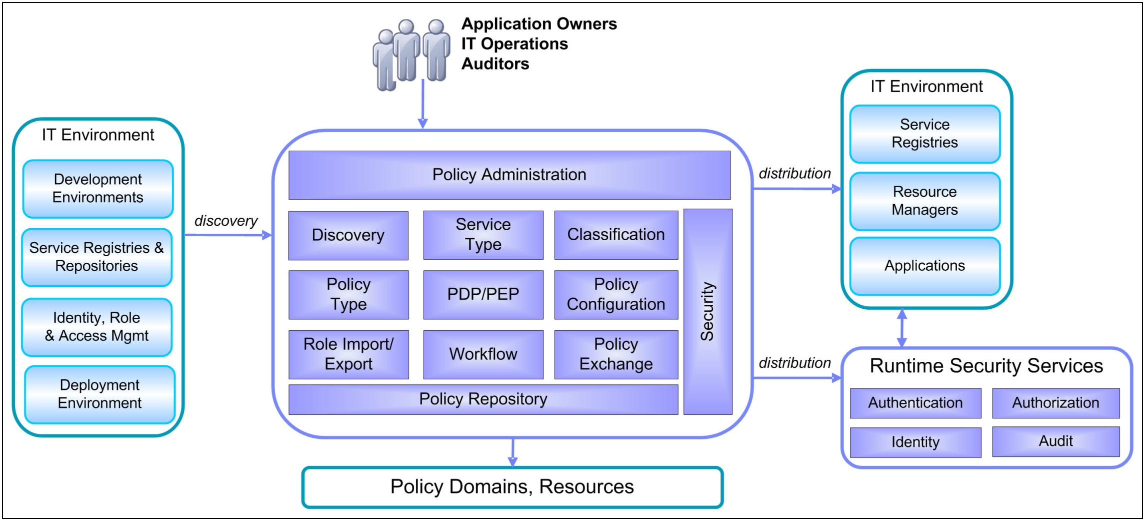


Figure 3-4 The logical components of Tivoli Security Policy Manager

The policy administration framework is based on OSGi, with extension points defined that allow the addition of service, policy, policy domain, policy decision and enforcement points, policy distribution target, and other capabilities of a policy administration point.

Most notably, the *service type* and *policy type* extension points define and manage the structure and semantics of instances of those kinds of services and policies. For example, the *web service* service type plug-in manages the processing of an imported WSDL[4] to define an instance of a web service, as well as any user interface pages specific to the contents of a web service.

[4] WSDL: Web Services Description Language. Refer to `http://www.w3.org/TR/wsdl` for more information about this topic.

Similarly, the *authorization* policy type plug-in manages the user interaction of authoring an authorization policy, as well as the import, calculation, and generation of the XACML authorization policy created and distributed by Tivoli Security Policy Manager.

The *discovery* capability employs service definition and other registry plug-ins to query the IT environment for service data and any associated metadata.

The *distribution* capability provides the standardized (WS-Notification and WS-MetadataExchange) secure policy exchange of notification and policy retrieval, while also allowing plug-ins to provide custom distribution logic, for example, to distribute policy to custom policy distribution targets using its available APIs.

Finally, you can deploy the Tivoli runtime security services co-located with the policy server or throughout the IT environment to provide a number of policy distribution points.

3.2.2 Policy server architecture

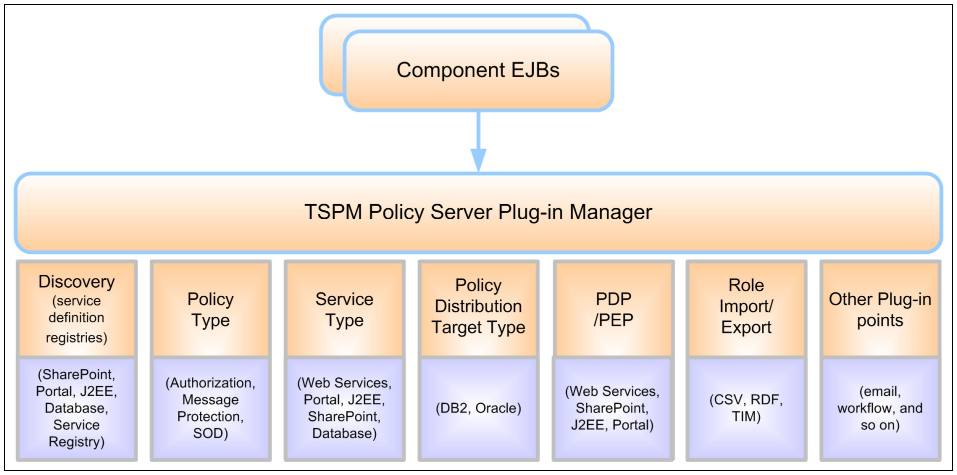
A Tivoli Security Policy Manager policy server is the central component for IT security policy administration, storage, and distribution. It is designed to be extensible in order to support different policies and policy distribution targets.

Policy administration framework plug-in architecture

Tivoli Security Policy Manager needs to support multiple types of artifacts, which include services, resources, policy types, service endpoints, policy distribution targets, service definition, metadata registries, and so on. The policy server consists of a security policy administration framework and the plug-ins that provide security policy management functions. The underlying flexibility of the security policy administration framework allows the product to be extended to support any new type of service to be protected, type of policy to be authored, type of registry to be queried, type of target to receive policy distributions, and so on. The security policy administration framework exposes a collection of extension points for extending functions, and can be used by other IBM products, customers, and Business Partners. It also allows of each of these types of plug-ins to contribute user interface panels to fully express their semantics and configuration requirements.

The Tivoli security policy administration framework is built on the eclipse Equinox OSGi runtime and extension registry. Tivoli security policy administration framework plug-ins are developed using Eclipse IDE V3.3.

Figure 3-5 shows this Tivoli security policy administration framework plug-in architecture.



Component EJBs
TSPM Policy Server Plug-in Manager
Discovery (service definition registries)
(SharePoint, Portal, J2EE, Database, Service Registry)
Policy Type
(Authorization, Message Protection, SOD)
Service Type
(Web Services, Portal, J2EE, SharePoint, Database)
Policy Distribution Target Type
(DB2, Oracle)
PDP /PEP
(Web Services, SharePoint, J2EE, Portal)
Role Import/ Export
(CSV, RDF, TIM)
Other Plug-in points
(email, workflow, and so on)

Figure 3-5 Tivoli security policy administration framework plug-in architecture

Tivoli security policy administration framework plug-in architecture defines a rich set of service provider interfaces and implementations. These service provider interfaces are used by Tivoli Security Policy Manager itself, and can also be used by customers and third-party vendors to develop their own modules to extend the functions of Tivoli Security Policy Manager.

The following pluggable functions are currently available for Tivoli Security Policy Manager:

- Discovery (service definition registries)

  Service definition registries are the existing repositories from which service definitions can be retrieved. Examples are WebSphere Service Registry and Repository, Database, WebSphere Application Server, and so on.

- Policy type

  Policy standards and languages that are authored by an administrator, applied to services, and enforced by policy enforcements points. Examples are authorization policy, message protection policy, separation of duty (SOD), and so on.

- Services and resources type

  Services and resources are the objects in Tivoli Security Policy Manager to which policies can be applied. Examples are web services, customer applications, Microsoft SharePoint, portal resources, and so on.

- Policy distribution targets

  Policy distribution targets receive policies from Tivoli Security Policy Manager policy server directly, using either “push” or “pull” methods, depending on the nature of the target and network environment. By default, the policy distribution targets “pull” the latest policies from the policy server, and some types of the policy distribution targets can accept policy update “push” from the policy server. Examples are registries such as WebSphere Service Registry and Repository.

- Policy decision points (PDPs) and policy enforcement points (PEPs)

  PDPs make authorization decisions for PEPs based on the policies defined from the policy administration point (PAP). PEPs are the components that perform the access control functions. Examples are Datapower (it can act as both PDP and PEP, or PEP only), Enterprise Service Bus (acts as a PEP), and WebSphere Application Server (acts as a PEP). New PDPs and PEPs can be defined through this plug-in point.

  The policy configuration data is specific to the PEP. It is also part of the definitions of a new type of PEPs.

  Typically, a PDP is also a policy distribution target.

- Role import and export

  Import and export roles from or to various formats such as Comma-Separated Values (CSV), Resource Description Framework[5] (RDF), and so on.

- Other plug-in points

  In addition to above pluggable functions, there could be other pluggable functions that work with Tivoli Security Policy Manager to support new functions such as workflow, and so on.

It is worth noting that different Tivoli security policy administration platform plug-in service provider interface (SPI) extension points have different requirements for the implementation of different Java interfaces. For example, a service type plug-in requires implementation of a service type class, a service factory class, a UI provider class, and a point class for each type of element within the service structure. For a policy type plug-in, a policy type class, a policy factory class, policy authoring class, a UI provider class, and, optionally, a configuration class need to be implemented. A service definition registry plug-in requires implementation of four Java interfaces: a registry type class, a registry factory class, a registry client class, and a UI provider class.

[5] To find more information about the Resource Description Framework, go to the following address: http://www.w3.org/RDF/

Details about how to develop a custom plug-in can be found in the Tivoli Security Policy Manager wiki.[6]

Policy server management APIs

Tivoli Security Policy Manager policy server also provides a set of management APIs to manage policy artifacts, which are used by the console and can also be used by third-party applications or integrated with customer policy management systems.

The management APIs cover the following areas:

- Object creation, list, modification, and deletion.

  Tivoli Security Policy Manager objects include services, policies, classifications, distribution mappings, roles, rule parameters, and so on. These APIs handle object management.

- Services, policies, classifications, and roles import and export.

  These APIs handle the exchange of service information, policy information, and classification and role hierarchies between developer tools and portfolio products.

- Status and reportable data manipulation.

  These APIs handle the status information, such as current state of objects, and historical data, such as policy distribution history.

3.2.3 Tivoli runtime security service architecture

The RTSS is a Java application that provides the runtime capabilities for applications to evaluate policies produced by Tivoli Security Policy Manager. There are two valid patterns of RTSS deployment: server and client in remote mode (referred to as remote client), and client in local mode (referred to as local client). The RTSS server maintains a copy of the effective policy that is distributed to it from the policy server and handles remote client authorization queries using the XACML/SOAP protocol.

[6] Visit the Tivoli Security Policy Manager wiki at the following address:
`https://www.ibm.com/developerworks/mydeveloperworks/wikis/home?lang=en#/wiki/Tivoli%20Security%20Policy%20Manager/page/Home`

An RTSS local client also maintains a copy of the effective policy distributed from the policy server. The local client operates completely independent and does not require an RTSS server. Figure 3-6 shows the two valid patterns for RTSS deployment.

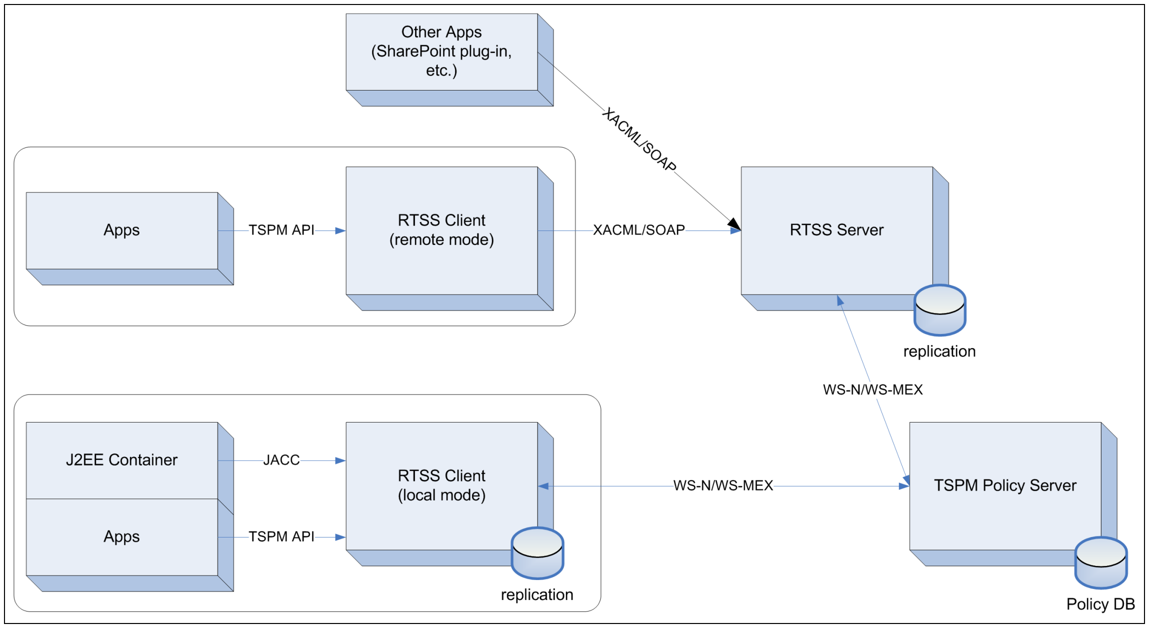


Figure 3-6 RTSS deployment patterns

RTSS server and remote client

Figure 3-7 shows the component architecture of an RTSS server and a remote client, and the interfaces that the RTSS server and client supports.

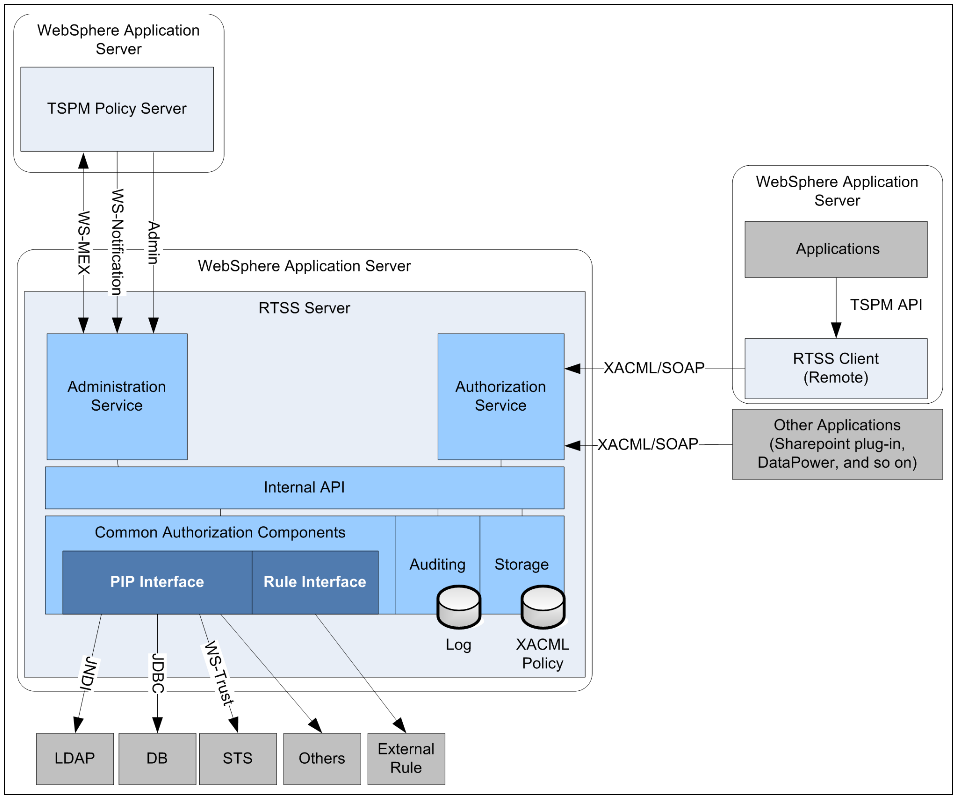


Figure 3-7 Tivoli runtime security service architecture (server and remote client)

RTSS server

An RTSS server runs inside WebSphere Application Server, and consists of an administration service, authorization service, common authorization components, which include a PIP interface and a rules interface, auditing service, and policy storage service, and so on. An RTSS server is used as a PDP, and multiple RTSS servers can be deployed and clustered to improve availability and performance.

The RTSS administration service handles remote administration and policy exchange with the Tivoli Security Policy Manager policy server. After an RTSS server is registered with a Tivoli Security Policy Manager policy server, the policy server can perform remote administration tasks on the RTSS server.

RTSS server policy update and exchange follows WS-Notification and WS-MetadataExchange protocols. Each RTSS server is registered as a policy distribution target (PDT) in the Tivoli Security Policy Manager policy server, and it subscribes to the policies related to its local services. Whenever a policy subscribed by the RTSS server has been updated, the policy server will send a notification to the RTSS server using the WS-Notification protocol. The RTSS server then retrieves the updated policy from the policy server using the WS-MetaDataExchange protocol and replicates it locally. The storage service maintains the local replica of the XACML policy.

The RTSS server provides an authorization services interface for RTSS remote clients and applications that support the XACML over SOAP protocol directly. For example, a .NET application can use the XACML over SOAP interface for getting authorization decisions. One RTSS server can support multiple RTSS remote clients.

RTSS common authorization components provide a PIP interface and a rule interface. The policy information and external rules engine can be integrated into the authorization check at run time. RTSS supports the use of information from existing identity management systems, identity and attribute repositories, and rules engines when evaluating authorization policy. The context of the policy information can be based on identity, service, environment, and business. Currently RTSS provides integrations with context providers through JNDI for access to user repositories, JDBC for access to databases, WS-Trust for access to Trust Services, and Java interface for generic purposes. The rule interface can be used to integrate with external rules engines such as IBM WebSphere ILOG® JRules.

RTSS remote client

The RTSS remote client is a lightweight proxy to an RTSS server. As such, it does not maintain a local copy of the effective policy; instead, it forwards all authorization queries to an RTSS server. The RTSS remote client provides an API for applications to build their own PEPs. This API, called the *Tivoli Security Policy Manager Authorization API*, extends the JACC standard to allow application-level context to be used to make decisions.

In addition to the RTSS client for the WebSphere platform that is shipped with the product, there can be additional RTSS clients or plug-ins for different platforms, for example, an RTSS plug-in for the .NET platform.

RTSS local client

The Tivoli Security Policy Manager RTSS local client is most similar to the RTSS server, except that it is deployed as part of a WebSphere Application Server instance. As with the RTSS server, the RTSS local client maintains its own local copy of the effective policy from Tivoli Security Policy Manager. It has a local authorization service that only serves local applications, and does not provide a remote authorization service for other RTSS clients or external applications. A Java Authorization Contract for Containers (JACC) provider is provided as a part of its local authorization service to support container based authorization. A JavaEE application can also use the Tivoli Security Policy Manager Authorization API to obtain authorization decisions from the RTSS local authorization service directly. Figure 3-8 shows the RTSS local client architecture.

The RTSS local client is also registered as a policy distribution target (PDT) with the Tivoli Security Policy Manager policy server. The policy exchange procedure between the policy server and the RTSS local client is the same as with the RTSS server; it only retrieves the policies that apply to its local services. The “local mode” here indicates that the client replicates the related XACML policies locally and that it makes authorization decisions based upon those policies.

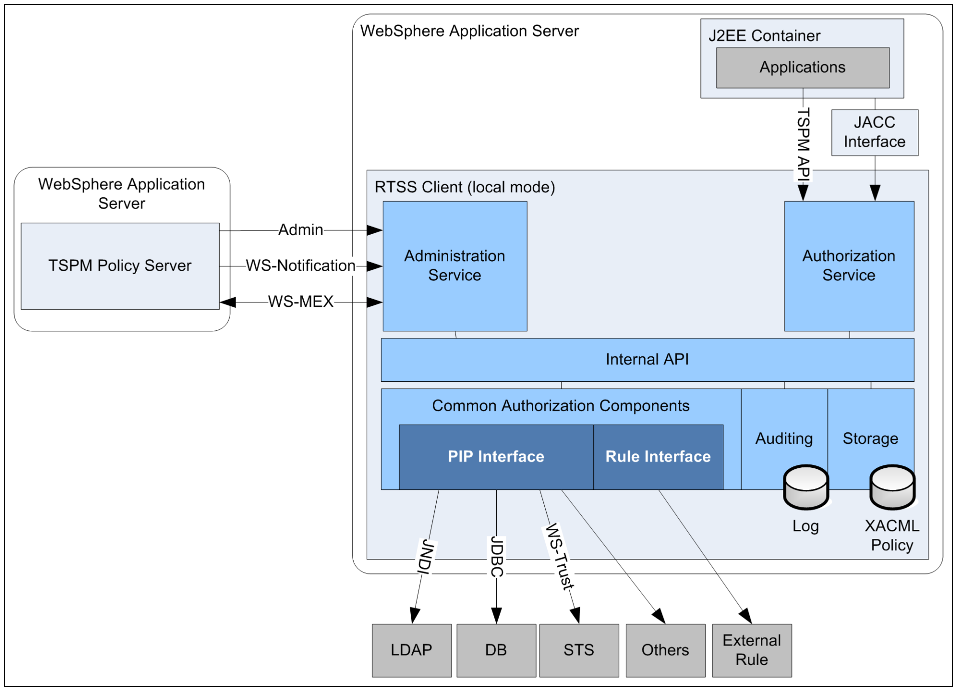


Figure 3-8 RTSS local client architecture

3.2.4 Policy data model, repository, and exchange

Tivoli Security Policy Manager has a dynamic and ontology based data model. It uses the Java persistent data architecture and can be extended by exploiters. Tivoli Security Policy Manager uses an additional layer to abstract the use of a database so it can work with major RDBMSs on the market. Tivoli Security Policy Manager works with Apache Derby, and has been tested with IBM DB2 as well. You can choose several databases from major vendors for policy storage.

The Tivoli Security Policy Manager policy server maintains the policy database, and distributes *calculated effective policy* to the policy distribution targets. The policy distribution targets can be an RTSS server, RTSS local client, WebSphere DataPower, WebSphere Service Registry and Repository, and other systems. The policy distribution targets must be registered with the Tivoli Security Policy Manager policy server before they can participate in the policy exchange. The purpose of the registration is to establish a public key based mutual trusted relationship and a secure communication between the policy server and the policy distribution targets. The policy distribution target registration is secured through a certificate exchange and unique identity assigned to each target. The policy server acts as a Certificate Authority (CA) and issues public key certificates for each policy distribution target. The *tspmRegisterRTSS* tool is used for the RTSS policy distribution target initial registration, and the *tspmRegisterPDT* tool is used for other policy distribution target's (such as WebSphere DataPower) initial registration.

When a policy has been updated on the Tivoli Security Policy Manager policy server, a *policy update notification* is sent out to the policy distribution targets. The actual policy exchange messages are signed with the certificate created during the registration process, and all policies are sent over a secure communication channel.

The policy distribution mechanism is *pluggable*, but WS-Notification and WS-MetaDataExchange protocols are the preferred approaches. Figure 3-9 shows the procedure of the policy distribution target registration and policy exchange with related web services standards that can be used.

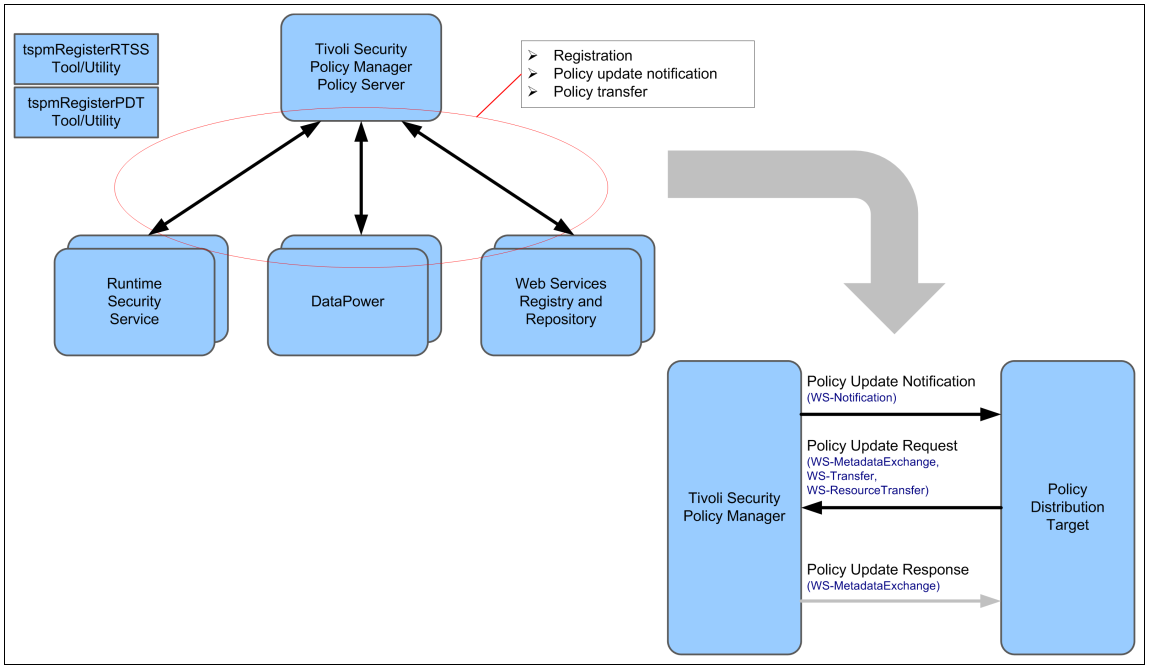

tspmRegisterRTSS Tool/Utility
tspmRegisterPDT Tool/Utility
Tivoli Security Policy Manager Policy Server
Registration
Policy update notification
Policy transfer
Runtime Security Service
DataPower
Web Services Registry and Repository
Tivoli Security Policy Manager
Policy Update Notification (WS-Notification)
Policy Update Request (WS-MetadataExchange, WS-Transfer, WS-ResourceTransfer)
Policy Update Response (WS-MetadataExchange)
Policy Distribution Target

Figure 3-9 IT Security policy data exchange

3.2.5 Policy administration and classification

From a security policy administrator's point of view, the process to administrate a new IT security policy involves the following steps in Tivoli Security Policy Manager:

1. Add a service definition in the policy server.
2. Define a policy.
3. Attach the policy to a service.
4. Configure the policy.
5. Distribute the policy.

Managing the security of large-scale SOA environments is a key capability of Tivoli Security Policy Manager. Classifications can reduce the administrative impact by allowing common security requirements to be configured once and applied to multiple services, ensuring that cross-cutting security requirements are applied consistently. Any service-specific policy is then combined with the classification's policy to form the effective policy for the service. A service can be included in multiple classifications.

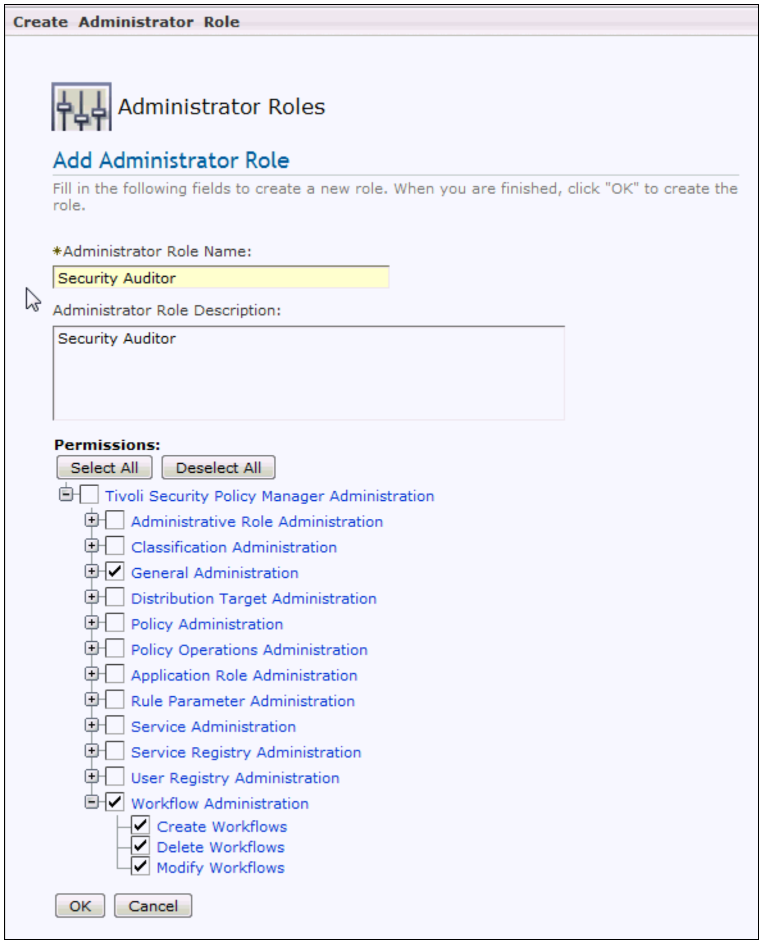
For example, a *confidential information* classification can be created that contains both a message protection policy specifying encryption, and an authorization policy containing certain role membership requirements. This classification can then be attached to any service deemed to be using confidential information.

3.2.6 Delegated administrative security

Tivoli Security Policy Manager provides a powerful delegated administration security framework. Administrators can delegate responsibility for certain tasks to specific users. For example, a line-of-business owner can author an attribute-based policy but then delegate the actual binding of that field to a concrete LDAP attribute to the IT operations team.

This policy management administration framework allows the appropriate individuals within the organization to handle the most appropriate tasks for their job role in a secure manner. A Tivoli Security Policy Manager administrator can define additional administrative roles in the policy management console.

Figure 3-10 shows how a new sample *security auditor* role is defined with appropriate privileges.


Create Administrator Role

Administrator Roles

Add Administrator Role

Fill in the following fields to create a new role. When you are finished, click "OK" to create the role.

*Administrator Role Name: Security Auditor

Administrator Role Description: Security Auditor

Permissions:

- [ ] Tivoli Security Policy Manager Administration
  - [ ] Administrative Role Administration
  - [ ] Classification Administration
  - [✓] General Administration
  - [ ] Distribution Target Administration
  - [ ] Policy Administration
  - [ ] Policy Operations Administration
  - [ ] Application Role Administration
  - [ ] Rule Parameter Administration
  - [ ] Service Administration
  - [ ] Service Registry Administration
  - [ ] User Registry Administration
  - [✓] Workflow Administration
    - [✓] Create Workflows
    - [✓] Delete Workflows
    - [✓] Modify Workflows

Figure 3-10 Administrative role definition using Tivoli Security Policy Manager console

Application owners do not need to know specific environment details (such as the IP address and bind credentials of an LDAP server). However, the IT operations team does not need to define an IT security policy, but simply ensure that it is implemented effectively.

After an IT security policy is authored, it often requires enforcement of additional criteria for the policy by the policy enforcement points. Tivoli Security Policy Manager makes an important distinction between policy authoring and policy configuration, primarily based on the roles within your organization. Fundamentally, *policy authoring* refers to the user interaction that results in a policy that is standards-compliant and inter-operable but might or might not contain sufficient implementation detail to enforce on an enforcement point platform. *Policy configuration* refers to the additional level of detail that is required to augment the policy and to make it enforceable, potentially binding platform-specific data based on the policy's ultimate target platform.

In most use cases, the policy configuration data is not specific to any PDP. For some use cases, however, configuration data is dependent specifically on the intended decision point. For example, WebSphere DataPower may require the presence of a summary element at the top of the message protection policy document that describes the policy domain for the given policy. Different PEPs will often have different policy configuration requirements.

Every aspect of the Tivoli Security Policy Manager management server is built on a pluggable framework based on OSGi run time. Business Partners, OEM vendors, and consultants can build custom solutions based on Tivoli Security Policy Manager to provide unique value to their clients.

3.2.7 Auditing and reporting

Tivoli Security Policy Manager provides auditing capabilities on both policy server and runtime security services. According to customer requirements, auditing can be enabled or disabled. The auditing record is based on Common Base Event[7] format, and stored in multiple stores: policy server, RTSS, and policy enforcement points.

The Tivoli Security Policy Manager built-in reporting uses the Tivoli Common Reporting (TCR) tool. TCR is based on the Eclipse BIRT project, and organizations can create their own reports. Tivoli Security Policy Manager reporting requires a full database product (for example, IBM DB2) as the back end, and provides some basic reports.

[7] To learn more about the Common Base Events Specification, go to the following address: http://www.ibm.com/developerworks/autonomic/books/fpy0mst.htm#HDRAPPA

Tivoli Security Policy Manager can be integrated with Tivoli Security Information and Event Manager to provide advanced reporting functions. Tivoli Security Information and Event Manager can collect Tivoli Security Policy Manager auditing records in a secure and reliable way from the policy server, RTSS, and policy enforcement points; the audit logs are stored in an efficiently compressed depot. Tivoli Security Information and Event Manager performs data normalization before it attempts to generate any reports. The data normalization is based on a patented methodology; it breaks the audit record into Who, What, on What, When, Where, from Where, and Where to aspects. The Tivoli Security Information and Event Manager analysis and reporting is based on the normalized data rather than the raw log from different systems. Built-in compliance modules support reporting for government regulations and industry standards such as Sarbanes Oxley (SOX), Health Insurance Portability and Accountability Act (HIPPA), ISO 17799, PCI-DSS, and so on.

Note: For more product information about Tivoli Security Information and Event Manager, refer to the following address:

`http://www.ibm.com/software/tivoli/products/security-info-event-mgr/index.html`

3.3 Example deployment physical architecture

In an IT production environment, reliability, availability, and scalability play an important part of the service level agreement (SLA). Major Tivoli Security Policy Manager components, including the policy server, RTSS server, and RTSS client, are built on the J2EE platform; they can use the WebSphere Application Server clustering technology for reliability, availability, and scalability. The Tivoli Security Policy Manager console is the GUI administration tool; multiple consoles can be set up for failover purposes.

Figure 3-11 shows an example deployment architecture for Tivoli Security Policy Manager.

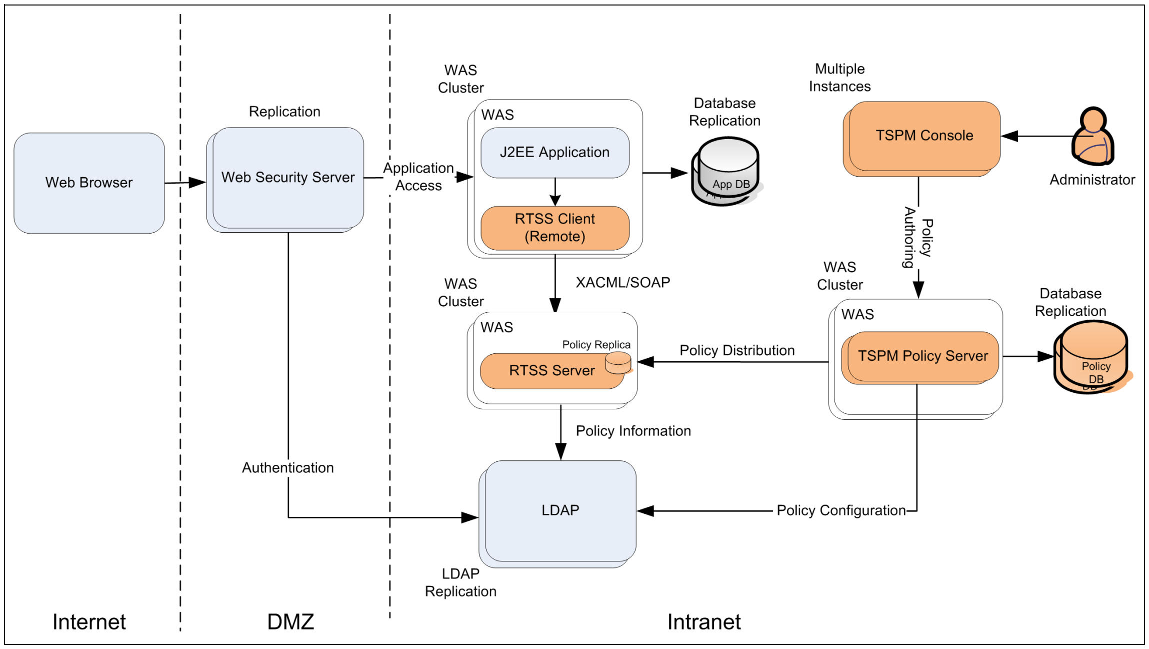


Figure 3-11   Example physical architecture

The Tivoli Security Policy Manager policy server can rely on database technologies such as DB2 high availability disaster recovery (HADR) to achieve policy database high availability and performance. The policy server is deployed within a WebSphere Application Server clustering environment, and thus can provide failover and high availability for policy authoring, management, and distribution tasks. One thing worth noting is that when you define the policies, you want to place the relatively stable entitlements policies based on roles and resources into the policy database; policy information that needs to be evaluated at run time, user attributes, environment variables, and rules should be handled by the runtime security services through the PIP and external rule interfaces. A policy that is designed this way can help simplify the management of the Tivoli Security Policy Manager policy server, reduce the size of the policy database, and thus improve the overall system performance for both policy server and runtime security services.

A Tivoli Security Policy Manager *RTSS server* and *local client* do not contain full replicas of the policy database, but receive the effective policy for an administrator-defined subset of the services, which avoids replicating unnecessary information in the IT environment. A Tivoli Security Policy Manager *remote client* does not replicate any policy information, but instead requests policy information at run time. Multiple RTSS servers can be used to improve the scalability of the system, because each server can support multiple RTSS clients; this is especially useful in a segmented network environment or a network environment spanned across WANs.

In Figure 3-11 on page 89, the J2EE application uses the Tivoli Security Policy Manager entitlements service. Tivoli Security Policy Manager policy servers, RTSS servers, and RTSS clients are all deployed in WebSphere Application Server clustering environments. For illustration purpose, we deploy the RTSS remote client in the WebSphere Application Server cluster, and it queries the authorization decisions from the RTSS server through the XACML/SOAP protocol. The Tivoli Security Policy Manager policy server distributes the necessary policy subsets to the RTSS server. The RTSS server queries user attributes and other information from the replicated LDAP server using the PIP interface.

For simplicity, we did not place components such as network load balancer or IBM HTTP servers into Figure 3-11 on page 89. For more detailed information, to WebSphere Application Server, DB2, and Tivoli Directory Server product documentation about how to set up cluster and replication configurations.

Changing into a local mode client deployment: The RTSS local mode client architecture will be similar to Figure 3-11 with a single box representing the RTSS client in local mode, as opposed to separate boxes for RTSS client (remote) and RTSS server, with the policy being replicated to this single box from the Tivoli Security Policy Manager policy server. Figure 3-8 on page 82 depicts this general configuration.

There are other factors that need to be considered when designing the Tivoli Security Policy Manager deployment architecture. For example, for applications that require a large number of authorization queries during a short period of time, an RTSS remote client is impractical due to performance degradation. For container based authorization requests in a WebSphere Application Server environment, currently only RTSS local client can be used.

Note: Refer to Chapter 9, “Deployment considerations” on page 267 for more information about deployment considerations.

3.4 Conclusion

In this chapter, we gave an overview of Tivoli Security Policy Manager, followed by a discussion of Tivoli Security Policy Manager architecture in detail. An example deployment architecture was provided to illustrate how Tivoli Security Policy Manager can be deployed in a production environment to support application level entitlements service.

Tivoli Security Policy Manager is the central piece of the IT security policy life cycle management solution. It is based on open standards and can be customized and extended to meet different requirements, and it is built on WebSphere technology and uses the reliability, availability, and scalability from the WebSphere Application Server infrastructure.

4

Integration with external systems

Tivoli Security Policy Manager provides a wide range of integration points that provides the flexibility to deliver end-to-end management of centralized application and data entitlements and SOA security policy. In this chapter, we discuss some of the commonly used external systems that Tivoli Security Policy Manager integrates with, which include:

- Identity management
- Access management
- Role management
- User repositories
- Trust services
- Application repositories
- Classification management tools
- Compliance management
- Rules engines

4.1 Identity management

Identity and access management governance is driving organizations to seek end-to-end solutions to manage the full life cycle of identities and their associated entitlements. These initiatives extend to how organizations manage their electronic identities, access to IT systems, and processes. Organizations aim to:

- Manage visibility of IT resources critical to business operations.

  Organizations seek to understand what their critical business related IT assets are and who has access to them.

- Control identity aspects of an IT infrastructure.

  After identities and resources are clearly understood, policy based management systems can be put in place to control which identities can have access to which systems under what circumstances.

- Automate identity related processes.

  Manual IT processes are error prone, difficult to control, and hard to effectively audit. Organizations seek to automate identity related processes to improve their ability to govern their critical systems and comply with regulations.

Managing identities and the systems they have access to is a challenging task for today's organizations. Keeping track of which identities have access to which systems is a complex task, especially as an organization grows. To deal with this problem effectively, a structured, automated approach needs to be implemented.

Identity management provides a policy based mechanism to control and monitor provisioned IT entitlements using a centrally defined policy based on roles. An identity management system should allow a business to implement modeled business processes involved in provisioning and ongoing management of entitlements, which allows an organization to bring strong governance to the process of entitlements provisioning.

4.1.1 Integration with identity management

Both identity management and security policy management systems make use of a role based approach. The roles defined in an identity management system are used to drive the provisioning of IT entitlements, whereas roles in a security policy management system are used to drive policy based runtime authorization decisions. There is often some overlap between the role definitions required for the identity management system and the role definitions required for the security policy management system.There may be a one-to-one mapping between the roles in the two systems or business logic may need to be applied to transform the set of roles in an identity management system into an appropriate role structure that can be applied to a security policy management solution.

A tool that can interface with both systems and can apply business logic is required to integrate the two systems, as shown in Figure 4-1.

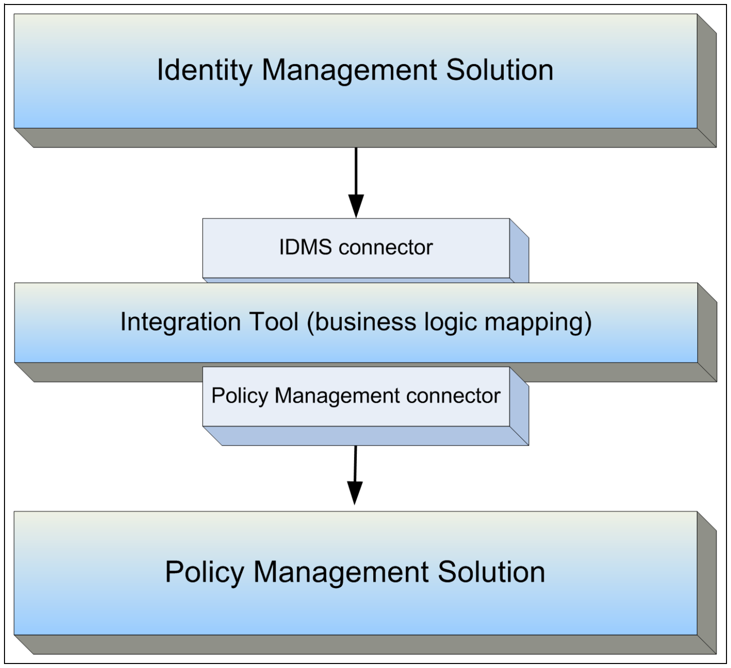

Identity Management Solution
IDMS connector
Integration Tool (business logic mapping)
Policy Management connector
Policy Management Solution

Figure 4-1 Integrating identity management roles with security policy management

The flow of information between the two systems consists of three steps:

1. An identity management systems typically exposes an interface for retrieving data from its repository. An integration tool needs to provide an Identity Management System (IDMS) connector that can connect to the IDMS and extract the required information, which in this case is role metadata.

2. Business logic may need to be applied to the data retrieved from the IDMS, such as mapping a set of roles definitions in the IDMS to a set of roles useful for a security policy management system. This transformation may include using a subset of roles from IDMS, a super-set of the roles from the IDMS, a consolidated set of roles from the IDMS, and so on.
3. A security policy management system typically exposes an interface for adding data to its repository. An integration tool needs to provide a security policy management system connector that can connect to the system and store data in its repository.

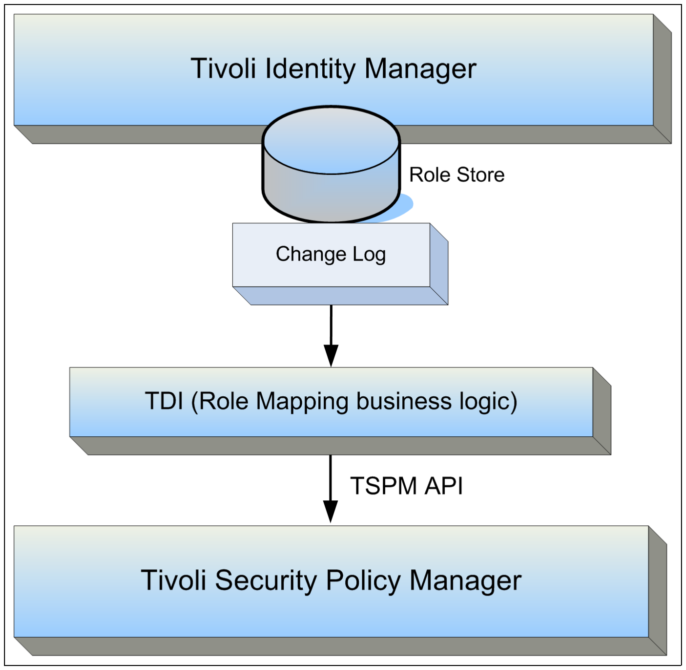
4.1.2 Integration with Tivoli Identity Manager

Tivoli Identity Manager is a market leading and role based provisioning solution that allows an organization to manage their provisioned entitlements in line with a business policy.

The roles defined in Tivoli Identity Manager can be useful when defining policy in Tivoli Security Policy Manager. Consider a scenario where only a subset of roles defined in Tivoli Identity manager are appropriate for use in Tivoli Security Policy Manager. Business logic needs to be applied to define which roles should be exported from Tivoli Identity Manager into Tivoli Security Policy Manager.

IBM Tivoli Directory Integrator enables you to integrate data from different repositories in an easy and flexible way. Tivoli Directory Integrator provides a means of connecting to a range of IT systems to collect data, normalize it and apply logic to transform the data into new forms. It provides a large set of software components known as connectors that allow it to interface with a wide range of IT systems.

Tivoli Directory Integrator could be used to map roles between Tivoli Identity Manager and Tivoli Security Policy Manager, as shown in Figure 4-2.



Tivoli Identity Manager
Role Store
Change Log
TDI (Role Mapping business logic)
TSPM API
Tivoli Security Policy Manager

Figure 4-2 Integrating Tivoli Identity Manager with Tivoli Security Policy Manager for importing roles

The integration works as follows:

1. Tivoli Directory Integrator can interface with Tivoli Identity Manager to listen for updates to Tivoli Identity Manager roles through a change log connector.
2. Business logic can be defined in a Tivoli Directory Integrator AssemblyLine to map Tivoli Identity Manager roles to a set of roles appropriate for Tivoli Security Policy Manager.
3. Tivoli Directory Integrator can interface with Tivoli Security Policy Manager using the API discussed in 8.2, “Policy management API” on page 237 to import the new role definitions.

4.1.3 Integration with other identity management systems

Tivoli Security Policy Manager can use Tivoli Directory Integrator to interface with many different types of external systems. Patterns similar to the one outlined in 4.1.2, “Integration with Tivoli Identity Manager” on page 96 can be used if the identity management vendor provides a public interface that role data can be retrieved from using Tivoli Directory Integrator.

4.2 Access management

The goal of access management is to provide and ensure appropriate access to an organization’s resources. In terms of an organization’s IT resources, it aims to secure access to a set of heterogeneous systems, built on a range of modern and established technology. This task is accomplished typically by offering a set of security services that integrate with an organization’s IT infrastructure. The following areas are key components of access management systems:

- Authentication services

  There are a variety of technology, organizational, and business reasons why different authentication schemes are used throughout an organization. Access management systems typically offer a range of authentication capabilities to support these needs. After an identity is authenticated, identity tokens should be propagated to IT systems within the organization to avoid the cost of unnecessarily implementing authentication services at multiple points within the organization, thus enabling an organization wide single sign-on (SSO) solution.

- Authorization services

  An access management system should allow integration with existing and emerging infrastructure to provide secure and centralized policy management capabilities. Resources need to be discovered and modeled so that policy can be authored against them. This component requires a flexible authorization framework and a range of supported integration points.

- Federated identity services

  Organizations are increasingly looking to use strategic relationships with partners and customers. These relationships are often defined at the business level through contracts and agreements. To facilitate the implementation of these commitments, IT systems may be required to interface with systems outside the traditional scope of their environment. Providing a mechanism to federate identities across organizations is a key component of this strategy.

A system that provides federated identity services should propagate identities between organizations in an standards compliant manner. An important part of federation services is a trust service component that allows the transformation of a range of security tokens into consumable formats.

- Audit services

  Comprehensive auditing capabilities should be provided by an access management system to support identity and access management governance. Auditing systems should be able to be used by compliance systems to generate a holistic record of access to business critical systems.

Traditionally, access management solutions have provided authentication, authorization, federation, and audit services for web and operating system based resources. New standards and technology such as web services, service orientated architecture (SOA), and cloud initiatives are driving new integration points in the access management space. When combined with requirements to implement authorization based on both data-level and context sensitive attributes, new patterns for access management need to be considered.

It is desirable to use existing investments in access management technology to provide true end-to-end and flexible policy management solutions. Authentication and trust services can be used as key components when implementing policy management solutions.

As an example, consider an organization that has an existing investment in web-based access control systems. A reverse proxy server provides authentication services at the perimeter of the network. In this example, consider that the authentication processing involves significant complexity and that it is highly desirable to re-use this investment if possible. The access management system stores identity information in a proprietary authentication token that can be used for single sign-on with other systems within the organization. The identity token contains identity information and attributes about the identity that can be useful for defining policies. To implement a solution that takes advantage of identity information from an existing access management system, a trust service may be required to transform the identity token into a normalized format that the policy management system can consume, extract identity information from and apply policy to. After a normalized identity is established, policy can be authored, distributed, and enforced throughout the environment using attributes from the normalized token.

4.2.1 Integration with Tivoli Access Manager for e-business

Organizations are increasingly using web-based technologies to expose business services to customers, employees, and partners. A rapidly evolving web technology landscape makes securing access to these services difficult. Tivoli Access Manager for e-business (TAMeB) helps organizations improve identity and access management governance by providing authentication, authorization, and single sign-on capabilities for web based resources.

Tivoli Access Manager for e-business provides several components that can implement access control at various points within an organization. A common pattern is to implement a defense in-depth strategy to provide a layered security approach. Access is authorized in the outer most layer of the network, known as the demilitarized zone (DMZ). The Tivoli Access Manager for e-business WebSEAL component provides a reverse proxy solution that can be used to implement authentication, authorization, and single sign-on capabilities in the DMZ. This setup allows users to be authenticated and authorized in line with a centrally defined policy before proceeding to applications residing in more protected areas of the network.

As part of the Tivoli Access Manager for e-business authentication process, a credential is built that can contain detailed information about subjects. The credential artifact is used in the authorization process implemented by WebSEAL and can optionally be used by other downstream components to propagate trusted subject information.

Tivoli Security Policy Manager introduces an integration to allow Tivoli Access Manager for e-business credentials and the attributes they contain to be used in policy management. Figure 4-3 shows the flow of the integration:

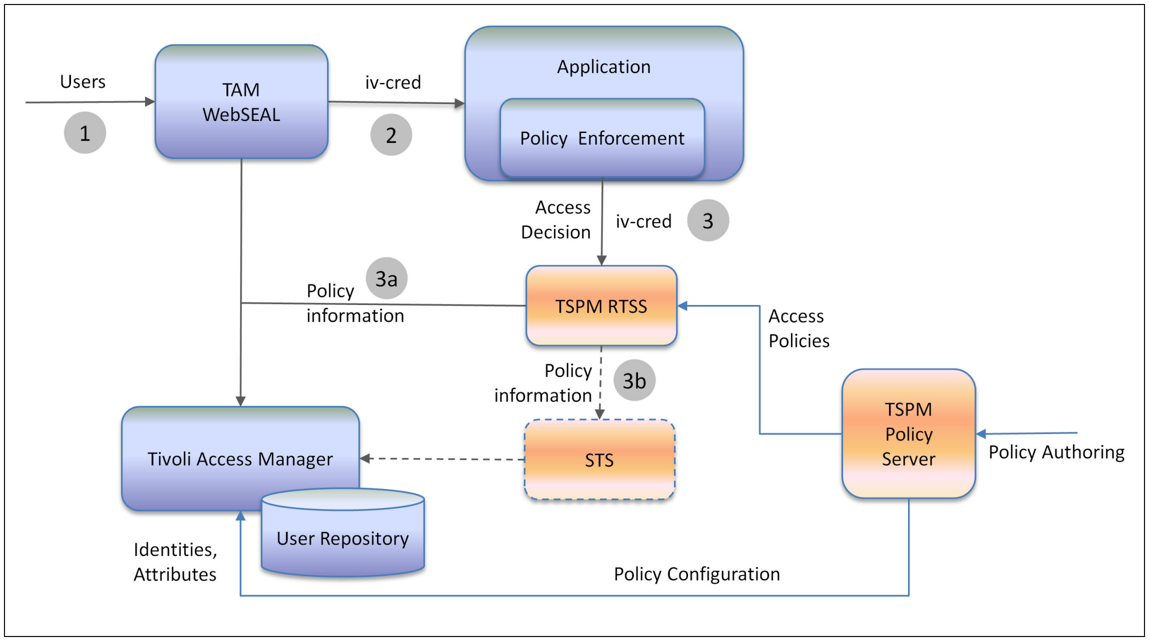


Figure 4-3 Tivoli Access Manager integration with Tivoli Security Policy Manager

The logical flow to use a Tivoli Access Manager for e-business credential to enforce complex policy is as follows:

1. A user requests a web-based resource, protected by Tivoli Access Manager for e-business WebSEAL, which authenticates the user using the configured authentication mechanism. Upon successful authentication, a credential is constructed that contains a list of relevant attributes about the user. This credential is used to authorize the user based on Tivoli Access Manager for e-business' authorization policy.
2. If the user is permitted, the request can be passed to the target web resource. In this scenario, Tivoli Access Manager for e-business WebSEAL can be configured to pass an authentication artifact known as *iv-creds*, which contains identity information.
3. The security token, in this case iv-cred, is passed to the Tivoli Security Policy Manager Runtime Security Service (RTSS) for processing in one of two ways. It can be converted to a collection of XACML subject attributes (3a) or left in its original format (3b) for use in a Tivoli Security Policy Manager access decision.

a. Process the iv-cred security token to create a collection of XACML subject attributes.

   The JACCPlus API extends the Java Authorization Contract for Containers (JACC) API to allow an application to control the context within which the authorization decision is made. It supports parsing an iv-cred token, which can be processed by the JACCPlus API to produce a collection of XACML subject attributes that can be passed to the Tivoli Security Policy Manager policy decision point. The JACCPlus API to pass the iv-cred is discussed in 8.1.1, “Tivoli Security Policy Manager authorization API” on page 214.

   From a policy authoring point of view, a policy must be authored using attributes names from the expected token. When JACCPlus is configured to pass attributes to the RTSS as a collection of XACML subject attributes, a policy should be authored using attribute names as they appear in the Tivoli Access Manager for e-business credential.

b. Process the iv-creds security token as an generic token.

   An extension to the JACCPlus API has been implemented to support passing the iv-cred token directly to the RTSS. To enforce policy, the RTSS requires that the iv-cred token be normalized into a standard format. Tivoli Security Policy Manager introduces a new policy information point (PIP) to manage security tokens called the *Security Trust Service (STS)* PIP. Figure 4-3 on page 101 shows how RTSS makes use of the STS PIP. The PIP uses WS-Trust to communicate with an STS.

   An STS PIP should be configured to use a product that implements a trust service. A trust service provides the ability to convert a number of different security tokens to alternate formats. Tivoli Federated Identity Manager provides a Security Token Service (STS) that can convert the iv-creds token into a format consumable by RTSS for authorization. An RTSS security token PIP should be configured to use the Tivoli Federated Identity Manager Security Token Service (STS) to validate the iv-cred token, map any attributes required, and issue a SAML 1.x token.

   SAML requirements: The STS PIP requires that a Security Assertion Markup Language (SAML) 1.x token be returned. Policy can be authored using the SAML attribute identifiers from the transformed SAML 1.x token.

4.2.2 Integration with other access management systems

Tivoli Security Policy Manager can integrate with virtually any access management system as long as there is a mechanism to propagate identity information. In 4.5, “Trust services” on page 110, we show how this task can be accomplished with a trust service by normalizing authentication tokens from the access management system in question.

4.3 Role management

Organizations often classify users by some aspect of their responsibilities within the organization. These grouping are often based on data from human resources systems such as job category. These grouping can be used for a variety of useful purposes, including administration of business processes, controlling access to systems and data, and so on.

Roles are typically discovered, managed, and rationalized using a role management tool. This is a task in its own right and is not in the scope of a security policy management tool. However, the output of a role management process is useful for importing a role structure that can be used to define IT security policy.

Tivoli Security Policy Manager provides three mechanism for importing role definitions from external role management tools:

- Rule Interchange Format[1] (RIF) file

  Tivoli Security Policy Manager provides a mechanism to input role definitions from an interchange file.

- API

  The Tivoli Security Policy Manager API can be used to programmatically import role definitions.

- Manually

  Roles can be added manually through the Tivoli Security Policy Manager administration console.

[1] To discover more information about the Rule Interchange Format standards, go to the following address: `http://www.w3.org/2005/rules/Overview.html`

4.4 User repositories

Organizations typically use some form of user repository to capture and store identity related information. User repositories store details such as identity definitions, groups membership, and attributes related to an identity. It is important that a security policy management system provides the flexibility to integrate with an organization's user repositories to use this information. Several aspects must be considered in the context of a security policy management solution:

- Role mapping

  When defining policies in a policy management system, role definitions can be defined in abstract terms to allow policies to be modeled at the business level. To transform a policy into a form that IT operations can implement, role definitions should be mapped to real IT resources. User repositories provide the mechanism to map groups of users to roles.

- Policy information point

  A PIP allows a policy management system to retrieve attributes from an external system for use in an authorization decision. As a user repository stores detailed information about identities, it is important that security policy management systems integrate with user repositories to retrieve information for use in authorization operations.

- Role mapping for administration

  A security policy management system requires role definitions to define administrative groups and controls. User repositories often contain the group mappings required for this purpose.

4.4.1 Integration with Tivoli Directory Server

Tivoli Security Policy Manager can integrate with IBM Tivoli Directory Server through the use of the Java Naming and Directory Interface (JNDI) using the Lightweight Directory Access Protocol (LDAP). There are two points where LDAP can be used:

- Role mapping

  Tivoli Security Policy Manager supports the mapping of abstract roles to data stored in a directory as part of the RTSS configuration. Roles are configured by specifying the appropriate LDAP server details and adding LDAP search parameters to define the role mapping.

- Policy information point

  A PIP can provide a source of attributes from an external system for use in an authorization decision. The LDAP PIP allows data to be extracted through an LDAP query. As part of the RTSS configuration, an LDAP query can be specified to define the data required for the authorization decision.

4.4.2 Integration with Tivoli Directory Integrator

A modern IT infrastructure consists of a heterogeneous set of systems using many different interfaces, standards, protocols, and technology. Different lines of business often need to support technologies from different vendors for business reasons. The technology that implements the data layer that underpins these technologies are often not compatible and this can make system integration difficult. Synchronization and integration of data from these repositories is sometimes required to solve complex business and IT problems.

IBM Tivoli Directory Integrator enables you to integrate data from different repositories in an easy and flexible way. There is virtually no limitation on the type of data or system with which Tivoli Directory Integrator is able to work. Tivoli Directory Integrator provides a means of connecting to many different IT systems to collect data, normalize it, and apply logic to transform the data into new forms. It provides a large set of software components known as connectors that allow it to interface with a wide range of IT systems. The key to integrating data collected from these systems is to convert system specific data into a common format that can then be manipulated by logic defined in Tivoli Directory Integrator, which is useful in many scenarios, including consolidating and transforming user repositories and application data.

Tivoli Directory Integrator uses the concept of an *AssemblyLine*. At each stage of the AssemblyLine, work is performed. This can include importing data from systems, transforming data, or exporting data to systems.

Figure 4-4 shows the concept of using an AssemblyLine to process data from multiple sources.

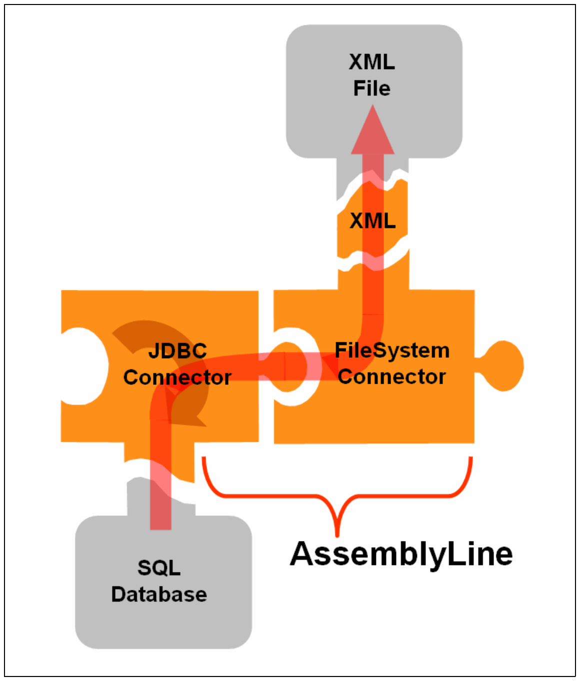


Figure 4-4 An example of a Tivoli Directory Integrator AssemblyLine

There are four integration points where Tivoli Security Policy Manager can benefit from the integration capabilities of Tivoli Directory Integrator:

- Attribute data from external systems

  Tivoli Security Policy Manager provides a PIP interface to retrieve attributes from external sources.

  Tivoli Directory Integrator provides server connectors that allow it to act as a server for a given protocol. For example, it provides an LDAP server connector that allows it to act as an LDAP server. This allows Tivoli Directory Integrator to be used as the following:

  – Target for an LDAP PIP using LDAP JNDI

  – Target for a Java PIP using a Java interface

  Tivoli Directory Integrator can integrate with many different systems and apply logic to manipulate the data it retrieves. Using Tivoli Directory Integrator to implement PIPs provides a policy management solution with considerable flexibility when integrating with existing identity management systems, identity and attribute repositories, application databases, rules engines, and so on.

- Policy distribution targets

  Tivoli Security Policy Manager supports a number of policy distribution targets. As there are many possible integration points; some policy distribution targets may require a custom solution. To support custom policy distribution targets, the pluggable Tivoli Security Policy Management platform can be used by writing a custom policy distribution target plug-in. The interface to implement a custom policy distribution plug-in is discussed in 8.2, “Policy management API” on page 237.

  As Tivoli Directory Integrator provides a large set of connectors that integrate with external systems, it can be useful to use it to implement a custom policy distribution target plug-in.

- Import services

  An important aspect of policy modeling and authoring is the discovery of service definitions. In some cases, it is desirable to automate this process. A custom service registry plug-in can be written to import resources into Tivoli Security Policy Manager. As Tivoli Directory Integrator supports connectivity to a wide range of systems, it can be useful to use its integration capabilities to discover services from service registries and import them into Tivoli Security Policy Manager by implementing a service registry plug-in.

- Role mapping

  During policy configuration, roles defined in Tivoli Security Policy Manager must be mapped to groups of users in external repositories. This mapping is typically defined in a user registry, but there are circumstances where the mapping may be defined in other sources such as a flat file. Tivoli Directory Integrator can provide the flexibility to implement custom logic for role mapping in such cases.

As an example of how Tivoli Directory Integrator can be used to provide integration with external systems, consider the scenario shown in Figure 4-5. It shows how Tivoli Security Policy Manager can be used to extract attribute data from an external system for the purpose of performing an authorization decisions. Tivoli Security Policy Manager is configured to use an LDAP PIP implemented by Tivoli Directory Integrator to extract and consolidate information from externals systems.

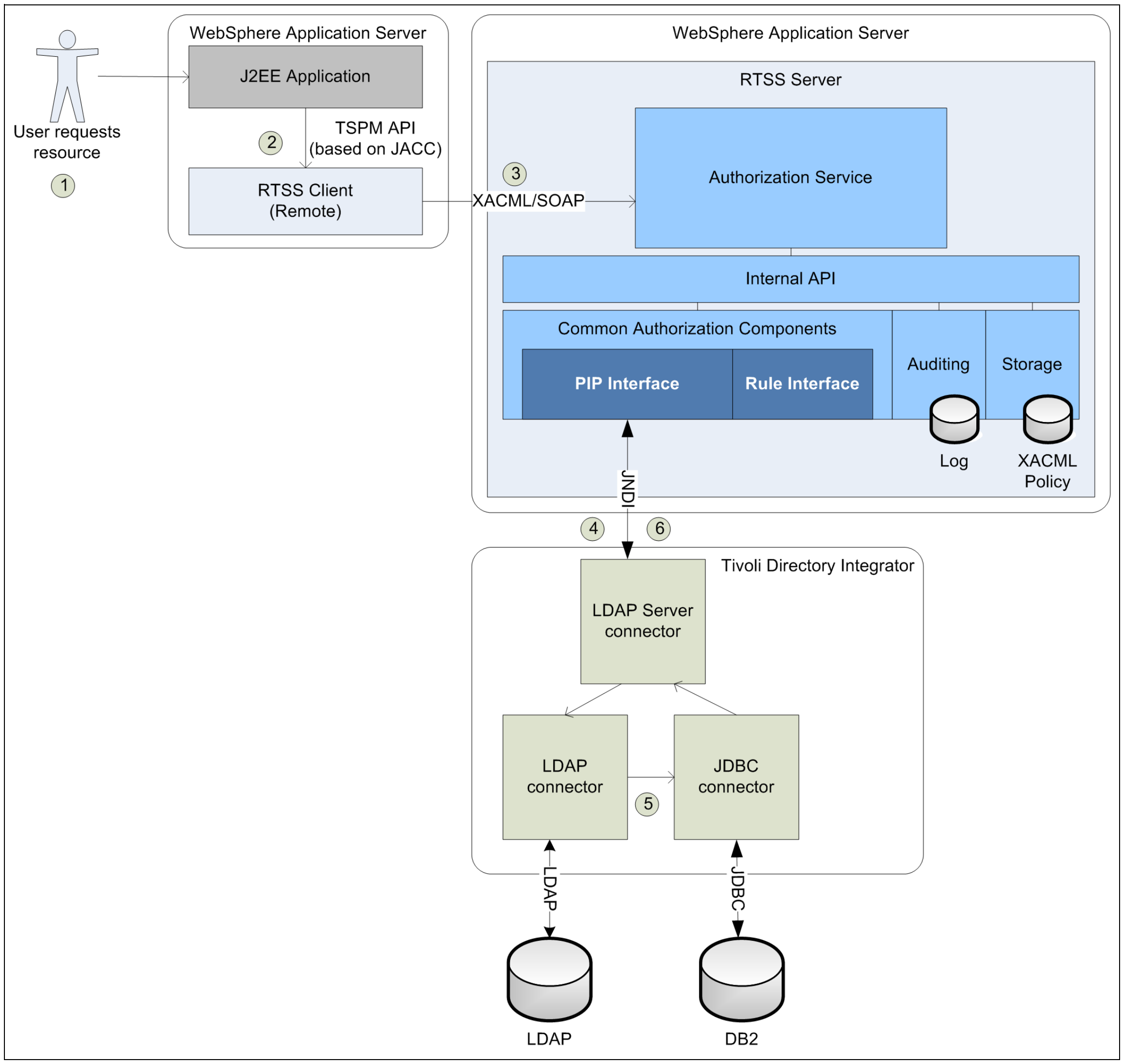


Figure 4-5   LDAP PIP using Tivoli Directory Integrator

The goal is to collect data from two different external systems and transform it into a single attribute required by Tivoli Security Policy Manager. The flow to achieve this process is as follows:

1. A user requests a resource from an application, which uses the JACCPlus API to enforce access control.
2. The application calls the JACCPlus TSPM authorization API. Details about this interface are discussed in 8.1.1, "Tivoli Security Policy Manager authorization API" on page 214.
3. The remote RTSS client makes an authorization request to the RTSS authorization service.
4. The RTSS server is configured to retrieve attributes from an external system using an LDAP PIP. This is set up during the configuration of Tivoli Security Policy Manager. The required configuration is:
   - The details of the target LDAP server
   - An LDAP query to define the data required

   Tivoli Security Policy Manager calls Tivoli Directory Integrator through an LDAP PIP, which uses the Java Naming and Directory Interface (JNDI) to make LDAP queries. In the example above, a single attribute is requested from the LDAP server. This attribute will be used to communicate a result containing composite data from two sources using Tivoli Directory Integrator.
5. A custom Tivoli Directory Integrator AssemblyLine is configured to perform the interaction with external systems. An LDAP Server Connector is used to accept requests from the Tivoli Security Policy Manager LDAP PIP. The AssemblyLine implements logic takes the details of the LDAP query and passes normalized data to the next connector. In this example, information from a directory and database system is retrieved using standard Tivoli Directory Integrator Connectors. The data from these two sources are combined using logic defined in Tivoli Directory Integrator configuration to produce a new composite attribute based on the data.
6. Tivoli Directory Integrator returns a single LDAP attribute containing the consolidated data to Tivoli Security Policy Manager for use in an authorization decision.

4.4.3 Integration with other user repositories

For user registries that do not support LDAP through JNDI, Tivoli Directory Integrator can be used to implement a custom solution, as outlined in 4.4.2, "Integration with Tivoli Directory Integrator" on page 105.

4.5 Trust services

Security tokens allow identity related information to be securely passed within and between organizations. A security token contains a trusted source of identity information about the holder of the token. This information typically includes principal, group membership, and attribute information.

Due to the complexity of modern IT systems, there are many types of security token formats in use. Some of these are older or proprietary formats and some use open standards. Unfortunately, systems rarely understand all token formats. Implementing custom logic in applications to process multiple security token types is expensive and inefficient. To manage security tokens in a cost-effective way, organizations can implement a trust service to deal with the complexity of managing different security token types.

Implementing an IT security policy requires trusted sources of identity information against which a policy is enforced. Security tokens contain the identity information required and subject attributes, which can be useful when making policy based authorization decisions. Trust services can be used where required to simplify the management of security tokens when implementing a security policy management project.

4.5.1 Integration with Tivoli Federated Identity Manager

IBM Tivoli Federated Identity Manager provides a Security Token Service (STS) component that can validate incoming security tokens, transform security tokens, and issue new token formats to consumers. The STS provides the ability to allow validation of tokens to be performed if integrity checking is required. Mapping is done by normalizing all incoming token types into an abstract, token independent format, which allows logic to be easily configured to map attributes into a format required by token consumers. Tokens can be issued in any supported format.

Tivoli Security Policy Manager introduces a new type of PIP that allows calls to a STS for security token validation and mapping.

The STS PIP communicates with the STS using WS-Trust 1.2. To configure Tivoli Federated Identity Manager, an STS trust chain could be configured to:

- Validate the incoming token type using an appropriate module
- Add mapping modules to map any required attributes
- Issue a SAML 1.x token

SAML requirements: The Tivoli Security Policy Manager STS PIP requires a SAML1.x token in the Request Security Token Response (RSTR).

The Tivoli Security Policy Manager STS PIP parses the SAML assertion received from the STS and adds subject attributes for authorization decisions.

The STS PIP can cache subject attributes received from the STS to enhance performance. The caching of subject attributes is controlled by checking assertions for “time valid” if it is available. If “time valid” is not available, the “max time to live” value is used instead. The “max time to live” value is configurable through the Tivoli Security Policy Manager PIP user interface.

4.6 Application repositories

Authorization services may require the ability to query application repositories, such as data bases and user registries, for information on which to base authorization decisions at run time.

Data for authorization decisions can be retrieved from several points in an IT system. Sometimes it makes sense to retrieve the data when a subject is authenticated and propagate the information using a security token such as a SAML assertion. This situation has the advantage of retrieving the data once so that it can be used by multiple systems within the organization. However, if the data in question is dynamic and likely to change within the lifetime of the security token, it may not make sense to perform this action. The alternative is to dynamically retrieve the data at run time during an authorization decision. Tivoli Security Policy Manager provides the PIP interface to retrieve data for authorization decisions at run time, which is potentially a more costly approach, but ensures that the latest version of the data is retrieved.

4.6.1 Databases

Tivoli Security Policy Manager provides Java Database Connectivity (JDBC) integration to support the retrieval of database information for use as policy attributes in authorization decisions. A JDBC PIP is used to retrieve the required data. The JDBC PIP is configured during RTSS configuration. Configuration information includes details of the JDBC data source and an SQL query that specifies what data to return for use in the authorization decision. The PIP interface is discussed in 8.1.4, “Policy information point” on page 233.

4.6.2 User registries

As discussed in 4.4, “User repositories” on page 104, Tivoli Security Policy Manager can integrate with user repositories to retrieve access decision information, which can be useful for basing authorization decisions on information over and above the data that can be established about an identity at authentication time.

4.6.3 Proprietary repositories

Tivoli Security Policy Manager provides the flexibility to integrate with virtually any system. As discussed in 4.4.2, “Integration with Tivoli Directory Integrator” on page 105, Tivoli Directory Integrator can be used to connect to and retrieve data from proprietary systems. For example, a flat file may contain data required to make a Tivoli Security Policy Manager authorization decision. Tivoli Directory Integrator could use a File System Connector to parse the file and provide the data to RTSS through a PIP as access decision information.

4.6.4 Java

Integration points are provided to interface with external systems such as JDBC, LDAP, and STS. These points represent common sources of identity and application data required for authorization decisions that are provided as part of the product. If additional flexibility is required, there are two options:

- Implement custom logic using Tivoli Directory Integrator, as discussed in 4.4.2, “Integration with Tivoli Directory Integrator” on page 105.
- Implement a custom Java PIP.

A Java PIP allows Java code to be executed to retrieve data from virtually any external system required. The systems that can be integrated with are only limited by the Java code that can be written. This situation allows for a high degree of flexibility, but requires custom code to be written and maintained. As an example, a PIP might be required to retrieve data from a web service for an authorization decision. A custom Java class could be written and registered as a Java PIP to retrieve information from the web service. The details of the PIP interface are discussed in 8.1.4, “Policy information point” on page 233.

4.7 Classification management tools

Data classification is the process of identifying common sets of entities and grouping them together using logical labels, which provides a way of rationalizing the vast amount of data into logical groups so that business processes can be defined in terms of these groupings. Data classification is a task in its own right, and is a process that is performed in many parts of an enterprise.

In the context of implementing a security policy management solution, a classification process is useful for defining groups of resources, subject, and policies to simply the management of these entities. A security policy management solution needs to be flexible enough to integrate with external classification tools to allow the importation of classification definitions.

There are a wide range of tools available to classify resources. Tivoli Security Policy Manager provides a separate policy design tool that allows architects to import resource definitions, define policies, model and classify this data, and export a policy template that includes classifications of services and policies. The exported policy can be used by Tivoli Security Policy Manager to establish classification for use during policy authoring.

Tivoli Security Policy Manager provides three mechanisms for integrating with external classification systems:

- Rule Interchange Format (RIF) file

  Tivoli Security Policy Manager provides a mechanism to input classification definitions from an interchange file.

- API

  The Tivoli Security Policy Manager API can be used to programmatically import classification definitions.

- Manually

  Classification definitions can be added manually through the Tivoli Security Policy Manager administration console.

4.8 Compliance management

Compliance management is the process of ensuring that an organization operates in accordance with expectations. Organizations define IT security policies to ensure that IT systems are being operated according to all applicable laws, regulations and ethical standards, such as:

- Sarbanes-Oxley
- Basel II
- Food and Drug Administration (FDA)
- NERC-CIP
- Health Insurance Portability and Accountability Act (HIPPA)
- Gramm-Leach-Bliley Act (GLBA)
- Payment Card Industry Data Security Standard (PCI DSS)
- ISO 27001 / 27002

IT security compliance is the process that safeguards the operations of an organization to meet the requirements expressed in IT security policies. A system that manages an IT security compliance process requires the ability to identify compliance criteria and to assess, analyze, consolidate, and to report on the compliance status of security controls. An IT security compliance management system can be helpful in many situations, for example, to create audit reports, to prevent or to clarify IT security incidents, or just to gather evidences for future compliance activities.

A Security Information and Event Management (SIEM) system helps an organization address IT security compliance management problems, such as the collection, analysis, and archiving of audit data from many different computing solutions across an organization. Organizations face three major challenges:

- Demonstrating compliance to regulatory requirements
- Ensuring appropriate protection of intellectual capital and privacy information
- Being able to manage security operations securely and effectively

An SIEM system should be able to collect data from log files and alerts from a variety of infrastructure components such as firewalls, routers, antivirus systems, servers, and many others. It can inform IT teams about unusual behavior on these systems, and then these teams can decide what kind of action to take.

An SIEM architecture can be broken down into two elements:

- Security Information Management (SIM)

  SIM provides reporting and analysis of data primarily from host systems and applications, and secondarily from security devices to support regulatory compliance initiatives, internal threat management, and security policy compliance management. It can be used to support the activities of the IT security, internal audit, and compliance organizations.

- Security Event Management (SEM)

  The SEM component improves security incident response capabilities. It processes near-real-time data from security devices, network devices, and systems to provide near real-time event management for security operations. It helps IT security operations personnel be more effective in responding to external and internal threats.

An SIEM system provides complementary features to those provided by a security policy management system and completes the policy life cycle discussed in earlier chapters. Policy management enforces an IT security policy in real time to control access to resources in line with a centralized security policy. An SIEM system manages the reliable collection of log data from systems of interest, analysis and reporting of archived log data for regulatory compliance, and provides real-time event information for threat management.

To facilitate integration with an SIEM system, individual systems need to produce security related events in a format that an SIEM system can consume. A key aspect of an SIEM system is the capability to understand and consume data from systems of interest. This data is normalized into a common format so that analysis across a range of systems can be performed.

4.9 Rules engines

Business Rules Management Systems (BRMS) implement decision logic that allows an organization to define, manage, and enforce a set of business rules required by the business. They typically provide a mechanism to express rules at the business level and a rules engine to implement the business logic in the context of the IT environment. Business rules management systems are used to define and implement re-usable rules that can be used by systems across the organization to simplify the application environment. A rules engine might be used to implement a fraud detection system that can be used by multiple systems within the business.

When implementing a security policy management system, it is useful to use existing investment in BRMS. Tivoli Security Policy Manager provides the RTSS client external rules interface for interfacing with external systems such as rules engines, as shown in Figure 4-6.

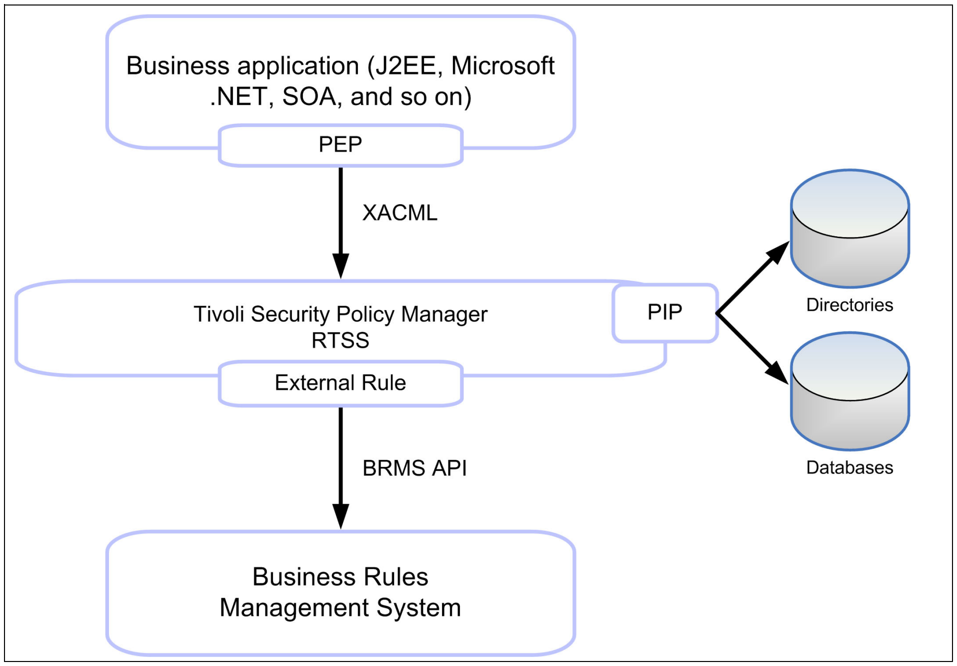

Business application (J2EE, Microsoft .NET, SOA, and so on)
PEP
XACML
Tivoli Security Policy Manager RTSS
PIP
External Rule
BRMS API
Directories
Databases
Business Rules Management System

Figure 4-6 Tivoli Security Policy Manager client external rule calling BRMS API

An example of a BRMS system that Tivoli Security Policy Manager can interface with is IBM WebSphere ILog JRules. JRules provides the JRule Java API that can be used to implement a Tivoli Security Policy Manager external rule using the IExternalRule interface. The RTSS external rules API is discussed in 8.1.5, “External rules” on page 235.

4.10 Conclusion

A key capability of a policy management system is the ability to integrate with external systems. This chapter has examined the capabilities of Tivoli Security Policy Manager to integrate with external systems and the solutions it provides.

Several Tivoli Security Policy Manager integration points were outlined and use cases discussed, including integration with identity and access management, application repositories, role management, classification management, rules engines, and trust services. Finally, this chapter discussed compliance management and its role in completing the policy management life cycle.

Part 3

Usage patterns for IT security policy management

In this part, we take a closer look at the different architecture patterns for externalizing security from applications and services, including the intermediary level, container level, database level, and application level approach.

5

Intermediary level integration

In this chapter, we introduce and describe intermediary level integration with Tivoli Security Policy Manager.

In Chapter 2, “Architecture patterns for externalizing security from applications and services” on page 41, we explored architectural patterns when using Tivoli Security Policy Manager; we now take a closer look at intermediary level integration using this methodology by focusing on the following topics:

- Concepts and benefits
- Java Web Application Servers
- Web Application Firewalls
- Enterprise Service Bus
- Third-party intermediaries
- Conclusion

5.1 Concepts and benefits

When applying an externalized security policy, there are a number of architecture patterns that can be applied. In this chapter, we discuss intermediary level enforcement, the benefit of implementing this architectural design, and the considerations that must be made.

Traditionally, security policy has been implemented at an application level, using custom code and a customized security policy on a per-application basis. Application development using this method is often considered undesirable due to the inherent complexity involved, the business understanding required, and the lack of central policy management. The logical evolution of security policy enforcement is to externalize the policy management and enforcement through security policy server application programming interfaces (API). This method is discussed in more detail in Chapter 8, “Application level integration” on page 213.

Naturally, there are situations when an application specific security implementation is infeasible or incapable of enforcing the required policy. These situations can be easily identified as organizations move towards a common format of service delivery, most notably through *service-oriented architecture* (SOA) implementations, SOAP based web services, and RESTful web services. Tivoli Security Policy Manager can be used to take advantage of this common service delivery environment and reduce the complexity in developing security solutions across a range of environments and business architectures.

Let us consider three example deployment scenarios to explain the implementation detail:

- Scenario A: Established application environments
- Scenario B: Services external to the enterprise
- Scenario C: SOA message protection policies

We briefly explore the scenarios in the following sections and then provide a more detailed explanation of intermediary patterns using Tivoli Security Policy Manager.

5.1.1 Scenario A: Established application environments

A local government has a collection of application server based web services that it needs to secure. New compliance regulations stipulate the need for a more fine grained authorization model, the collection of access audit records, and stringent controls of their security policy governance.

These applications have not been changed by development teams in a number of years and are only supported under a maintenance agreement. The source code is either unavailable or there are insufficient skills available to modify the core application operations and the project team does not have budget or time to reinvest in them. This scenario is shown in Figure 5-1.

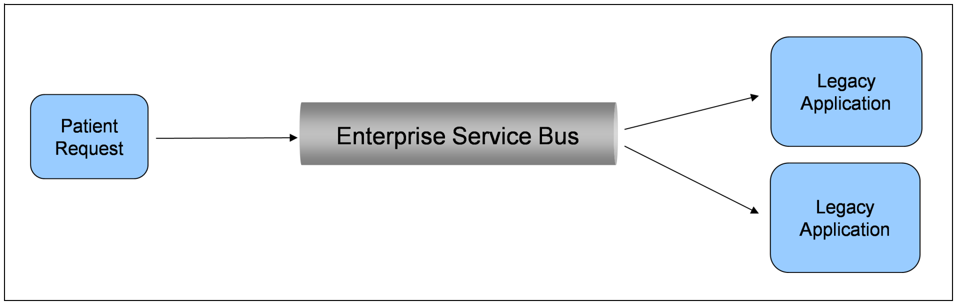


Figure 5-1 Established application business problem

The solution

This is a common scenario in modern computing environments where various elements have been assembled over the lifetime of the environment and may be at different stages of the development and maintenance software life cycle. Tivoli Security Policy Manager has the ability to abstract the security policy and enforcement operations away from the core software architecture and apply decisions without any programmatic modifications to established or third-party application environments.

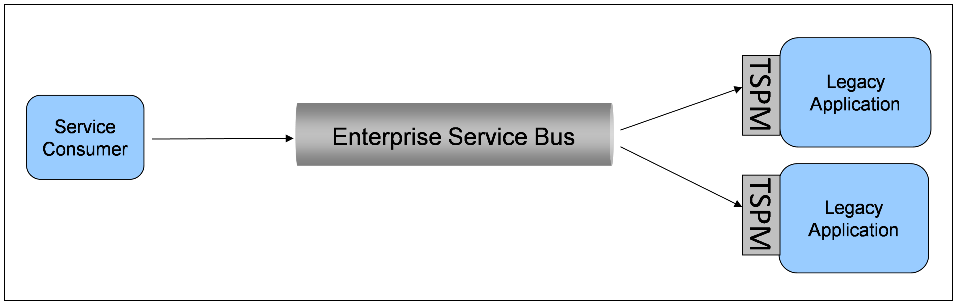


Figure 5-2 Established application solution

When such a solution is deployed in a Java application server environment, the enforcement is performed with Tivoli Security Policy Manager enabled JAX-RPC and JAX-WS interceptors, as shown in Figure 5-2. The policy life cycle and architectural designs are covered in more detail in 1.9, “Introduction to IT security policy life cycle management” on page 32 and Chapter 2, “Architecture patterns for externalizing security from applications and services” on page 41.

5.1.2 Scenario B: Services external to the enterprise

One of the benefits for an SOA is the ability to delegate non-critical parts of a business process to partners or other third parties. For example, a pharmacy may have contracts with different drug suppliers. Although a business relationship is in place between the pharmacy and the supplier, the pharmacy still requires control over who can request drugs from the supplier and what type and quantity of drugs can be requested.

In this example, shown in Figure 5-3, the pharmacy has no control over the remote service. The solution described in Figure 5-2 on page 123 is not suitable for this reason. The reporting available from the partner may not be adequate to properly invoice the cost of drugs to the appropriate patient.

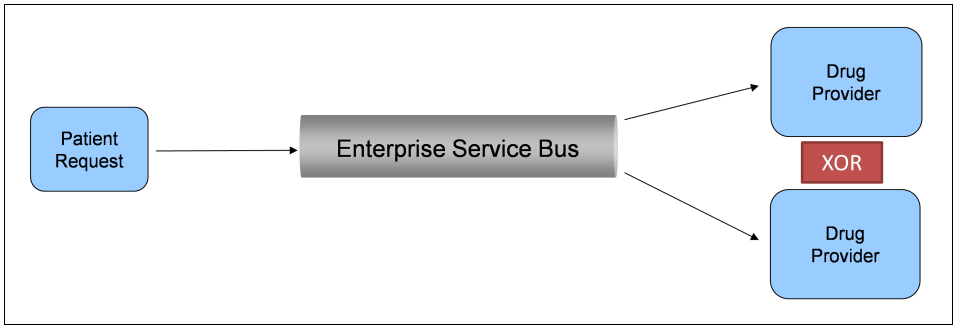


Figure 5-3 High level authorization business problem

The solution

This scenario is becoming more common as more businesses move to focus on their core competencies and delegate non-critical processes to third parties. Other examples include shipping and order fulfillment, payroll processing, and employee benefits.

Tivoli Security Policy Manager provides the capability to integrate at an intermediary level within the enterprise. When integrating with an XML Firewall or Security Gateway, Tivoli Security Policy Manager can be used to author policy for intermediaries that examine the full context of incoming messages and access a selection of *policy information points* before the request is sent to the partner or supplier.

An example of evaluating business requirements from the context of the message is shown in Figure 5-4. An authorization decision can be made allowing a patient to only access goods from a non-interacting pharmaceutical supplier, when their other medications are recorded and available as attributes in their health record using content enrichment services.

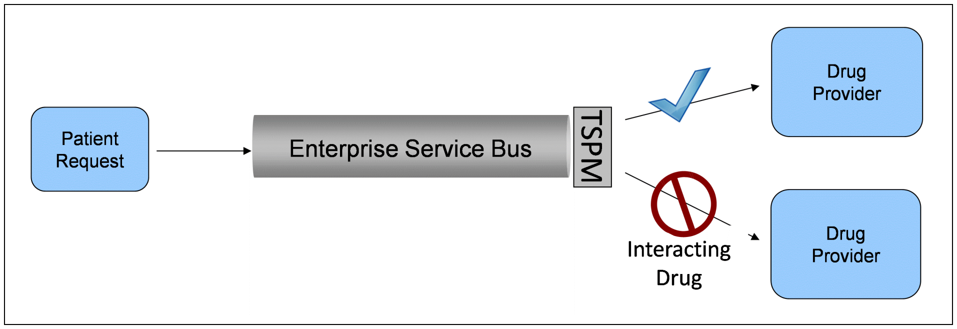


Figure 5-4 High level authorization solution

5.1.3 Scenario C: SOA message protection policies

Consider a bank is wanting to provide a number of SOA based web services to outside mortgage brokers. The availability of these services will allow greater interaction among their partners through up to date loan data and instant loan applications. The messages passed between these providers will likely contain sensitive personal information such as social security numbers, birth dates, home addresses, and confidential banking data, such as account numbers and contract terms.

The bank needs to develop a collection of polices that can ensure that the messages contain the appropriate authentication credentials, that any confidential or sensitive information is appropriately concealed from eavesdroppers, and that the messages have been delivered without tampering. These policies need to be available to the partners invoking the bank’s web services, and mechanisms needs to be in place to ensure that these policies are enforced.

This concept is shown in Figure 5-5.

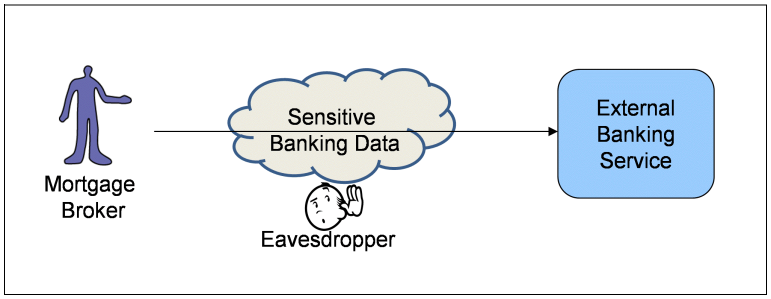


Figure 5-5 SOA message protection business problem

The solution

Organizations across the globe are taking advantage of the ability to collaborate with partners and customers by exposing web services to the Internet. Tivoli Security Policy Manager allows the involved parties to generate policies that govern the security parameters of the messages that are sent across public networks. These policies can be distributed and attached to service definitions, which, when accessed by various enterprise service buses and application server endpoints, can be readily enforced.

By implementing message protection policies, as shown in Figure 5-6, compliance regulations can be met, and various sensitive data can be protected through selective encryption, strong authentication protocols, and methods to ensure that message integrity has not been compromised.

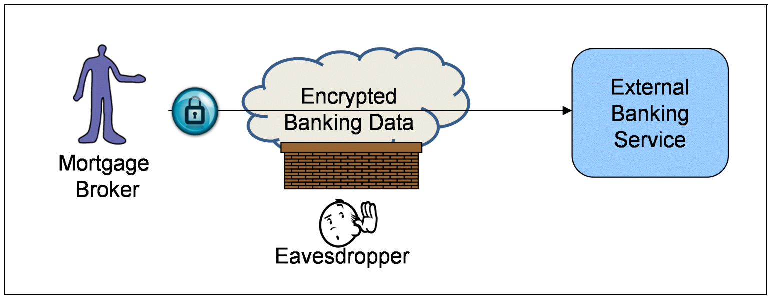


Figure 5-6 SOA message protection solution

5.1.4 Conclusion

An intermediary approach for authorization and message protection policy provides the flexibility to integrate into a SOA environment without the need to maintain or redeploy existing applications. Common business drivers for implementing an intermediary solution include:

- When circumstances dictate that the established application cannot be modified.
- When the party requiring authorization of the service does not control the implementation of the service.
- When the organization is using applications not hosted within their own secure networks.

Tivoli Security Policy Manager can be deployed into existing and new environments providing intermediary integration and adding full capabilities of external authorization, including content rich authorization decisions, central policy management, and IT governance.

5.2 Java Web Application Servers

Intermediaries placed in front of established applications commonly involve using the capabilities of the application server on which the established application is running. Two common Java-based frameworks for implementing web services are JAX-RPC and JAX-WS.

IBM WebSphere Application Server is the implementation by IBM of the Java Platform, Enterprise Edition (Java EE) specification. WebSphere Application Server provides the runtime environment for enterprise applications, including the JAX-RPC and JAX-WS runtime environments.

Tivoli Security Policy Manager provides interceptors for both JAX-RPC and JAX-WS that can be installed on WebSphere Application Server. Before we discuss this action in detail, we briefly introduce some of the underlying security foundation pertinent to our WebSphere Application Server and Tivoli Security Policy Manager integration.

This section focuses on Java Web Application Server integration using an intermediary pattern using JAX-WS JAX-RPC Interceptors. In 6.2, “WebSphere Application Server” on page 161, we further explore container based integration patterns with WebSphere.

5.2.1 Foundation for integration

WebSphere Application Server drives business agility with a performance-based foundation to build, reuse, run, integrate, and manage SOA applications and services. Many organizations use WebSphere Application Server to host their mission critical applications, such as Java EE applications, Portlet applications, and Session Initiation Protocol (SIP) applications.

An advantage of integrating with a Java Web Application Server such as WebSphere Application Server is that applications deployed using container security can be easily managed without needing to implement custom security logic or modules within the application itself. Integration at the container level is discussed in Chapter 6, “Container level integration” on page 159.

In some cases it may not be possible to consume the container level security that requires a different pattern, such as the intermediary integration. Using this pattern, web services, applications, and security gateways generally do not require any maintenance or configuration to enable the use of an external security system for central policy management and governance.

In this section, we briefly introduce the basic concepts that characterize the environment of intermediary integration support with Tivoli Security Policy Manager and Java Web Application Servers.

5.2.2 Java Web Application Server integration and using the policy life cycle model

In this section, we describe the intermediary integration points available within a Java Web Application Server, using Tivoli Security Policy Manager interceptors, by following the policy life cycle model.

Policy modeling and simulation

The intermediary integration pattern is typically used for established application environments where the security constraints for the applications are also generally well established. In such scenarios, much of the policy modeling and simulation may have been completed when the applications were first deployed. At that time, the capabilities added by Tivoli Security Policy Manager to the scenario may not have been possible, so it is therefore essential to review the policy modelling to ensure sufficient coverage of the business rules for accessing the services.

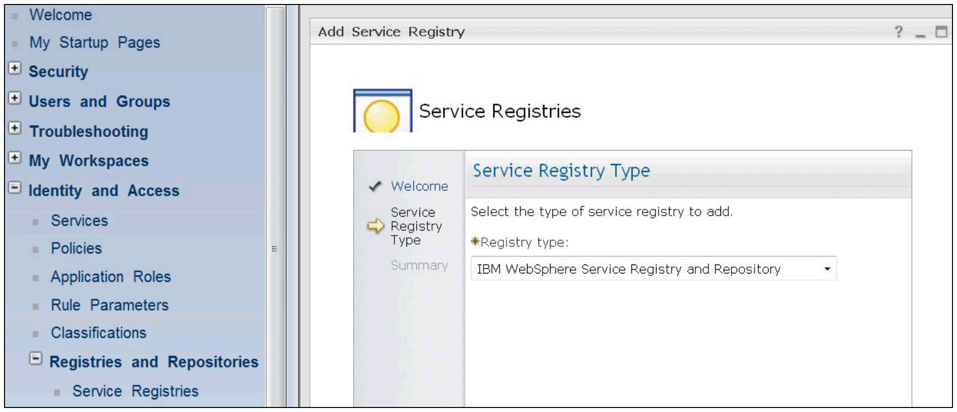
You should follow a structured policy modeling and simulation procedure, for example, by using the Policy Design Tool (discussed in “Policy Design Tool” on page 68). This approach can be used for both established and new web service application deployments, ensuring that the services, resources, and roles are optimized for the operating environment.

Importing resources and metadata

Services that are secured using the Tivoli Security Policy Manager JAX-RPC or JAX-WS interceptors are typically established applications in your environment, of which you may not have access to the original source code or deployment files. As a result, it is generally not possible to add these services as *J2EE applications* or *Portal applications*, so an intermediary pattern for integration is required, which will represent the services as web services in the Tivoli Security Policy Manager console.

The two typical methods to import the definition of the web service into Tivoli Security Policy Manager is to import from a registry such as WebSphere Service Registry and Repository or export the definitions from the deployed application in WebSphere Application Server. It is also possible to import from an Interchange file that can be exported from a utility, such as the Policy Design Tool.

To import services directly from WebSphere Service Registry and Repository, you must first add it as a *Service Registry*, as shown in Figure 5-7. During the import process, select **Services from a service registry** as the Service Type and then select the desired repository from those available in the list and continue with the import steps.



Service Registry Type

Select the type of service registry to add.

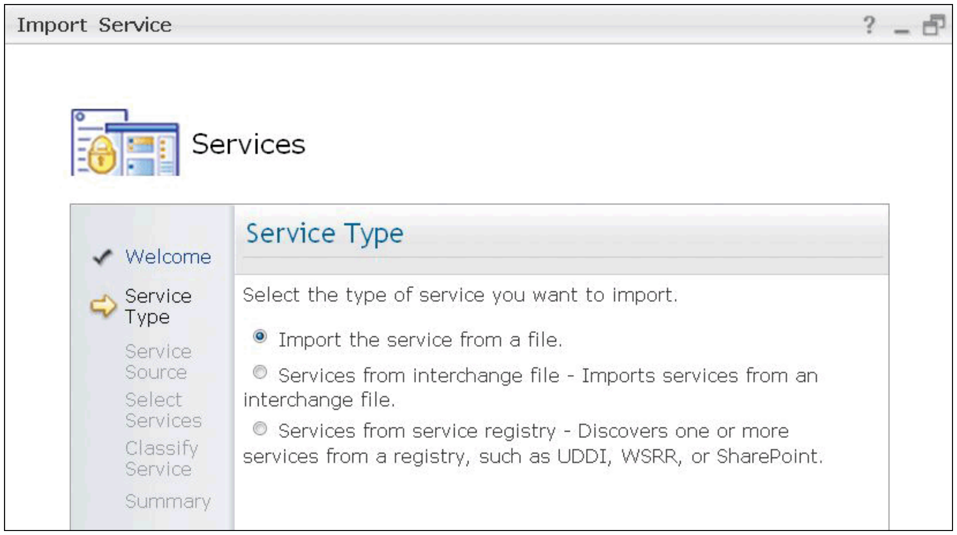
*Registry type: IBM WebSphere Service Registry and Repository

Figure 5-7 Adding IBM WebSphere Service Registry and Repository as a Service Registry

If you want to export the definitions from an application in WebSphere Application Server, first log onto the Integrated Solutions Console (ISC) for the server or domain manager. Navigate to **Applications** in the left menu, expand **Application Types**, and select **WebSphere enterprise applications**. Select the check box of the application containing the web services to secure and then choose **Export**.

The archive that is produced from these steps contains the WSDL files for every web service in the application's modules.

In the Services import window, select the **Import the service from a file** option for the Service Type, as shown in Figure 5-8, and browse to the location where you uploaded the WSDL in the Service Source window.



Service Type

Select the type of service you want to import.

[✓ Import the service from a file.]

Services from interchange file - Imports services from an interchange file.

Services from service registry - Discovers one or more services from a registry, such as UDDI, WSRR, or SharePoint.

Figure 5-8 Importing a service from a file

Now that you have imported the web service from WebSphere Service Registry and Repository or the exported WSDL files for your application, a new *web services* structure is created. An example of a service imported from a WSDL is shown in Example 5-1.

Example 5-1 Web Service resource structure imported from WSDL

```
Web Services
   {http://echo.test.rtss.tscc.ibm.com}EchoService
      EchoService
         echo
         toLowerCase
```

```
    toUpperCase
    whoAmI
```

The imported resource example shown in Example 5-1 on page 130 includes multiple resource types from the web service, each represented as a child service. Table 5-1 shows the mapping of each resource type to the corresponding child service.

Table 5-1 Web service nodes in the services listing

Nodes in services listing	Example resources
Namespace	{http://echo.test.rtss.tscc.ibm.com}EchoService
Service Name	EchoService
Operations	echo, toLowerCase, toUpperCase, and whoAmI

After you have imported your web service resources into Tivoli Security Policy Manager, you can author the policy to be applied to them.

Policy authoring

The policy created in Tivoli Security Policy Manager can be built as a role based policy, which makes it similar to the default programmatic capability of J2EE security. In this case, the centralized approach still has its typical advantages, such as consistency across all deployed instances, audit support, management efficiency, administrative delegation, and so on.

In the policy authoring phase, these options are abstract attribute identifiers and are bound to specific pieces of information in the target IT environment during the policy configuration phase.

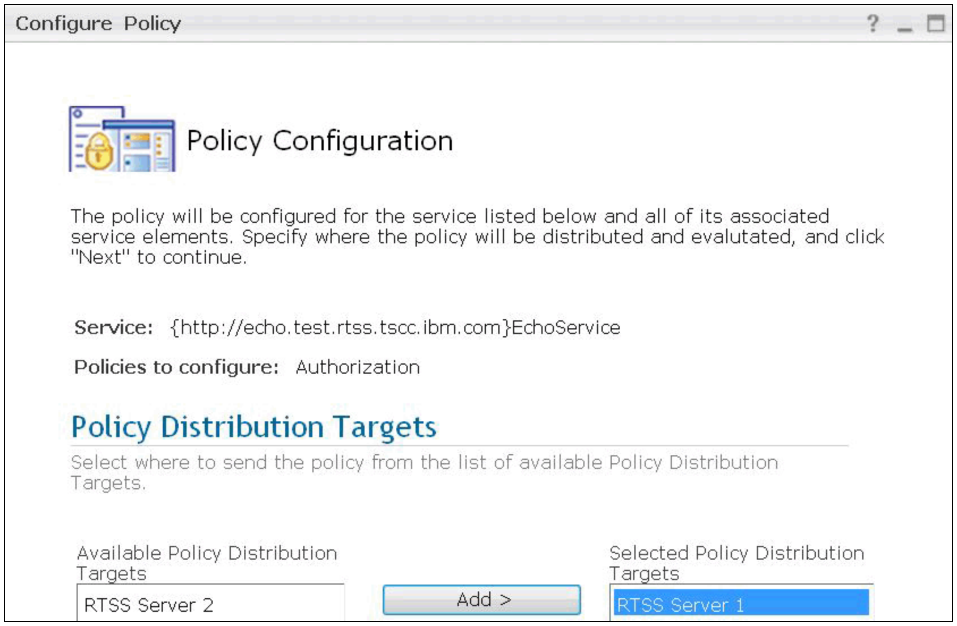
You can attach the policy to the parent node in the service that will be inherited by all the child nodes and you can attach the same policy to more than one service. For web services, the only applicable action is *invoke,* which must be selected when attaching to the desired service.

The full capabilities of rule-based policies can be used during policy authoring. Application Roles and Rules Parameters, including rule parameters and external rules, can be introduced for enriched authorization decision making. Rules are processed in a logic If, Else If sequence that can be reordered as necessary to maintain the desired processing order. Ensure you have selected the correct policy evaluation action (**Permit** or **Deny**) and that your processing logic is correct. During policy evaluation, the access decision of the first rule that is matched will determine the final result.

Policy configuration

Policy configuration provides the mapping between the abstract concepts of application roles and rule parameters to real pieces of information in the IT environment, such as LDAP groups and JDBC queries. Sources of external user repositories must already be configured as User Registries under Registries and Repositories and any User Registry Attribute Queries, Database Attribute Queries, Security Token Service (STS) Attribute Queries or External Rules must be configured in the Authorization Service found under Tivoli Security Policy Manager Runtime Security Service (RTSS) for where the policy will be distributed.

When performing the configuration, first add your desired RTSS instance to the list of the Selected Policy Distribution Targets, as shown in Figure 5-9.


Configure Policy

Policy Configuration

The policy will be configured for the service listed below and all of its associated service elements. Specify where the policy will be distributed and evalutated, and click "Next" to continue.

Service: {http://echo.test.rtss.tscc.ibm.com}EchoService

Policies to configure: Authorization

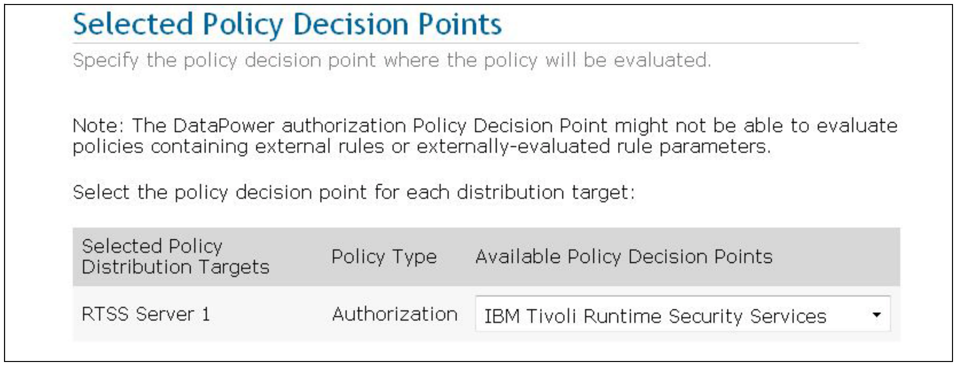
Policy Distribution Targets

Select where to send the policy from the list of available Policy Distribution Targets.

Available Policy Distribution Targets		Selected Policy Distribution Targets
RTSS Server 2	Add >	RTSS Server 1

Figure 5-9 Adding runtime security services as a Policy Distribution Target

Next, click **IBM Tivoli Runtime Security Services** as the Authorization Policy Type for that instance, as shown in Figure 5-10, before clicking **Next**.



Selected Policy Decision Points

Specify the policy decision point where the policy will be evaluated.

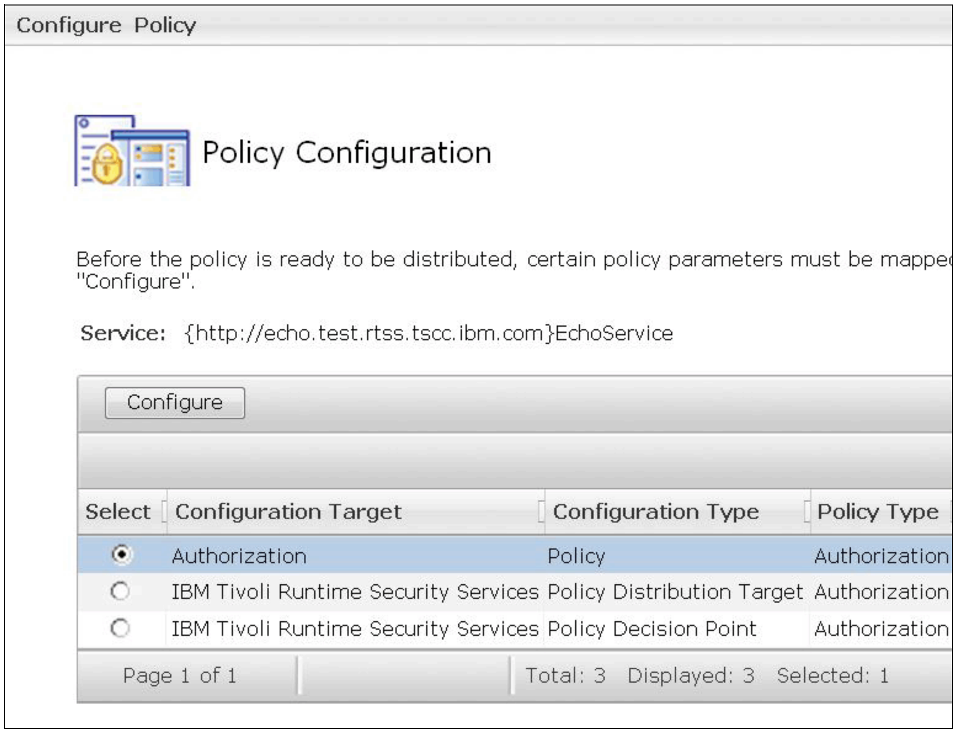
Note: The DataPower authorization Policy Decision Point might not be able to evaluate policies containing external rules or externally-evaluated rule parameters.

Select the policy decision point for each distribution target:

Selected Policy Distribution Targets	Policy Type	Available Policy Decision Points
RTSS Server 1	Authorization	IBM Tivoli Runtime Security Services

Figure 5-10 Selecting the Authorization Policy Type for your Policy Distribution Target

In the next window, you need to select the **Authorization** Configuration Target and click the **Configure** button to begin the mapping process for your Application Roles and Rule Parameters, as shown in Figure 5-11. For each Configuration Target to be mapped, first select the target and click the **Update Mapping** button, which launches the appropriate wizard to allow you to query the appropriate content enrichment sources for your roles and rules.



Configure Policy

Policy Configuration

Before the policy is ready to be distributed, certain policy parameters must be mappe
"Configure".

Service: {http://echo.test.rtss.tscc.ibm.com}EchoService

Select	Configuration Target	Configuration Type	Policy Type
●	Authorization	Policy	Authorization
○	IBM Tivoli Runtime Security Services	Policy Distribution Target	Authorization
○	IBM Tivoli Runtime Security Services	Policy Decision Point	Authorization

Page 1 of 1   Total: 3   Displayed: 3   Selected: 1

Figure 5-11 Configuring the Authorization Configuration Target

Application Roles are mapped to existing groups in your configured user registry and you can map a role to include more than one group. Ensure that you select the correct user registry that contains the group you want to map to, enter the group query parameters, and click **Search** to look up the registry. Select the group or groups for the mapping and click **OK** to save the mapping.

Map roles to groups: Ensure that only groups are selected. Both users and groups may be returned from the user registry, but application roles should only be mapped directly to groups.

A unique capability when configuring Rule Parameters in a policy for web services enforcement is the ability to access the entire SOAP body at decision time. This action allows rule parameters to be bound to XPath queries into this SOAP body, meaning decisions can be made based on the message content and not only request parameters or attributes. When you perform the rule mapping, you should select **Rule parameter is contained in the request, but in a non-standard** location as the value for where the rule parameter is located to allow you to use an XPath query.

Using the SOAP message shown in Example 5-2, a rule parameter could be bound to the `<app:CustID>` value using an XPath expression such as "`//*[local-name()='CustID']`". This setup allows the customer ID 1234, or any other value in the message body, to be used in evaluation of a rule that could then look up another value in an external database to determine the authorization decision.

Example 5-2 SOAP message body extract

```
<soap:Body>
   <app:GetCustomerData>
      <app:CustID>1234</app:CustID>
   </app:GetCustomerData>
</soap:Body>
```

When you have configured all of your Application Roles and Rule Parameters, you can prepare for policy distribution.

Policy distribution

During the installation and configuration of your Tivoli Security Policy Manager environment, it is likely that you have created at least one RTSS instance. The RTSS that you distribute the authorization policy to must correspond to the RTSS instance that the Tivoli Security Policy Manager JAX-RPC and JAX-WS interceptors are configured to use. If you have not yet added an RTSS instance, you must create one before you can distribute your policy.

Handling policy distribution targets: Tivoli Security Policy Manager V7.1 requires that you use the **`tspmRegisterRTSS`** command-line utility to add or delete Tivoli Runtime Security Services policy distribution targets. If they are added using the console, distribution will not complete successfully.

Policy enforcement

To protect your web services running on a Java Web Application Server using an intermediary pattern, you must manually configure either the Tivoli Security Policy Manager JAX-RPC or JAX-WS interceptors. The installation can be performed on a single server or a cluster, and unlike the container scenario described in 6.2, "WebSphere Application Server" on page 161, RTSS can be installed in either local or remote mode.

An important consideration about whether to configure local or remote mode is if the RTSS client is not running in local mode note; each authorization decision request has to leave the local JVM, which may impact performance. Although not strictly required, the local mode installation positively influences performance.

Installing the appropriate interceptors for the JAX-RPC and JAX-WS frameworks is trivial and is as simple as placing a library for each framework in the appropriate location.

After you have installed an interceptor on a server, all of the web services of that same type are automatically protected by Tivoli Security Policy Manager. If no policy has been defined and distributed in Tivoli Security Policy Manager, then all requests for that service are denied due to the *deny bias*. Authoring a specific policy with a Permit authorization decision allows access to the services.

Access through an interceptor - the *deny bias*: After an interceptor is installed on an application server, all matching requests will be intercepted and access will be denied unless explicitly allowed by distributed policy.

To demonstrate the transparency of interceptors and to assist in migration where no previous policy existed, applying an *Any user* policy with *Permit* authorization decision to the service effectively returns the access control to which it was before the interceptor was installed. Of course, appropriate authorization policies should be authored and distributed in a timely fashion to secure your resources by way of intermediary integration.

Auditing and reporting

Because JAX-RPC and JAX-WS frameworks run within the Java Web Application Server, applicable logs for recording and reporting access requests and services on the server should be used for auditing and reporting.

When the Tivoli Security Policy Manager interceptors are used as an intermediary pattern, they add value in this scenario by transparently adding auditing capabilities in a manner consistent with other enforcement types, such as J2EE container enforcement and programmatic API invocation. To view these records, refer to the Tivoli Security Policy Manager audit logs for the appropriate RTSS instance.

5.2.3 Conclusion

The WebSphere Application Server container provides a number of integration points to provide external authorization for web services using Tivoli Security Policy Manager.

The use of Tivoli Security Policy Manager to intercept web service calls provides rich policy constructs that are not normally available to traditional containers. To achieve this scenario, Tivoli Security Policy Manager uses standards-based interfaces for integration. The significant value-add when using Tivoli Security Policy Manager for external authorization is that the application being protected does not have to be modified, or even be aware that any authorization is being performed by the interceptor. This situation ensures the application logic and process flow is not compromised while still being able to be protected using fine grained authorization security policy.

5.3 Web Application Firewalls

Many organizations are continuing to adopt *cloud-based* solutions in their enterprise and transform their IT operations to SOA models. In doing so, there is a need to protect applications exposed through XML based interfaces such as WSDL and to scan Extensible Markup Language (XML) traffic. With these services hosted outside of the traditional network boundaries, it is not only inbound traffic to the organization that must be secured, but also the outbound requests.

A Web Application Firewall, also referred to as a *hardware appliance*, *network device,* or *gateway,* is typically deployed in the Demilitarized Zone (DMZ) of an enterprise network. These specialist appliances are used to secure, accelerate, and transform XML based traffic. Security of the traffic for Web Services and XML based resources can include validation of content or messages and to apply security policy, which may not be provided natively by the resources themselves, but which are needed to meet organizational or regulatory requirements. They can also be used to filter XML content and perform XML acceleration and transformation as part of the processing.

5.3.1 Foundation for integration

Web Application Firewalls are important network-based appliances for organizations using XML based technologies such as web services. These appliances can reduce the processing impact on existing web application servers by parsing, validating, transforming, and routing XML messages using XPath and XSLT on specifically designed hardware rather than in software before passing the messages off to the back-end server. Web Application Firewalls can also perform mediation such as translation of authentication tokens, expanding the capability of integration with external parties.

By having an appliance perform the bulk of the processing and translation, these Web Application Firewalls are generally able to provide performance optimization, especially when all of the required business attributes for authorization decisions are included in the request. Relying solely on the message contents does limit the flexibility and capability of the policy constructs to those supported by the appliance.

The capability to call external content enrichment data stores allows an authorization policy to be deployed that not only uses the business data available in the request, but also to derive data from other information sources. To achieve this scenario, an appliance must be capable of integrating with an external policy decision point (PDP), of which the appliance then acts only as the policy enforcement point (PEP).

Where more than one appliance exists in the organization, such as multiple front-end systems or internal and external appliances, the management of multiple devices becomes time consuming and prone to error. The capability for central policy management becomes highly desirable, which allows the management of multiple appliances for consistent policy and governance.

5.3.2 WebSphere DataPower SOA Appliance integration with Tivoli Security Policy Manager

IBM WebSphere DataPower SOA Appliances provide message transformation, integration, and routing functions in a network device, cutting operational costs and improving performance. The appliances are quickly and easily configurable, helping you protect you environment against cross-site scripting, SQL injection, and a wide variety of XML threats.

The appliances provide reliability and scalability by securing services at the network layer with XML, SOAP, and WS-Web services processing and authorization policy enforcement, which can be integrated with and delegated to Tivoli Security Policy Manager using RTSS.

WebSphere DataPower can be configured to perform the role of a PDP by consuming the distributed XACML policy (on box) or it can be configured to call a remote RTSS instance (off box). In either case, it always acts as the PEP for authorization decisions determined by the PDP.

Integration between WebSphere DataPower and Tivoli Security Policy Manager can also be used to author and distribute a message protection policy that is enforced locally on the appliance (on box). In scenarios where WebSphere DataPower and WebSphere Service Registry and Repository are already deployed, Tivoli Security Policy Manager can be added to the integration architecture, providing a central security policy management and governance capability. This setup provides the mechanism to author an authorization policy that is enforced by WebSphere DataPower and a message protection policy that is updated in WebSphere Service Registry and Repository.

Additionally, WebSphere DataPower can configure multiple on-board PDP objects that are represented as distinct policy distribution targets (PDTs) in Tivoli Security Policy Manager, which provides support for multiple WebSphere DataPower domains.

Integration of WebSphere DataPower with Tivoli Security Policy Manager for message protection policy is discussed in 5.4, “Enterprise Service Bus” on page 147. In the remainder of this section, we focus on integrating WebSphere DataPower with Tivoli Security Policy Manager for an authorization policy.

5.3.3 IBM WebSphere DataPower integration and using the policy life cycle model

Now that we have explored the integration points available within WebSphere DataPower, we can prepare to deploy Tivoli Security Policy Manager as part of the security landscape by following the policy life cycle model.

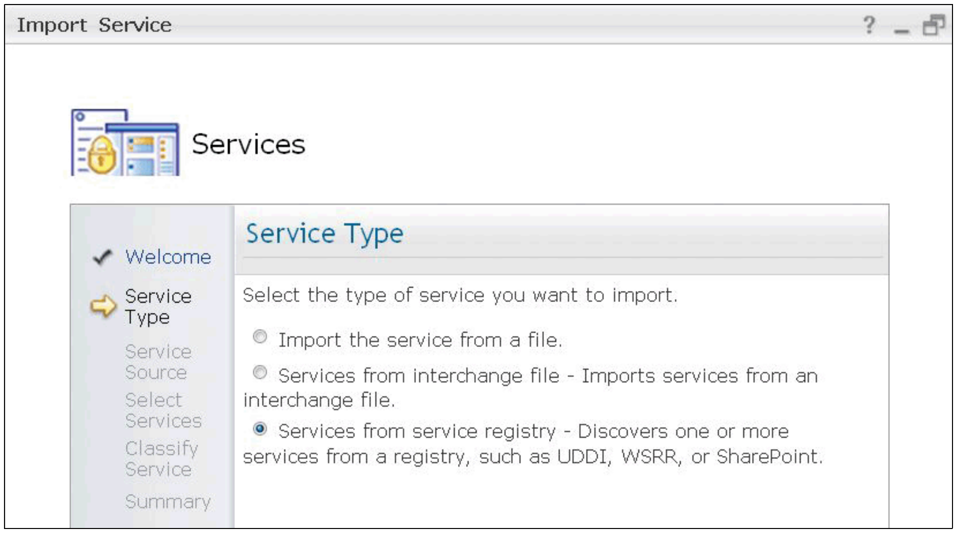
Policy modeling and simulation

As an intermediary, WebSphere DataPower in many cases is already acting as a web security gateway and policy enforcement point for your XML-based applications. In this scenario, much of the policy modeling and simulation may have been completed when the applications were first deployed. However to ensure an optimized integration approach, existing services, resources, and roles should be reviewed as part of the integration planning. Tivoli Security Policy Manager integrates into this scenario by proving the central authorization policy management for existing services and resources exposed by WebSphere DataPower.

For new web service application deployments, you should follow a structured policy modeling and simulation procedure, for example, by using the Policy Design Tool.

Importing resources and metadata

For deployments that contain WebSphere Service Registry and Repository in the system landscape, services that are protected by WebSphere DataPower can be imported directly into the Tivoli Security Policy Manager console. To import services directly from WebSphere Service Registry and Repository, you must first add it as a Service Registry. During the import process, select **Services from a service registry** as the Service Type, as shown in Figure 5-12, and then click **Next**. In the Service Source wizard, select the service registry you have configured from those available in the Service registry drop-down menu.



Select the type of service you want to import.

( ) Import the service from a file.

( ) Services from interchange file - Imports services from an interchange file.

(●) Services from service registry - Discovers one or more services from a registry, such as UDDI, WSRR, or SharePoint.

Figure 5-12 Importing a service from IBM WebSphere Service Registry and Repository

If you do not have WebSphere Service Registry and Repository, or WebSphere DataPower is protecting resources that are only known to it, you can also import the service from the WSDL file or interchange file. To import from a WSDL file, you must export the file from DataPower and copy it locally to where the console is running. Be aware of referenced WSDL files, as these are not supported and result in an error during the import process. If you use an interchange file, you must also copy this locally to the console.

Fix pack information: Tivoli Security Policy Manager V7.1 Fix Pack 3 handles referenced XSD files by disregarding them.

After importing the web service either from a registry or file, a new service is created under the Web Services service type, as shown in Example 5-3.

Example 5-3 Web Services resource structure

```
Web Services
   {http://echo.test.rtss.tscc.ibm.com}EchoService
      EchoService
         echo
         toLowerCase
         toUpperCase
         whoAmI
```

The imported resource example shown in Example 5-3 includes multiple resource types from the web service, each represented as a child service. Table 5-2 shows the mapping of each resource type to the corresponding child service.

Table 5-2 Web service nodes in the services listing

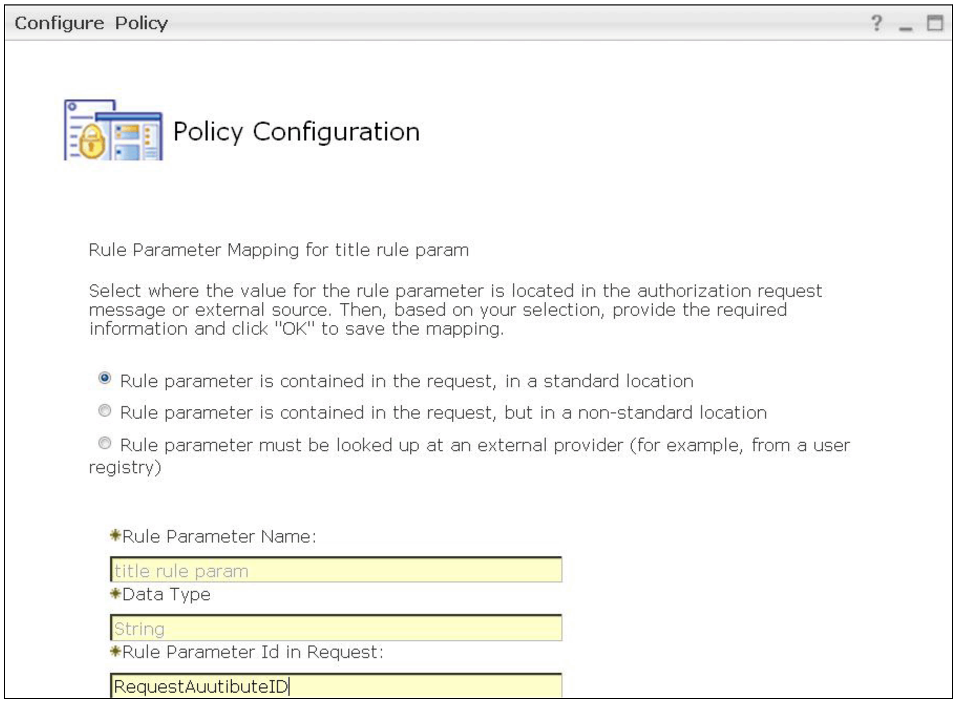
Nodes in services listing	Example resources
Namespace	{http://echo.test.rtss.tscc.ibm.com}EchoService
Service Name	EchoService
Operations	echo, toLowerCase, toUpperCase, and whoAmI

After you have imported your web service resources into Tivoli Security Policy Manager, you can author the policy to be applied to them.

Policy authoring

During authorization policy authoring, you must first consider the deployment scenario for your WebSphere DataPower appliance and whether you will use the on-board PDP capabilities or call an external RTSS PDP. The decision of which type of PDP depends on the attributes available in the request, which can be used to evaluate the fine grained authorization rules.

When using the on-board PDP, the only supported content enrichment sources are LDAP PIPs, although customized support can be extended by authoring custom XSL. During LDAP PIP evaluation, you can only use attributes *in the request in a standard location*, as shown in Figure 5-13, which is mapped during policy configuration. External rules are not supported and DataPower encounters an error during processing, not at policy distribution time.



Policy Configuration

Rule Parameter Mapping for title rule param

Select where the value for the rule parameter is located in the authorization request message or external source. Then, based on your selection, provide the required information and click "OK" to save the mapping.

- [✓ Rule parameter is contained in the request, in a standard location]
- Rule parameter is contained in the request, but in a non-standard location
- Rule parameter must be looked up at an external provider (for example, from a user registry)

*Rule Parameter Name: title rule param

*Data Type: String

*Rule Parameter Id in Request: RequestAuutibuteID

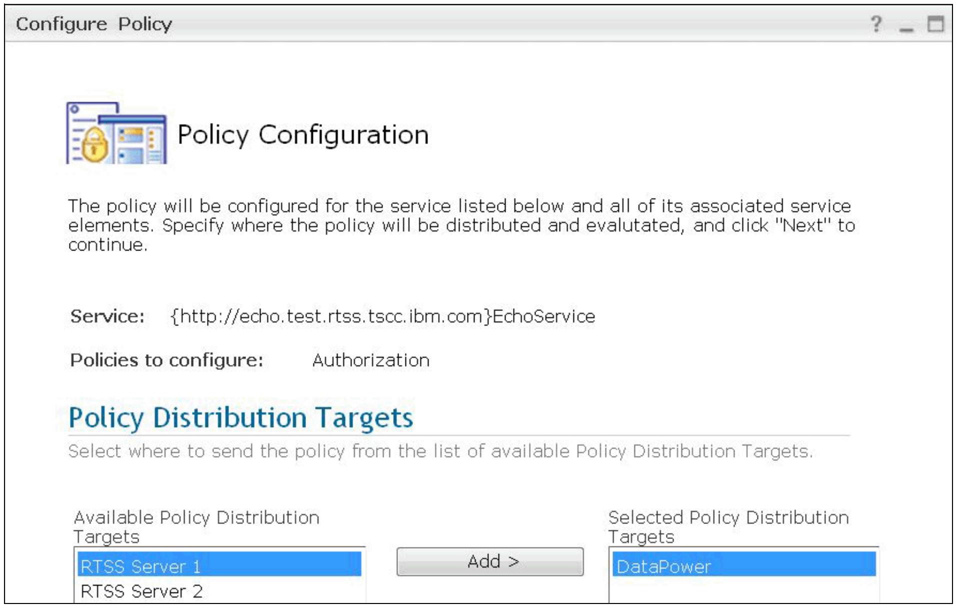
Figure 5-13 Rule parameter mapping during Policy Configuration using request attributes

If the off-box PDP is configured, the full capability and support of Application Roles and Rules can be incorporated into your authorization policies and are evaluated by the remote RTSS.

When attaching your authorization policy, ensure you select the action of Invoke for the web service resource you want to protect.

Policy configuration

Before configuring your policy, you must add your WebSphere DataPower appliance as a PDT using the add process for PDTs in Registries and Repositories in the console. You need the host name of the target appliance and the *Web Service Client Port*, which is configured on the appliance. In the Policy Configuration window, select your appliance as the PDT, as shown in Figure 5-14, and select **DataPower** as the Authorization Policy Type, as shown in Figure 5-15.



The policy will be configured for the service listed below and all of its associated service elements. Specify where the policy will be distributed and evalutated, and click "Next" to continue.

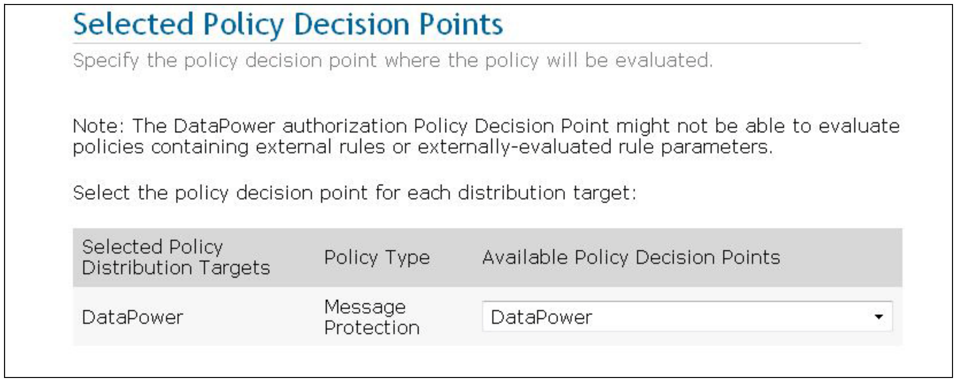
Service: {http://echo.test.rtss.tscc.ibm.com}EchoService

Policies to configure: Authorization

Select where to send the policy from the list of available Policy Distribution Targets.

Available Policy Distribution Targets	Selected Policy Distribution Targets
RTSS Server 1	DataPower
RTSS Server 2	

Figure 5-14 Adding IBM WebSphere DataPower as a Policy Distribution Target



Specify the policy decision point where the policy will be evaluated.

Note: The DataPower authorization Policy Decision Point might not be able to evaluate policies containing external rules or externally-evaluated rule parameters.

Select the policy decision point for each distribution target:

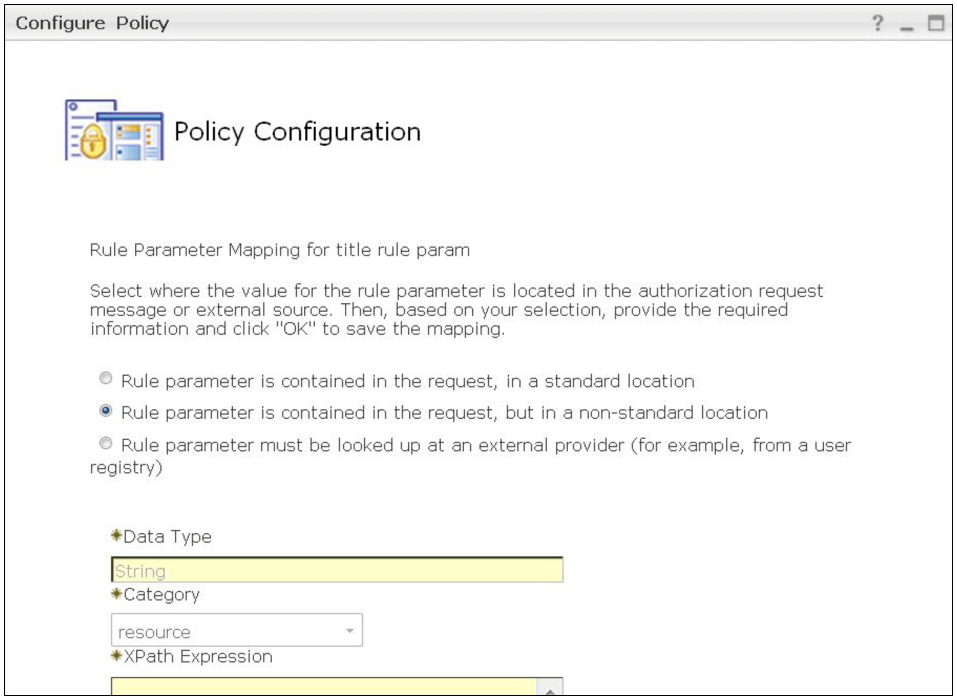
Selected Policy Distribution Targets	Policy Type	Available Policy Decision Points
DataPower	Message Protection	DataPower

Figure 5-15 Selecting the Authorization Policy Type for your Policy Distribution Target

When configuring your authorization policies, you must again consider whether the authorization policy will be evaluated by an on-box or off-box PDP, as this choice dictates from which locations in the request that attributes can be accessed and used.

If you are using an on-box PDP, when you configure the Authorization Configuration target, ensure that you only use attributes that are in the request in a standard location that can be retrieved using the credential mapping stylesheet.

When you use an off-box PDP, not only can you use the request attributes as mappings for you rules, you can use XPath to reference data in the SOAP body to access attributes and data contained in the request in a nonstandard location, which provides application context authorization decisions, as shown in Figure 5-16.



Policy Configuration

Rule Parameter Mapping for title rule param

Select where the value for the rule parameter is located in the authorization request message or external source. Then, based on your selection, provide the required information and click "OK" to save the mapping.

- Rule parameter is contained in the request, in a standard location
- [✓ Rule parameter is contained in the request, but in a non-standard location]
- Rule parameter must be looked up at an external provider (for example, from a user registry)

*Data Type: String

*Category: resource

*XPath Expression

Figure 5-16 Rule parameter mapping during Policy Configuration using XPath

After you have completed the mapping of your Application Roles and Rules in the Authorization Configuration target, no further configuration is necessary. The WebSphere DataPower SOA Appliances and DataPower configuration targets do not require configuration.

Policy distribution

In addition to adding the WebSphere DataPower appliance as a PDT in Tivoli Security Policy Manager, a number of configuration steps must be completed on the appliance before the authorization policy can be enforced. Ensure that you have completed the required steps for your appliance version and Tivoli Security Policy Manager version.

A policy is distributed that is secured both at the message level and by enforcing authentication and authorization so that only valid appliances can consume the policies. Tivoli Security Policy Manager distributes policy using a protocol based on WS-Notification and WS-MetadataExchange, using XML protection measures, such as digital signatures, to ensure the integrity of the transferred policy. Policy updates can be configured to be sent only over SSL for an additional layer of protection and confidentiality.

When the PDT is configured to consume the policy notifications, the WebSphere DataPower appliance requests the updated policy from Tivoli Security Policy Manager and then receives the actual policy documents over the secured channel. The appliance stores the policy as a file in a predefined PDT directory, which acts as a Tivoli Security Policy Manager policy store for a DataPower domain. The directory can also be configured as a source of data for a DataPower XACML PDP object., which allows it to be used in an on-box XACML authorization decision point object as part of a AAA Policy object.

The effective authorization policy that is distributed to WebSphere DataPower is in the form of XACML, and as such, no example XACML is shown here.

On the DataPower appliance, the policy update is handled by an XML Firewall service or a Multi-Protocol Gateway service that is listening for the Tivoli Security Policy Manager notification. When the DataPower service receives the notification, the service's policy fetches the relevant policy or policies and marshals their storage in the specified PDT directory.

You can publish both an authorization policy and a message protection policy for the same web service to a DataPower PDT at the same time. There can be multiple DataPower PDTs on an appliance, but the receive updates stylesheet (`tspm-receive-updates.xsl`) shipped with the appliance can only be used in one PDT per domain. It is also not possible for a single DataPower PDT to support policy stores in multiple domains.

Policy enforcement

With intermediary based enforcement, a WebSphere DataPower appliance intercepts the request and enforces the policies before providing access to the applications. The appliance can function both as a PDP and a PEP or, alternatively, the appliance can function simply as a PEP using RTSS as an external PDP.

WebSphere DataPower uses an AAA policy, which is configured to reference the configured XACML PDP object. During enforcement, the configured PDP is called and examines the directory containing policy distributed from Tivoli Security Policy Manager. The appliance then grants or denies access to the requested web service using the result from the PDP.

Auditing and reporting

As an intermediary, WebSphere DataPower acts as the PEP and, depending on the configuration, also as the PDP. The appropriate access and security logs from the appliance should be used to provide event data for auditing and reporting, which can be used to evaluate policy effectiveness and coverage.

If an off-box PDP is configured, you can use the RTSS audit logs for central auditing and reporting.

5.3.4 Conclusion

Many organizations use WebSphere DataPower appliances to enforce secure and controlled access to their web service resources and may also use IBM WebSphere Service Registry and Repository to manage the publishing and message protection policies (also known as *message security policies*) for those resources. Tivoli Security Policy Manager can be quickly and easily integrated into existing deployments of either product, providing central authorization and message protection policy management. This setup reduces the impact of managing policies across multiple appliances, reducing time and increasing consistency of policy application.

This section demonstrated how to use Tivoli Security Policy Manager as an external PDP by configuring an appliance to call an off-box RTSS, adding the full capabilities of Application Roles and Rules to add content enrichment data for authorization policy decisions.

5.4 Enterprise Service Bus

Enterprise Service Bus (ESB) is a service-oriented infrastructure component that makes large-scale implementation of SOA principles flexible, reusable, and manageable in a heterogeneous world.

Applications are represented as business services built from core services that provide a set of capabilities that are worth advertising for use by other services. Typically, a business service relies on many other services in its implementation. Services interact through the ESB, which facilitates mediated interactions between service endpoints. The ESB supports event-based interactions as well as message exchange for service request handling.

One innovation of the ESB is a common model for messages and events. All messages can become events if deploying the service binds the message to a topic in the event space, which reduces the complexity of the applications being integrated.

Implementing an ESB facilitates greater reuse of IT assets by separating application logics and integration tasks, so you can reduce the number, size, and complexity of integration interfaces. In doing so, you can add or change services with minimal interruption to the existing IT environment, reducing cost and risk involved as business changes and new opportunities arise.

There are multiple products from a range of vendors that can provide implementations of the logical functionality of an ESB. Using IBM technologies, some of the ESB product offerings include:

- WebSphere Enterprise Service Bus
- WebSphere Message Broker
- WebSphere DataPower

The decision as to which product implementation to use as your ESB upon your environment and business requirements. In this section, we will provide a foundation of a typical integration environment and explore a WebSphere DataPower implementation in depth.

5.4.1 Foundation for integration

Many organizations using an ESB such as WebSphere Enterprise Service Bus or WebSphere Message Broker use the integration capabilities of IBM WebSphere Service Registry and Repository to:

- Provide endpoint resolution based on a service level agreement established in WebSphere Service Registry and Repository with the Governance Enablement Profile and policy resolution support extended to cover all Service Component Architecture (SCA) binding types and manual endpoints configured in WebSphere Service Registry and Repository
- Use entity (document) information for routing, dynamic transformation, and other types of policy processing in an up-to-date memory cache.

In an SOA environment using an ESB, events sent and received to and from the bus represent an important security consideration, especially if those event messages are not isolated within a trusted corporate network. Applying and managing native message security policies within the ESB or WebSphere Service Registry and Repository can become error prone and time consuming as additional applications are added to the repository. To protect the integrity of the messages and confidentiality of the data exchanged, measures must be implemented to protect these applications.

A message protection policy is a set of conditional security requirements that define the circumstances under which a message can be sent or received (exchanged) in a web services environment. A message must meet these requirements before it can be exchanged on the bus.

Tivoli Security Policy Manager can be used for a message protection policy by:

- Evaluating the message against the conditions in a policy. Each condition evaluates to either true or false depending on the content of the message request and the conditions of the policy
- Permitting or denying the message exchange, depending on the result of the evaluation and the permit or deny settings in the policy.

The policy also defines the services with messages that you want to protect. The messages that you author and manage with Tivoli Security Policy Manager must adhere to OASIS Web Services Security (WSS) specifications. To successfully author and manage message protection policies, you must be knowledgeable about the OASIS Web Services Security elements; the Tivoli Security Policy Manager console provides assistance in building these rules. The elements include assertions, tokens, and bindings that are used in the messages you want to protect.

WS-SecurityPolicy defines the assertions, such as security tokens and security bindings, that can be used in a policy for a web services environment. Tivoli Security Policy Manager supports WS-SecurityPolicy Version 1.2.

Integration for message protection policies within an ESB deployment can be achieved using Tivoli Security Policy Manager and either:

- IBM WebSphere DataPower SOA appliance, as the PDT, PDP, and PEP
- IBM WebSphere Services Registry and Repository, as the PDT

The remainder of this section focuses on integrating WebSphere DataPower with Tivoli Security Policy Manager for a message protection policy. Usage of WebSphere DataPower for an authorization policy is described in 5.3, “Web Application Firewalls” on page 137.

5.4.2 WebSphere DataPower SOA Appliance integration with Tivoli Security Policy Manager

In 5.3.3, “IBM WebSphere DataPower integration and using the policy life cycle model” on page 139, we explored the integration points available within WebSphere DataPower using an authorization policy. We now consider message protection policy using Tivoli Security Policy Manager as part of the security landscape by following the policy life cycle model.

Policy modeling and simulation

This phase is similar to those steps followed when using WebSphere DataPower as an intermediary for an authorization policy, as discussed in 5.3.2, “WebSphere DataPower SOA Appliance integration with Tivoli Security Policy Manager” on page 138. Typically, authorization and message protection policy are modeled at the same time when applications are first deployed. By combing the authorization and message protection policy modeling and simulation, you can achieve much tighter levels of integration with greater coverage across your enterprise applications.

When you model the business rules and requirements in this phase, it is important to consider the origin and destination of your messages and whether those networks are trusted. Using this information, you can then plan the level of confidentiality and integrity needed for those messages and the endpoints with which your policies will interact. Tivoli Security Policy Manager integrates into this scenario by proving the central message protection policy management for existing services and resources, which can be enforced by WebSphere DataPower, but could equally be applied to IBM WebSphere Service Registry and Repository to manage the message protection policy requirements.

For new web service application deployments, you should follow a structured policy modeling and simulation procedure, for example, by using the Policy Design Tool.

Importing resources and metadata

The service structure of your applications for which a message protection policy will be applied is the same as that defined for an authorization policy of web services. Further information about importing resources and metadata when using WebSphere DataPower for an authorization policy can be found in 5.3.3, "IBM WebSphere DataPower integration and using the policy life cycle model" on page 139.

You can either import your services from WebSphere Service Registry and Repository, directly from the service from the WSDL file, or an interchange file, for example, one exported from the Policy Design Tool. If you are importing from a file, remember to copy this file to the local file system of the Tivoli Security Policy Manager console.

After importing the web service, a new service is created under the Web Services service type, as shown in Example 5-4.

Example 5-4 Web Services resource structure

```
Web Services
   {http://echo.test.rtss.tscc.ibm.com}EchoService
      EchoService
         echo
         toLowerCase
         toUpperCase
         whoAmI
```

The imported resource example shown in Example 5-4 includes multiple resource types from the web service, each represented as a child service. Table 5-3 shows the mapping of each resource type to the corresponding child service.

Table 5-3 Web service nodes in the services listing

Nodes in services listing	**Example resources**
Namespace	{http://echo.test.rtss.tscc.ibm.com}EchoService
Service Name	EchoService
Operations	echo, toLowerCase, toUpperCase, and whoAmI

After you have imported your web service resources into Tivoli Security Policy Manager, you can author the policy to be applied to them.

Policy authoring

Using Tivoli Security Policy Manager, you can create and manage policies that evaluate the assertion types in the incoming messages that you want to protect.

Requests for access to web services are sent as SOAP messages, which contain information about the request and the requester, but they also contain information about how and under what conditions the message should be sent and received. This information is called an *assertion* and a message can include one or more assertions. You can use Tivoli Security Policy Manager to author policies with the assertions your organization requires or you can import policies from a file, such as one created in the Policy Design Tool.

Examples of some of the assertion types are shown in Table 5-4. Refer to the product documentation for all supported assertion types available for use in your policies.

Table 5-4 Assertions in a message protection policy

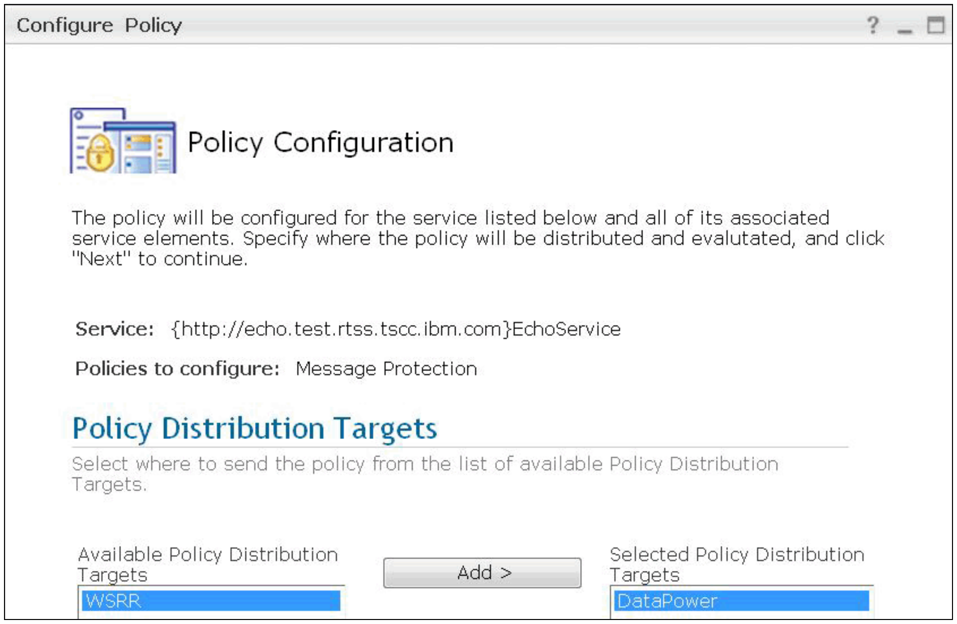
Assertion type	Example assertions	Purpose
Signature	SignedElements SignedParts	Specifies the parameters for digital signatures that must be applied to incoming messages.
Encryption	EncryptedElements EncryptedParts	Specifies the parameters for encryption that must be applied to incoming messages.
Require Message Parts	RequiredElements RequiredParts	Describes specific data that incoming messages must contain.
Security Binding	Binding type: TransportBinding SymmetricBinding AsymmetricBinding Security Header Formatting: Layout	Specifies parameters for the cryptographic algorithm and security token used to exchange messages securely. Specifies the formatting standard to which the SOAP security header must conform.

Assertion type	Example assertions	Purpose
Supporting Tokens	SupportingTokens EndorsingSupportingTokens	Specifies one or more security tokens to be used when exchanging messages between the web service and its clients.
Algorithm suite	Basic256 TripleDes Basic256Rsa15 Basic256Sha256 TripleDesSha256	Specifies the algorithms and key lengths that must be used for encryption and digital signature operations.
Security Token	X509Token KerberosToken SamlToken	Specifies the type and parameters of security tokens that must accompany incoming messages.
SOAP Message Security	Wss10 Wss11	Specifies the version of the SOAP Message Security specification. Also specifies the types of SOAP message security requirements.
WS-Trust Options	Trust13	Specifies information that defines additional extensions defined in the WS-Trust Version 1.3 specification.

When attaching your message protection policy, no actions can be configured for the web service resource you have selected, as the policy is applicable to the transport of the message itself, not the operation.

Policy configuration

Before configuring your policy, you must add your WebSphere DataPower appliance as a PDT using the add process for PTDs in Registries and Repositories in the console. You need the host name of the target appliance as well as the Web Service Client Port that is configured on the appliance. In the Policy Configuration window, select your appliance as the PDT, as shown in Figure 5-17.



The policy will be configured for the service listed below and all of its associated service elements. Specify where the policy will be distributed and evalutated, and click "Next" to continue.

Service: {http://echo.test.rtss.tscc.ibm.com}EchoService

Policies to configure: Message Protection

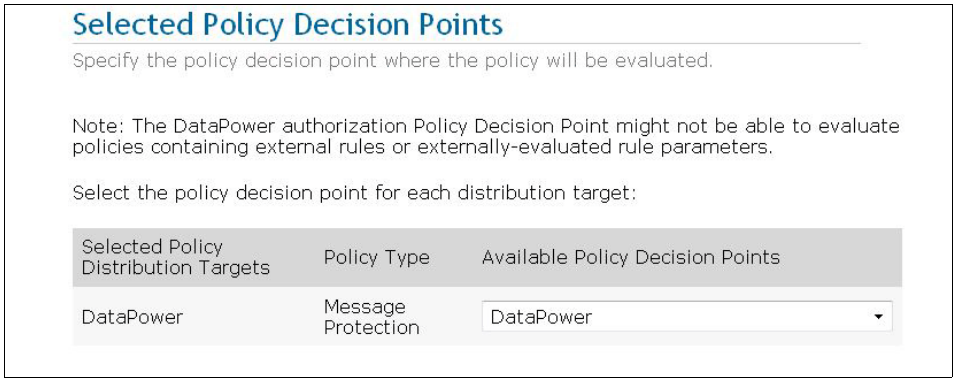
Policy Distribution Targets

Select where to send the policy from the list of available Policy Distribution Targets.

Available Policy Distribution Targets
WSRR
Add >
Selected Policy Distribution Targets
DataPower

Figure 5-17 Adding IBM WebSphere DataPower as a Policy Distribution Target

Select **DataPower** as the Message Protection Policy Type, as shown in Figure 5-18, before clicking **Next** to complete the policy configuration.


Selected Policy Decision Points

Specify the policy decision point where the policy will be evaluated.

Note: The DataPower authorization Policy Decision Point might not be able to evaluate policies containing external rules or externally-evaluated rule parameters.

Select the policy decision point for each distribution target:

Selected Policy Distribution Targets	Policy Type	Available Policy Decision Points
DataPower	Message Protection	DataPower

Figure 5-18 Selecting the Message Protection Policy Type for your Policy Distribution Target

No additional mapping or further configuration is required, Configuration targets with a Message Protection policy type and the policy can now be distributed.

Policy distribution

In addition to adding the WebSphere DataPower appliance as a PDT in Tivoli Security Policy Manager, a number of configuration steps must be completed on the appliance before the message protection policy can be distributed. Ensure that you have completed the required steps for your appliance version and Tivoli Security Policy Manager version. You can publish both an authorization policy and a message protection policy for the same web service to a DataPower PDT at the same time.

Policy distribution itself is secured both at the message level and by enforcing authentication and authorization so that only valid appliances can consume the policies. Further information about policy distribution can be found in 5.3.2, “WebSphere DataPower SOA Appliance integration with Tivoli Security Policy Manager” on page 138.

An example of a simple message protection policy is shown in Example 5-5. This policy requires that the entire message be signed to protect the contents of the message.

Example 5-5 Effective Message Protection policy

```
<wsp:Policy Name="urn:ibm:names:TSPM:7.1:Required_Signature"
wsu:Id="bb0c1b53-4eda-49bc-88eb-b80586e82444" >
   <sp:SignedParts/>
```

```
</wsp:Policy>
<wsp:PolicyAttachment>
   <wsp:AppliesTo>
      <wsp:URI>
         http://echo.test.rtss.tscc.ibm.com#wsdl11.service(EchoService)
      </wsp:URI>
   </wsp:AppliesTo>
   <wsp:PolicyReference URI="urn:ibm:names:TSPM:7.1:Required_Signature"
/>
</wsp:PolicyAttachment>
```

The message protection policy is stored on the appliance and is used during enforcement, which ensures that the message contains the required level of protection.

Policy enforcement

A Tivoli Security Policy Manager message protection policy is enforced in the DataPower Web Service Proxy service object by creating a WS-Policy object and attaching it to a web service element. The WS-Policy object uses the message protection policy distributed from Tivoli Security Policy Manager.

On the appliance, in the Policy tab of the Web Service Proxy service object, add a WS-Policy object based on the Tivoli Security Policy Manager message protection policy file. Note that the external sources for the WS-Policy object cannot be configured at the WSDL level of the Web Service Proxy. The WSDL level corresponds to the top level of the service definition in Tivoli Security Policy Manager.

Any Tivoli Security Policy Manager message protection policy can be used by WebSphere DataPower, but the relevant subject entries must be included in the WSDL configured in the Web Service Proxy object. The actual subject entry can be determined from the AppliesTo attribute in the PolicyAttachment element of the policy file produced by Tivoli Security Policy Manager.

If the requirements of the message protection policy are not met, the message itself will not be accepted and processed by WebSphere DataPower and an appropriate error message will be returned to the web services client.

Auditing and reporting

As an intermediary, WebSphere DataPower acts as the PDP and PEP for evaluating message protection policies. The appropriate access and security logs from the appliance should be used to provide event data for auditing and reporting, which can be used to evaluate policy effectiveness and coverage.

No external Tivoli Security Policy Manager components are included in the policy decision and enforcement process. The audit logs created by Tivoli Security Policy Manager can be used to audit and report on policy authoring, configuration, and distribution operations.

5.4.3 Conclusion

A common scenario for SOA deployments using an ESB is to define message security policy in WebSphere Service Registry and Repository or enforcement with WebSphere DataPower. This scenario can be significantly enhanced by using Tivoli Security Policy Manager to centrally manage the required message protection policies and distribute them to WebSphere DataPower for enforcement, WebSphere Service Registry and Repository for central registration, or both.

Tivoli Security Policy Manager can be quickly and easily integrated into existing deployments of either products, providing central authorization and message protection policy management. This setup reduces the impact of managing policies across multiple appliances, reducing time and increasing the consistency of the policy application.

5.5 Third-party intermediaries

Tivoli Security Policy Manager uses the XACML OASIS standard for authorization policy, and as such, can be integrated into existing enterprise deployments that are XACML aware. Further, through the use of custom interceptors for authorization and mediation modules for Enterprise Service Bus systems, authorization and message security can be centrally managed and integrated with Tivoli Security Policy Manager.

Using a custom interceptor for your web application server or mediation module or security node for your ESB, it is possible to call a Tivoli Security Policy Manager policy decision point, which provides rich policy constructs to leverage content enrichment data stores.

When used as an intermediary, this approach prevents the need for modifications or ongoing maintenance of your established applications or XML gateways while enjoying the benefits that Tivoli Security Policy Manager can provide.

5.6 Conclusion

Applying externalized security policy has been implemented at an application level, using custom code and customized security policy on a per-application basis. Application development using this method is often considered undesirable due to the inherent complexity involved, the business understanding required, and the lack of central policy management. Integrating in this way can often lead to application specific implementations that require application development resources for policy maintenance, which requires considerable specialized skills, knowledge, context, time, and expense.

Tivoli Security Policy Manager can be used to reduce the pain points of traditional security integration by providing the capability for central policy management for both authorization and message security. The developed policies can be easily maintained and applied to your services in many cases without the needing to modify the applications or for them to even be aware that authorization and message protection has been delegated.

The use of Tivoli Security Policy Manager can help ensure consistent policy application, management, and governance.

6

Container level integration

In this chapter, we discuss the container level enforcement approach and considerations for using this pattern. Earlier in this book, we discussed policy life cycle management when using Tivoli Security Policy Manager. The details of the container level integration approach will be outlined using this methodology.

We examine the following container based integration solutions with Tivoli Security Policy Manager:

- WebSphere Application Server
- Microsoft environment

6.1 Concepts and benefits

Traditionally, security policy has been implemented at an application level, using custom code on a per-application basis. Application development using this method is often considered undesirable for a number of reasons. The costs involved with this approach are generally higher, as each application maintains its own security implementation. The business understanding required to effectively implement security controls is significant, requiring the development organization to understand detailed business requirements, which is not always a trivial exercise. This approach also creates a fractured view of security that does not provide visibility into the organization's overall security defences.

Security services provide a mechanism for developers to create applications without the need to worry about the implementation details of security policy or the enforcement mechanisms that enforce it. In Chapter 5, “Intermediary level integration” on page 121, we examined the intermediary approach. In this chapter, we examine how a container based approach can help an organization deliver security in a cost-effective way.

Containers provide runtime services such as life cycle management, transaction management, deployment services, security, and so on. These services are common services used by application developers to develop solutions faster, more consistently, and more reliably.

In the context of security policy management, it is useful to integrate with the security services provided by container implementations, which allows the application developer to continue to use the benefits of container based security services while allowing policy to be centrally managed. This scenario brings visibility to the overall security solution across an organization, therefore creating a more consistent and secure solution. It also allows container based security to be integrated with business context information from external systems, providing a tool to express complex policy across an organization.

Significant cost savings and improvement in an organization’s overall IT security can be realized using this approach, as it removes complex security logic from applications while using a standards based security implementation.

This chapter discusses two container level integrations using Java and Microsoft .NET technology.

6.2 WebSphere Application Server

WebSphere Application Server is the IBM implementation of the Java Platform, Enterprise Edition (Java EE). It provides a container based runtime environment for enterprise applications so that they can be developed faster, more consistently, and more cost effectively.

WebSphere Application Server V7.0: Technical Overview, REDP-4482 provides more detail about the basics of WebSphere Application Server and can serve as a good starting point. In addition, the WebSphere Application Server Information Center is located at the following address:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.home.doc/welcome.html

We discuss the underlying security foundation pertinent to WebSphere Application Server and Tivoli Security Policy Manager integration, and then we examine the Java EE support in detail by stepping through the seven policy life cycle stages.

6.2.1 Foundation for integration

In this section, the basic concepts that characterize the environment of the J2EE integration of Tivoli Security Policy Manager are discussed. These concepts include:

- Containers as an application run time and the related J2EE security model
- The JACC standard for externalization of authorization
- Tivoli Security Policy Manager as a JACC provider

Containers and J2EE security

Containers provide runtime services for applications so that they can be created faster and more reliably. WebSphere Application Server provides the following container support that is relevant to the integration with Tivoli Security Policy Manager:

- The web container processes servlets, JSPs (processed as servlets), and other types of server-side includes. Requests are received through the HTTP protocol.
- The EJB container provides all of the runtime services that are needed to deploy and manage enterprise beans.

- The portlet container processes JSR 286 compliant portlets. JSR 286 is the Java portlet API 2.0. The portlet container is discussed in 8.3.2, "Integration with WebSphere Portal" on page 253.

Security in a WebSphere Application Server environment has many facets. We focus here on the topics of authentication and authorization at the container level, which is often referred to as Java EE Security. Unless otherwise specified, *application* implies both web and EJB based applications.

Java EE uses a role-based authorization model that consists of these definitions and mappings:

- The application defines *roles* and *protected resources* within its deployment descriptors. Deployment descriptors are files within the .ear or .war package that describe the application.
- The deployment descriptors also define *security constraints* that map the protected resources to a defined role. For example, to access a particular URL, the caller must possess the "CEO" role. For any given user request, the application server can identify the list of roles that are authorized to access the request resource. This is called *resource-to-role mapping*.
- The security administrator is responsible for assigning users and groups to roles. If a group is assigned a role and a user is a member of that group, then the user is said to *possess* that role. For any given user, the application server can identify the list of roles possessed by the user. This is called *user-to-role mapping*.

The authorization decision is then done by looking for a match between the two lists of roles. If the user possesses any permitted role, the access is allowed. Otherwise, it is rejected.

Introducing JACC

Java Authorization Container Contract (JACC) was introduced by the Java Specification Request (JSR) 115 process. This specification defines a *contract* (interfaces and rules) between a J2EE container and an authorization framework provider, which allows the latter to provide both authorization policy management and access decision services to the former.

The contract allows a third-party authorization provider to plug into a Java EE application server to make authorization decisions when a Java EE resource is accessed. WebSphere Application Server provides the required interface so that any third-party authorization solution can be used as a JACC provider.

Tivoli Security Policy Manager as a JACC provider

Tivoli Security Policy Manager provides a set of capabilities that allows it to act as a JACC provider. We explore this setup in more detail in the following sections, guided by the life cycle stages that were introduced earlier.

For the sake of completeness, it should be mentioned that there are alternate ways for applications to make authorization requests to Tivoli Security Policy Manager from within WebSphere Application Sever. These ways match the application level integration pattern and are not addressed here. For example, arrow (1) in Figure 6-1 shows that applications have the ability to call the Tivoli Security Policy Manager Runtime Security Service (RTSS) directly through the Tivoli Security Policy Manager authorization API. Interceptors are also available for JAX-WS and JAX-RPC web services, as discussed in 5.2, “Java Web Application Servers” on page 127 and applications can also call the generic XACML over SOAP interface, as discussed in “Authorization web service interface” on page 227. Note that Tivoli Access Manager is included in this figure. It is not a required component for container based integration, but can be used to propagate authenticated identities from Tivoli Access Manager to Tivoli Security Policy Manager, as discussed in 4.2.1, “Integration with Tivoli Access Manager for e-business” on page 100.

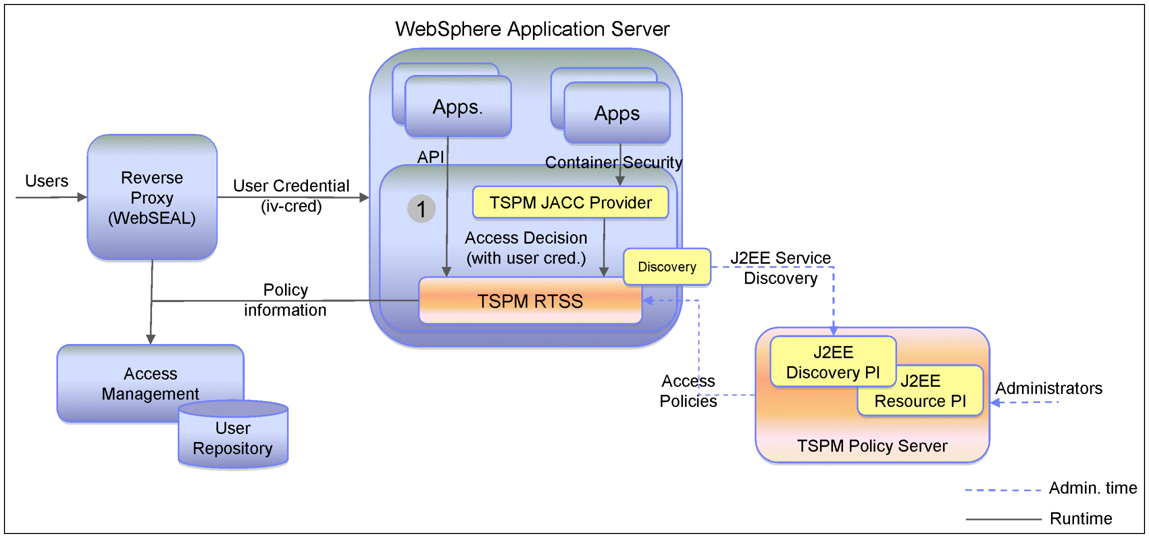


Figure 6-1 WebSphere Application Server JACC (container level) Integration

6.2.2 WebSphere integration using the policy life cycle model

The WebSphere Application Server container level integration is outlined in the security landscape of the policy life cycle model.

Policy modeling and simulation

Java EE applications declare security requirements through annotations or by using deployment descriptors. This existing policy should only be considered a starting point when managing the security of the application in Tivoli Security Policy Manager. The existing policy can be used, enhanced, or it can be ignored. This situation is especially useful when adding security to an existing or third-party application that does not have security, or security that does not meet the business requirements.

Importing resources and metadata

Tivoli Security Policy Manager can discover the resources contained in a Java EE application and model them using the service structure shown in Figure 6-2.

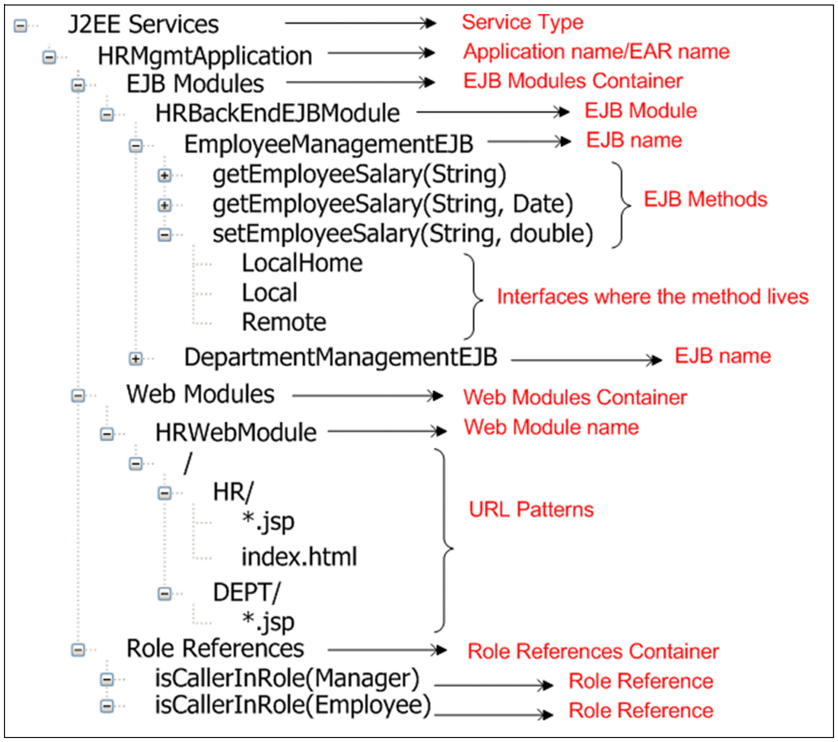


Figure 6-2 J2EE service model

There are several steps that need to be carried out to establish the connection between WebSphere Application Server and Tivoli Security Policy Manager. Those steps are described in detail in the following two books:

- *IBM Tivoli Security Policy Manager Version 7.1 Configuration Guide*, GC27-2713
- *IBM Tivoli Security Policy Manager Version 7.1 Administration Guide*, SC23-9476

We highlight the required steps here.

1. Install and deploy RTSS in local mode on the WebSphere Application Server server instance.
2. Configure Tivoli Security Policy Manager JACC provider on the WebSphere Application Server console.
3. Create the RTSS service registry on the Tivoli Security Policy Manager console.
4. Discover and selectively import J2EE application resources as service resources into Tivoli Security Policy Manager.

These steps will now be briefly discussed.

Installing and deploying RTSS in local mode

The installation can be performed on a single server or cluster. For this integration, it is mandatory to install the RTSS client in local mode because of the following reasons:

- Application resources can only be discovered by RTSS if both are running on the same application server.
- The default policy for the installed applications needs to be deployed to the JACC provider, which only works in local mode.
- Each authorization decision request has to leave the local JVM if the RTSS client is not running in local mode, which can impact performance. Although not strictly required, the local mode installation allows the solution to deliver much better performance.

Configuring the Tivoli Security Policy Manager JACC provider

Now that the RTSS client is known to WebSphere Application Server, it must be defined as an external JACC provider. This action is done through the WebSphere Application Server administration console. Restarting the application server enables JACC for the entire WebSphere Application Server server (WebSphere Application Server V7.0 for the security domain).

Creating the RTSS service registry

Tivoli Security Policy Manager should be configured to discover the applications on WebSphere Application Server. The local RTSS client created in the previous steps can be used as a service registry. Chapter 8, “Managing service registries”, of *Tivoli Security Policy Manager Version 7.1 Administration Guide*, SC23-9476 has a section named “Adding runtime security services as a service registry” that provides the details about how to accomplish this task. In essence, a Runtime security services type registry is added to the list of service registries by using the Tivoli Security Policy Manager console.

Discovering and importing applications as services

This step uses the Tivoli Security Policy Manager Import services function by using the service registry created in the previous step. Chapter 3, “Managing services”, of *Tivoli Security Policy Manager Version 7.1 Administration Guide*, SC23-9476 has a section named “Discovering J2EE application resources from a registry” that lists all the steps to accomplish this task. In essence, RTSS is contacted and returns to the console all the resources it discovers in the specified application as it is deployed in WebSphere Application Server. The Tivoli Security Policy Manager administrator can now import the application definition using the service structure shown in Figure 6-2 on page 164. The selected application resources are added to the list of services in Tivoli Security Policy Manager.

Coexistence of different policy management approaches

A distinction must be made between two different use patterns for RTSS as a JACC provider. The difference relates to the management, that is, how does the effective policy get created. The different methods do not affect the enforcement, which always involves the JACC provider. Figure 6-3 on page 167 shows the relevant flows:

1. Assume we have two applications, Application 1 and Application 2. We want to keep the original policy for Application 1, but we want to use Tivoli Security Policy Manager to manage the policies for Application 2. Initially, all policies must be propagated to the JACC provider because, once enabled, the JACC provider will have to make all the decisions for all applications.
2. In case of Application 1, any future changes will remain to be made on the application server. Every change will have to be re-propagated to the JACC provider, either by wsadmin scripting or by using the WebSphere Application Server console on WebSphere Application Server V7 or later.

3. For Application 2, however, there must be no re-propagation of the policy after the initial propagation. The effective policy will be provided by Tivoli Security Policy Manager using its policy distribution mechanism (see “Policy distribution” on page 168). If an unintended propagation has occurred by mistake (for example, by propagating all applications instead of specific ones), this situation can be corrected by re-distributing the proper policies from Tivoli Security Policy Manager.

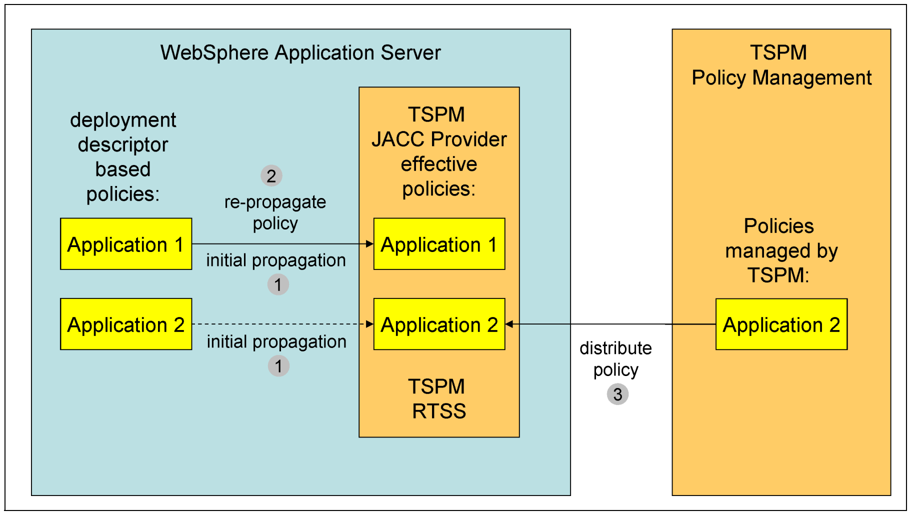


Figure 6-3 JACC effective policies

Policy authoring

Policy created in Tivoli Security Policy Manager can be built as a role based policy, which makes it similar to the default deployment descriptor based policy. In this case, the centralized approach still has its typical advantages, such as consistency across all deployed instances, audit support, management efficiency, administrative delegation, and so on. However, beyond that, the full capabilities of rule based policies can be used. Rule parameters and external rules can be introduced for enriched decision making.

In the policy authoring phase, policy is expressed as abstract attribute identifiers. These identifiers relate to method invocation parameters for EJBs, and HTTP query string parameters in the case of web applications. This relationship is established in the next stage of the life cycle, which is policy configuration.

When you integrate your deployment with IBM Tivoli Access Manager for e-business (which is a common scenario), all user credential attributes can be referenced in the same way.

Policy configuration

Policy configuration provides specific Java EE support that allows parameters to be explicitly mapped to either EJB method parameters or to HTTP request parameters. Chapter 6, “Managing authorization policies”, of *Tivoli Security Policy Manager Version 7.1 Administration Guide*, SC23-9476 has a section named “Mapping rule parameters”, which describes this support in detail, as well as the Tivoli Access Manager credential case. In essence, for the source of the rule parameter to be mapped, select **Rule parameter is contained in the request, in a standard location**, then complete the Rule Parameter ID in Request field as follows:

- `urn:ibm:jacc:1.0:ejb:parameter-value:`*<n>*

  Use this rule parameter for EJB method parameters (replace *<n>* with a parameter offset number, starting at 0).

- `urn:ibm:jacc:1.0:resource:http-param:`*<name>*

  Use this rule parameter for HTTP request parameters (replace *<name>* with an actual parameter name as seen in the query string).

The Rule Parameter Name is the name of the referred parameter in the policy.

In the case of Tivoli Access Manager credential attributes, the rule parameter identifiers must match the names in the credential and be uppercase characters.

Policy distribution

Regarding policy distribution, there are no special considerations for Java EE. The local RTSS client running on the target application sever serves as the distribution target. As mentioned before, policy distribution may become necessary even if no changes were made, such as in the case of overridden policies caused by inadvertent propagation of local default policies.

Policy enforcement

The Java EE/JACC integration described in this section supports both declarative as well as programmatic security. An application invoking isCallerinRole to check the user’s role membership, to exploit this action programmatically, does not communicate directly with the enforcement point. It is still the container that translates this request into permissions to call to the JACC provider, which in turn translates the call into XACML requests and evaluates against the deployed policy. The resulting Permission objects are as follows:

- Declarative:

WebUserDataPermission	Can the user connect via HTTP or HTTPS?
WebResourcePermission	Can the user access a given URL?

EJBMethodPermission	Can the user invoke this method on this EJB?

- Programmatic:

WebRoleRefPermission	Program calls isUserInRole() from a servlet / JSP.
EJBRoleRefPermission	Program calls isCallerInRole() from an EJB.

Auditing and reporting

Regarding auditing and reporting, there are no special considerations for J2EE. Remember that we have seen that Tivoli Security Policy Manager adds value in this scenario by transparently providing its auditing capabilities.

6.2.3 Conclusion

The WebSphere Application Server containers provide a number of integration points to allow external authorization for web applications, EJBs, and portlets using Tivoli Security Policy Manager.

The use of Tivoli Security Policy Manager for delegated authorization provides rich policy constructs that are not normally available to the containers. However, to achieve this scenario, Tivoli Security Policy Manager uses a standards-based interface (JACC) where this is applicable and provides its own extensions where there is no standard that could be exploited.

Note: Make sure to understand the distinction between programmatic exploitation of the container based approach, as described throughout this chapter, and application level integration, as described in Chapter 8, "Application level integration" on page 213.

6.3 Microsoft environment

Many organizations have a deployment of at least one Microsoft software application, if not their entire operating environment. As such, integration with Microsoft environments is of significant importance and Tivoli Security Policy Manager provides a number of integration points for providing external authorization. In this section, we describe the integration levels available in a Microsoft environment and provide examples of how integration can be achieved, with links to existing integration offerings.

6.3.1 Microsoft container integration

In the context of integration, this section refers to the Microsoft Windows operating environment as the *container* level that is providing the integration capabilities for Tivoli Security Policy Manager. This situation is due to all of the integration points being able to coexist and interoperate as though they were all operating at the same level, and security policy can be written to take a holistic approach to security for Microsoft based applications.

The Microsoft .NET Framework is the lowest level for interaction within the Microsoft environment for Tivoli Security Policy Manager. It can be used directly from within Microsoft Windows and can also be used in applications that are accessed through the web using ASP.NET, hosted on Microsoft Internet Information Services (IIS). At the core of the integration, the Microsoft .NET Framework provides the necessary resources for applications and custom code to externalize their authorization to Tivoli Security Policy Manager. Figure 6-4 shows the relationship between Microsoft Windows, the Microsoft .NET Framework and services accessed through the web, plus the *software stacking*.

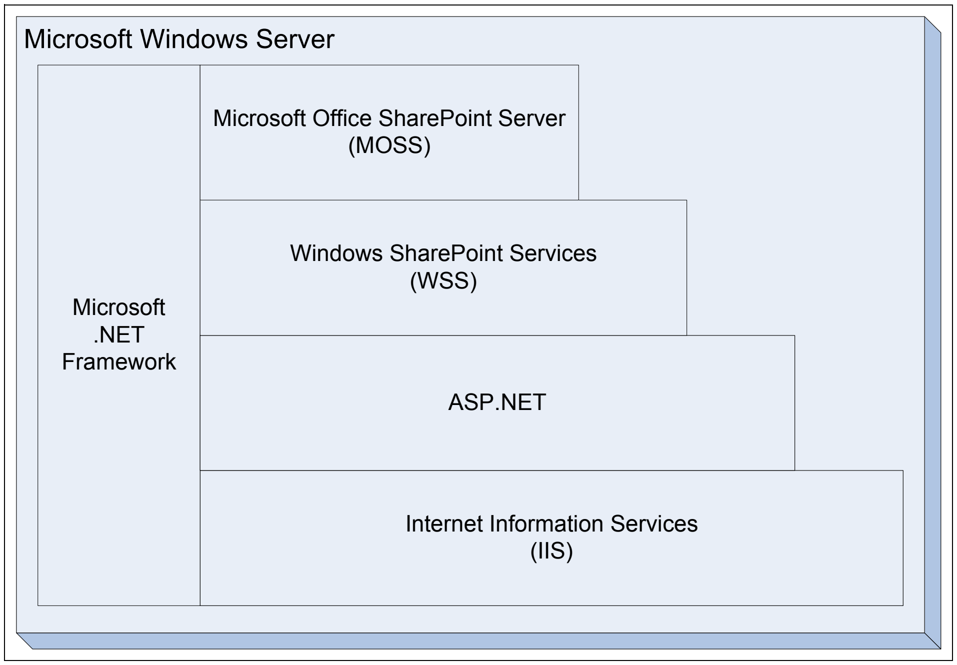

Microsoft Windows Server
Microsoft .NET Framework
Microsoft Office SharePoint Server (MOSS)
Windows SharePoint Services (WSS)
ASP.NET
Internet Information Services (IIS)

Figure 6-4 Microsoft container integration

Figure 6-4 on page 170 uses Windows SharePoint Services (WSS) and Microsoft Office SharePoint Server (MOSS) as a commercial application example that is layered upon the Microsoft .NET Framework (including ASP.NET) and Internet Information Services (IIS) to integrate in the Microsoft Windows container. Users of these applications are unaware of how the applications actually work and are really only interested in the fact that after they have logged in to their Microsoft Windows Desktop, they have access to all their individual applications from within the one container.

As an alternative to the WSS example, any custom application hosted on IIS and executed through ASP.NET could be shown, as could any stand-alone .NET application that is executed from the desktop.

MOSS can be considered as an integration of its own, as it is a commercial application that has been written to exploit the services and resources provided by WSS and the .NET Framework. It provides a ready to use solution that covers a number of major use cases, but still has the flexibility to be extended using custom coded solutions for additional functionality.

6.3.2 Integration with Tivoli Security Policy Manager

There are a number of integration points that can be used when using Tivoli Security Policy Manager to provide external authorization to Microsoft applications. The integration point, or combination of points, depend on what type of application is being integrated, such as a custom .NET application or commercial application (such as Microsoft Office SharePoint Server) and what additional data sources are required for authorization decisions (such as Microsoft Active Directory).

In Figure 6-5, the integration components are shown, including the layer in the Microsoft container they integrate with and how these components interact with the Tivoli Security Policy Manager server components.

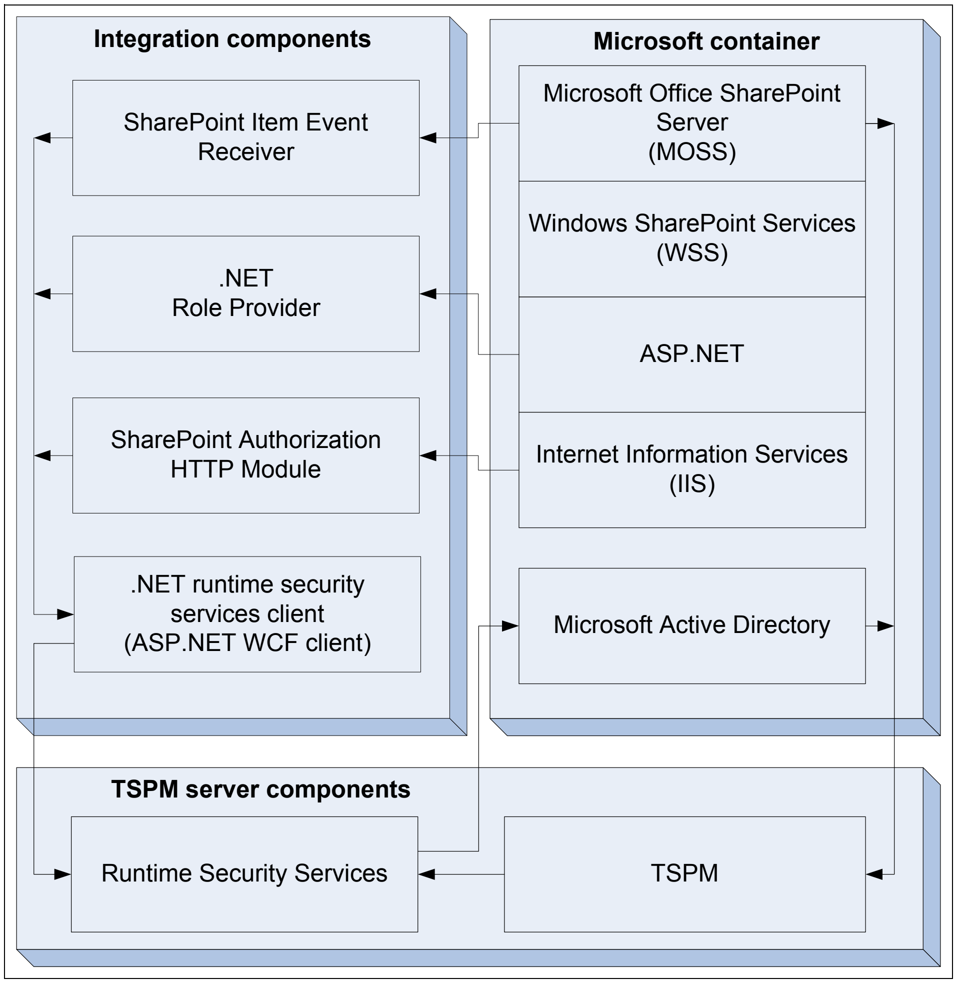

Integration components
SharePoint Item Event Receiver
.NET Role Provider
SharePoint Authorization HTTP Module
.NET runtime security services client (ASP.NET WCF client)
Microsoft container
Microsoft Office SharePoint Server (MOSS)
Windows SharePoint Services (WSS)
ASP.NET
Internet Information Services (IIS)
Microsoft Active Directory
TSPM server components
Runtime Security Services
TSPM

Figure 6-5 Microsoft integration architecture

An overview of each integration component, its function, and integration point is discussed below:

- .NET integration with the .NET runtime security service client

  This is the core component used by all the other integration components to evaluate authorization decisions and entitlements.

It can be executed directly from a Microsoft .NET application or through the web through ASP.NET, for example, as a SharePoint WebPart or within an ASPX page.

The .NET runtime security services client communicates through XACML over SOAP to call the RTSS web services, which are used to evaluate the authorization or entitlements request.

Details about the programmatic use of the .NET runtime security service client in a .NET application are provided later in this book.

- IIS integration with the SharePoint Authorization HTTP Module

  Integrating at the IIS layer, this component is configured in the Web.config of an ASP.NET as an authorization module. At this layer in the Microsoft environment, the component is able to authorize requests at the HTTP level, before the request is submitted to IIS for processing.

  This setup provides interception of requests during the processing, in which custom actions can be passed to IIS instead of the original request, such as redirection to an access denied or error page. The module uses the .NET runtime security services client for authorization requests.

  Although this component is SharePoint specific in its operations, the concept could be applied to write additional HTTP authorization modules for specific applications, or be generic modules that use the SubjectHandler and AttributeHandler interfaces of the .NET runtime security services client to populate the attributes required for security policy decisions.

  This component is generally used in conjunction with the SharePoint Item Event Receiver and both components are bundled together as the Enforcement Point for Microsoft SharePoint.

- ASP.NET integration with the .NET Role Provider

  This component is an implementation of an ASP.NET role provider that is executed through the .NET Role manager and uses the .NET runtime security services client for authorization and entitlements requests. By integrating at the ASP.NET layer though the Role manager, any existing Role manager aware application can be migrated to use Tivoli Security Policy Manager for role based authorization with minimal change or business impact. The Tivoli Security Policy Manager .NET Role Provider allows the use of rich constructs when authoring a security policy, which is generally not available when using Active Directory group memberships or an application’s native security for access control.

  Use of the .NET Role Provider requires the creation of a custom service application in the Tivoli Security Policy Manager console representing the roles for the organization and configuration the application to use the .NET Role Provider.

- MOSS integration with the SharePoint Item Event Receiver

  A custom Item Event Receiver provides pre- and post-evening processing within MOSS. When integrating at this level, fine-grained authorization decisions can be made using SharePoint specific actions and resources. This component uses the .NET runtime security services client for authorization requests.

  When registered for pre-events, this component can be used to authorize events running within the SharePoint workflow, which are system driven requests and are not available at the HTTP level, which are user driven requests. It can be used in conjunction with the SharePoint Authorization HTTP Module or in stand-alone mode.

  This component is generally used in conjunction with the SharePoint Authorization HTTP Module and both components are bundled together as the Enforcement Point for Microsoft SharePoint.

- Active Directory integration with Tivoli Security Policy Manager and RTSS

  During security policy authoring or security policy evaluation, Microsoft Active Directory can be used by accessing it as a LDAP repository. This scenario allows defining Application Roles using Active Directory groups and Rule Parameters using Active Directory user attributes.

  No specific configuration is required other than the standard host name, port, and an administrative account with access to the required Active Directory.

6.3.3 Microsoft integration using the policy life cycle model

Now that we have explored the integration points available within the Microsoft container, we can prepare to deploy Tivoli Security Policy Manager as part of the security landscape by following the policy life cycle model.

Policy modelling and simulation

The current and proposed deployment of .NET applications and their security configuration within your organization should form the basis for policy modelling and architecting the associated security policy needs.

Managing native security across multiple .NET applications can be extremely complex if they each implement their own security model, and some applications on their own, such as Microsoft Office SharePoint Server, are extremely complex regarding the management of fine-grained access control.

Traditionally, Microsoft based security has been implemented by adding users to groups and then permitting or denying access based on group membership, or explicitly permitting or denying individual users.

Using one of more of the integration components and the rich security policy constructs available in Tivoli Security Policy Manager, the traditional group based security of your Microsoft applications can be simplified and migrated to a combination of any or all of attribute, role, and identity based access control.

When using the .NET runtime security services directly, your policies can be represented in any form that fits your organizational needs, which can be mapped into a custom service application in the Tivoli Security Policy Manager console with associated security policy.

When using the .NET Role Provider, the security policy model looks similar to any existing role based models, but the Tivoli Security Policy Manager policies may be optimized, given the rich policy constructs available to make use of available attributes.

When using the SharePoint Authorization HTTP Module or SharePoint Item Event Receiver, the model is driven by your SharePoint deployment.

Importing resources and metadata

The Microsoft integration component that you use determines your ability to import or discover your resources and metadata. If your resources cannot be imported, they must be created manually in the Tivoli Security Policy Manager console.

When using the .NET runtime security services client directly, or when using the .NET Role Provider, the required service hierarchy must be created manually as a custom service in the Tivoli Security Policy Manager console.

- .NET runtime security services client

  The service hierarchy when using the .NET runtime security services client directly can be as simple as a single level, or as complex as your organization needs. Remember though that the more complex your service structure, the more likely it is that your policy modelling and simulation should be revisited to optimize your structure and policies.

  The hierarchy of the service created is extremely flexible, as the elements of the service are accessed directly from within your code that is calling the .NET runtime security services client. This action requires that the application developer has a working knowledge of the service structure created by the policy author to ensure that the correct elements are specified when making authorization decisions.

An example is shown in Figure 6-6 that represents the custom service as an organizational structure. Policies can be attached to any of the multiple points in the service, with the final authorization or entitlements evaluation involving the combination of the most specific policy plus any policies attached further up in the service.

```
National Healthcare Management
   |
   + ==> State and Territory
   |  |
   |  + ==> Regional
   |  |  |
   |  |  + ==> Hospital
   |  |  |  |
   |  |  |  + ==> Patient Records
   |  |  |  |  |
   |  |  |  |  + ==> Patient
   |  |  |  |  |  |
   |  |  |  |  |  + ==> ...
   |  |  |  |  |
   |  |  |  |  \/
   |  |  |  \/
   |  |  |
   |  |  + --> Clinic
   |  |  |  |
   |  |  |  + ==> Service Provider
   |  |  |  |
   |  |  |  + ==> ...
   |  |  |  |
   |  |  |  \/
   |  |  \/
   |  |
   |  + ==> Metro
   |  |  |
   |  |  + ==> ...
   |  |  |
   |  |  \/
   |  \/
   \/
```

Figure 6-6 Custom service for the .NET runtime security services client

- .NET Role Provider

  When using the .NET Role Provider, the service hierarchy is limited to a single level due to the flat nature of the .NET Role manager, although the .NET Role Provider does introduce support for an additional role context, which allows a single service to be shared among multiple .NET applications with their own role needs, but for management purposes can be contained within a single service.

  An advantage of using the Role Provider is that the application developer or systems administrator responsible for configuring the application's security does not need to be aware of any service structure or policies, only the roles that are required in the application's execution. The Role Provider, through the .NET Role manager, provides the roles for the user based on their Tivoli Security Policy Manager entitlements.

  An example of a single level role hierarchy is shown in Figure 6-7. Although the example shows a SharePoint use scenario, the Role Provider can be used with any ASP.NET application that supports the .NET Role manager.

```
SharePoint Roles Custom Service
   |
   + ==>Role 1
   |
   + ==>Role 2
   |
   + ==>Role 3
   |
   \/
```

  Figure 6-7   Custom service hierarchy for the Role Provider

  Further details about the Role Provider and creating a custom service are available in the integration documentation.

- Enforcement Point for Microsoft SharePoint

  Only when using the SharePoint Authorization HTTP Module and Item Event Receiver (the "Enforcement Point") can the service be imported. The import process is accomplished by using published SharePoint web services called from the Tivoli Security Policy Manager console to *discover* the SharePoint site collection.

The discovery results in a new hierarchical service being created that contains site, web, and list resources for the site collection that was discovered. Although list items are not discovered or displayed (due to the potential number of items that would make navigation difficult), they can be referenced within the policies.

When using the Enforcement Point, no native SharePoint security is required, and all access control is driven by Tivoli Security Policy Manager policy.

An example of a discovered SharePoint site collection that has been imported into Tivoli Security Policy Manager is shown in Figure 6-8.

```
SharePoint Site Collection
   |
   + ==>http://sps2007
   |     |
   |     + ==> Research Portal
   |           |
   |           + ==> California Claims
   |           |
   |           + ==> Cancer Treatment
   |
   + ==>http://sps2007:81
   |     |
   |     + ==> Quaterly Financial Reports
   |           |
   |           + ==> Quarterly Reports
   |
   \/
```

Figure 6-8 Discovered service hierarchy for the Enforcement Point

Further details about the Enforcement Point are available in the integration documentation.

Policy authoring

Aside from the capability to reuse security policy across multiple .NET applications, one of greatest advantages of integrating your .NET application security with Tivoli Security Policy Manager is the ability to use rich security policy constructs. The policy you author can continue to represent a traditional group based security model, common in many Microsoft environments if needed, but can be extended or redefined to include attribute based decisions, such as session data for fine-grained and dynamic authorization decisions.

All of the integration components use the .NET runtime security services client to perform their authorization and entitlements lookups and as such are able to access the standard attributes provided by the client and have the flexibility to provide custom attributes in the request, which can than be used in your security policy. Some integration components provide custom attributes as a core function of the integration and all integration components can populated custom attributes as needed.

- .NET runtime security services client

  When any of the integration components needs to call out to Tivoli Security Policy Manager for an authorization or entitlements decision, the .NET runtime security services client is used. In either call, the type of operation is being performed is sent as the *action-id* in the XACML request, which must correspond to an action defined for that service. When authoring your security policy, you must ensure that you select the appropriate action for which the policy applies.

  The .NET runtime security services client provides a pluggable framework that allows you to provide custom attributes for the resource being accessed, the actions being performed, and subject specific attributes, such as the user ID, group memberships, assigned roles, and an authentication token.

  Details about how to use this pluggable framework is described in detail in Chapter 8, “Application level integration” on page 213. All of the integration components below are able to make use of this framework to provide custom attributes for their corresponding services and security policies.

- .NET Role Provider

  The .NET Role Provider provides a mechanism for .NET applications to use the .NET Role manager to externalize authorization and entitlements decisions to Tivoli Security Policy Manager. Most of the calls performed by the .NET Role Provider are entitlements requests, such as GetAllRoles and GetRolesForUser. When an entitlements request is performed, the elements within the custom service that are accessible are returned and represented as roles to the Microsoft container. If your application uses the .NET Role manager methods, such as IsUserInRole directly, authorization calls are used instead.

  By default, the .NET Role Provider only supplies the action-id that is assigned programmatically to represent whether the request originated from a user or system driven process. User processes include user-centric calls, such as generating a role principal after their initial log in or attempting to gain access to a specific resource, while system processes are used for .NET Role manager requests, such as retrieving all roles within the service. You must author policy to cater for user and system driven processes using the action to differentiate between the two.

Due to the .NET Role manager's ability to cache a user's role principal, you should use dynamic attributes such as instance (not session) attributes with caution. For example, during the initial login, a role principal is created for the user representing their entitlements at that given point in time. If the security policy requires a custom attribute that was not assigned during the login, it is possible that user is not provided a particular role and because the role principal is cached, access is prevented to other resources in future requests that rely on a specific role.

An example custom attribute that would be static for the session but still significant for the decision could be the logon location, such as a public kiosk, department workstation, or VPN. The location may be significant in a security policy that controls the business role entitlements that prevent privileged operations from insecure environments. A custom attribute such as this one would be added to the request using the pluggable framework provided by the .NET runtime security services client.

Security policy for use with the .NET Role Provider generally uses Application Roles and possibly Rule Parameters for custom attributes that are significant for the entire session lifetime. If your Policy Modelling and Simulation from the policy life cycle model is completed thoroughly and your .NET application calls the .NET Role manager manually, it is possible to use entitlements calls (GetRoleForUser) for general, system-wide role assignments and use authorization calls (IsUserInRole) for context specific access. Applications such as Microsoft SharePoint, however, only use entitlements calls.

- Enforcement Point for Microsoft SharePoint

  The Enforcement Point for Microsoft SharePoint provides rich security policy constructs during policy authoring by providing a number of custom attributes that are populated in the XACML request. These attributes are provided to the .NET runtime security services client using the pluggable framework.

  The custom attributes added to the request by the Enforcement Point include resource attributes, such as the SharePoint web, site, list and list item name and ID being accessed, as well as action attributes, such as the function or operation being performed in SharePoint. The resource attributes allow your security policy to be structured around site resource based access and the action attributes to control what operations can be performed on the resource defined in the policy.

  The service for the Enforcement Point is automatically created by performing a service import, called a *discovery*. This process uses published web services provided by SharePoint to discover the resources and content of a site collection. During the service import, only resources up to the list level can be discovered, although list items can still be referenced in your security policies using the list item name or ID attributes.

An example of using SharePoint resource and action attributes could be a site collection for an educational institution. The site provides specialized document libraries for students and staff and a shared calendar accessible by any member (staff or student) of the school. The resource attributes can be use to control access to each resource in the site collection and the action attributes allow your security policy to control what type of operations can be performed on the resource, such as read only access (*view*), add content (*create*), update existing content (*edit*), remove items (*delete*), or any combination of those actions.

It is also possible to use the pluggable framework to provide your own custom attributes to the XACML request if your require additional custom attributes, such as document metadata.

Policy configuration

None of the integration components require specialized configuration at policy configuration time, only that which is required for any security policy specific items, such as application roles and rule parameters. If your security policy references custom attributes populated by the .NET runtime security service client, you must map these by referencing the attribute names in the XACML request.

The XACML attribute in which the custom attributes reside depends on the type of attribute. For example, SharePoint site collection data such as site, web, list and list item names are on the Resource XACML attribute, and the operation being performed is located in the Action XAML attribute. If you use the pluggable framework to provide your own custom attributes to the .NET runtime security services client, the configuration of your module determines the XACML attribute where the attribute names and values are populated. All attributes are in a standard location in the request.

In a Microsoft environment, typically all of the organizational data for users and even system resources is contained within an Active Directory. It may be necessary therefore to search the directory for additional attributes, such as a user’s department or title. In Tivoli Security Policy Manager, an Active Directory is treated in the same manner as an LDAP server, that is, to allow the retrieval of groups for Application Roles and runtime data for Rule Parameters.

After you have provided mappings for your Application Roles and Rule Parameters, the policy must be configured for runtime security services, as the .NET runtime security services client calls the RTSS web services for authorization and entitlements requests.

Policy distribution

After you have distributed your security policy to runtime security services, you have now completed the necessary steps for your .NET application to integrate using Tivoli Security Policy Manager for authorization.

The .NET runtime security services client used by the integration components does not have the capability for a local or remote mode of operation, such as a traditional Enforcement Point. This client uses the Authorization and Entitlements web services on the runtime security services distribution targets.

During configuration of the .NET runtime security services client, a runtime security services endpoint must be configured. The policy for the integration component must be distributed to this runtime security services endpoint.

In high load environments, additional runtime security services endpoints should be created and the configuration of the .NET runtime security services client updated to spread the load of requests to individual endpoints.

When calling the runtime security services web services, the .NET runtime security services client relies on Windows Communication Foundation (WCF), which does not support the configuration of multiple endpoints to achieve load balancing. If configuring individual instances of the .NET runtime security services client to a specific runtime security services endpoint is not desirable, consider implementing a load balancer in front of multiple endpoints and configure the .NET runtime security services client to use the load balancer as the endpoint. Ensure the appropriate SSL configuration is completed if necessary.

Policy enforcement

The integration component that you are using determines how the security policy is enforced, what users see in the event of an authorization failure, and if there are any cases where the Tivoli Security Policy Manager policy cannot be enforced.

- .NET runtime security services client

  The .NET runtime security services client provides you the lowest level of control over Tivoli Security Policy Manager policy enforcement in your .NET application. When using the client to integrate your .NET code, it is your code that serves as the policy enforcement point (PEP), using the result from the authorization or entitlements call for the enforcement.

  Your .NET application is responsible for handling the result appropriately, such as displaying the requested content or an appropriate access denied or error message to the user. Your .NET application should also output the required message data for audit and reporting needs.

The security configuration such as permissions for your .NET application is driven through your code, with Tivoli Security Policy Manager acting as the policy decision point (PDP).

- .NET Role Provider

  When using the .NET Role Provider, Tivoli Security Policy Manager acts as the PDP, but it is the .NET Role manager that becomes the PEP. The .NET Role manager queries its configured role provider to retrieve the list of roles for a user to build a Role Principal that it generally caches. When the user attempts to access a resource, the .NET Role manager checks to see if the Role Principal contains the required role; if not, it denies access.

  The .NET application using the .NET Role manager is responsible for displaying the appropriate content or messages to the user, and the security management, such as permissions and required roles, are also managed by the application directly.

  Microsoft SharePoint, for example, can be configured to use a role provider. When configured this way, permissions for accessing resources within a site are managed through the SharePoint site administration, and auditing data can be exported from the SharePoint site's database.

  Although the .NET Role Provider does provide trace messaging, runtime authorization decisions should be logged as necessary by the .NET application using the .NET Role manager. The .NET Role Provider may only be called at the initial creation of the role principal, and as such may not contain all of the resources that a user has requested access to during their session.

- Enforcement Point for Microsoft SharePoint

  The Enforcement Point for Microsoft SharePoint uses Tivoli Security Policy Manager for security management and for policy enforcement. The SharePoint Authorization HTTP module and the SharePoint Item Event Receiver are deployed to SharePoint and collectively perform the role of the PEP. Each of the modules is responsible for specific authorization decisions within SharePoint, but can be enabled or disabled individually.

  The security policies defined and attached to the discovered service in the Tivoli Security Policy Manager console provide the security management for SharePoint. No native SharePoint security permissions are required, although they can still be defined, but will be evaluated by SharePoint before the Enforcement Point is called.

Managing permissions within SharePoint directly is not recommended when using the Enforcement Point, as it can be difficult to resolve incorrect access failures and requires management in two separate locations, which can lead to inconsistencies. Performing security management within SharePoint when using the Enforcement Point is redundant and does not provide the rich security policy constructs that Tivoli Security Policy Manager offers.

When the Enforcement Point encounters a denial or error for a request, it redirects the user to the native SharePoint access denied or error page so that the user in unaware of the existence of Tivoli Security Policy Manager and their user experience is maintained.

The Enforcement Point calls the .NET runtime security services client on every request for a resource within the SharePoint site and outputs the result to the trace source. In some cases the Enforcement Point can only provide limited security policy enforcement due to the order in which the modules are called within the SharePoint workflow.

Auditing and reporting

All of the integration components provide auditing and reporting data through the use of an individual .NET TraceSource for each component. Generally, you only need to configure output for the integration component, but it may be necessary to output messages from the .NET runtime security services client if it is used directly in your .NET application or you need to perform a system diagnosis or trace.

The logging for each integration component is controlled by applying the necessary configuration in the `web.config` file of the .NET application. Control of the trace options, such as the level of detail and output location, is achieved through the use of the .NET trace classes in the System.Diagnostics namespace.

Each of the integration components output messages resulting from access decisions and operations, such as permitting and denying access or retrieving roles for a user and all roles in the system. Initialization and configuration messages is also output to validate your configuration and to determine version information.

The various output formats available in the .NET trace classes, such as Windows Event Log, text, or XML files, provide opportunities for central collection and analysis of messages. You can also configure the severity threshold of messages to increase or decrease the level of detail ranging from individual attributes, access denied messages, and only fatal errors. You should update the logging configuration to suit your collection and analysis needs, keeping in mine that excessive messages may lead to performance degradation.

Where the logging available from the integration components is not sufficient, or cannot be integrated with your collection and analysis applications, the runtime security services logs can be used for auditing and they can also be configured to suit your collection and analysis needs.

Examining the trace and audit logs closely is a best practice during the initial deployment of your .NET application integration to ensure that the correct attributes are being provided and the expected authorization or entitlements decisions match those expected for the resource being accessed. You should continue to monitor the data collected in your logs to ensure your security policies are sufficient for your business rules and that they continue to remain current and provide full coverage for your integration.

6.3.4 Conclusion

The Microsoft container provides a number of integration points to allow external authorization for .NET applications using Tivoli Security Policy Manager.

The use of Tivoli Security Policy Manager for delegated authorization provides rich policy constructs that are not normally available to traditional Microsoft Windows group based authorization and the use of Tivoli Security Policy Manager can also eliminate some of the complexities of managing individual native application security configuration.

An individual integration component or combination of components can be used to provide full coverage to all your .NET based applications in the Microsoft container.

6.4 Conclusion

This chapter discussed the container level approach and the value provided by externalizing security from applications using a container based approach. Several examples of container based integration were examined, including integrations with IBM WebSphere Application Server. Several examples of Microsoft based technology were also examined to show how Tivoli Security Policy Manager can be used to manage security for .NET based applications.

7

Database level integration

In this chapter, we introduce and describe database level integration with Tivoli Security Policy Manager.

Earlier in this book, we explored policy life cycle management using Tivoli Security Policy Manager; now we explore database level integration using this methodology.

In this chapter, we discuss the following topics:

- Concepts and benefits
- Database policy information point
- Database policy enforcement point
- Enterprise content management databases

7.1 Concepts and benefits

Databases are commonplace within almost every organization to satisfy the need to store, retrieve, report and analyze structured data, and this need increases every day. Whether a database is used directly by interfacing with the raw tables or indirectly by applications accessing the database programmatically, the value and sensitivity of the data must be considered.

Today, the volume of electronically stored information is continually increasing due to convenience and lower administrative, management, and storage costs when compared to manual paper records. Because more data is being retained for longer periods, the security of the data becomes an important consideration to the owners of the data. In some cases, regulatory compliance requirements may exist that impose significant fines for not adhering to the required practices.

Traditionally organizations have relied upon database administrators (DBAs) to implement manual checks and balances to ensure databases are secured to prevent theft, unauthorized changes, and unprivileged access, or to comply with regulatory requirements. This model has worked well and continues to do so for small server deployments with only a few databases. As the number of servers, databases, and tables increases, the impact of effectively managing these resources becomes prohibitive due to the time required to manually perform the operations, which increases the risk of error.

Given the complex nature of database security and the potential differences in security models between vendors, a DBA is required generally for each database type. By automating many of the manual steps required to secure and maintain their databases, organizations can focus on the value that the data can provide for reporting or implementing business rules, rather than the management of the databases themselves.

The solution

Tivoli Security Policy Manager provides support for managing native database security and support for specialized database systems, such as content management databases. It can also use information in databases to enable context-rich and attribute based authorization policy decisions.

In the context of Enterprise Content Management (ECM) systems, a significant amount of high value corporate data is handled and stored, although just by using an ECM, the data is well secured.

By managing centrally the security for databases, the level of effort required to ensure cross system compliance, as well as the complexity associated with managing multiple database products from different vendors, can be significantly reduced. Updates or alterations to corporate wide policy can be applied rapidly to new and existing systems, reducing the management impact and implementation time frames.

7.2 Database policy information point

Business rules often have complex logic and pre-conditions that must be met to permit a particular operation, as opposed to a simple allow or deny decision based on group or role membership in a typical user repository. In the context of authorization and entitlements management, it may be necessary to query external data sources to retrieve additional information to assist in making the decision, or to retrieve specific values against which to compare pre-conditions.

For example, a business role for a bank teller may allow the teller to approve an overdraft up to a certain value without requiring the next organizational level. If this policy is translated into traditional group or role based access, the hierarchy required becomes increasingly complex to manage effectively and does not provide flexibility to change the approval amount without needing to modify a number of role or group memberships.

This is one such example where information from an external resource, such as a record in an overdraft limits table from a database, can assist in the translation of a business rule to a security rule. In this example, only a record in the database needs to be updated, as the mapping between the user (or group) and their approval limits does not need to be hardcoded or fixed to their profile.

The capability to retrieve business rule context data at execution time from a database can reduce the number of individual user and group permissions and role hierarchy, helps improve compliance by simplifying management, and provides fine-grained attribute based authorization decisions.

In this section, we explore the capabilities of Tivoli Security Policy Manager to use data stored in traditional databases to author policies incorporating attribute data from a business context, not just traditional user and group based security constructs.

7.2.1 Foundation for integration

In many organizations, there are often multiple sources of authoritative information, specific to an application domain. For example, an organization may maintain a human resource management system containing social security numbers, employee IDs or serial numbers, and a home address, a corporate directory containing company emails, cell phone numbers, and manager names, and potentially many other repositories.

These systems may not be linked or synchronized in a well formed or logical manner, as the data in these repositories may not necessarily be related or meaningful when used in a stand-alone application context. There may, however, be value in making attribute based authorization decisions using the context data for an employee combined with attribute data from two or more sources.

The use of databases as a policy information point enables the capability to use attribute data from multiple sources in a single authorization decision.

7.2.2 Integration with Tivoli Security Policy Manager

In 3.1.1, "Tivoli Security Policy Manager components" on page 64, we introduced the concept of a policy information point (PIP), which facilitates integration and use of attributes from existing sources. In an organization, these sources may include identity management systems, identity and attribute repositories, and application databases. To adhere to business rules, it may be necessary to query an external system to validate a certain value to ensure pre-conditions or business requirements are met before a request or operation is authorized.

Tivoli Security Policy Manager supplies a database PIP allowing the authorization service to base decisions off of information that is stored in databases, which can be queried at the time of authorization.

Database PIPs can be used to execute standard or complex queries on a configured database to retrieve attribute data that can then be referenced in security policy as Rules. Only SELECT statements can be executed, because PIPs are intended as a source of data; they are not allowed to update the data as part of an authorization decision.

To extend the flexibility and capabilities of retrieving attribute data, database PIPs support *PIP chaining*, which allow the output of one PIP as input into any other PIP, not just databases. Chaining provides the capability to use unrelated or application specific data to build rich attribute based security policy.

Several high level processes for using a database PIP are:

- Using the Tivoli Security Policy Manager console, the Database Attribute Query wizard helps you create a *Database Attribute Query* within the required IBM Tivoli runtime security services.

  The Database Attribute Query wizard helps you write queries using the following form

  ```
  SELECT column name FROM table WHERE column name EQUALS request
  attribute
  ```

  Alternatively, by customizing the `security-services.xmi` file, you can author your own complex database queries in the form of the following examples:

  ```
  SELECT column name 1 FROM table 1 WHERE search value 1 IN (SELECT
  column name 2 FROM table 2 WHERE search value 2 = request attribute)
  ```

  and

  ```
  SELECT column name FROM table WHERE search value 1 = {request
  attribute 1} AND search value 2 = {request attribute 2}
  ```

- As necessary, chain any existing PIPs together with the database PIP, which allowing you to create nested queries, pattern matching, external lookup, and custom functions that provide the attributes necessary to evaluate the policy decision.
- In an Authorization Policy, add a *Rule* that represents the use of the rule parameter where the required match of the evaluation corresponds to the value returned by the Database Attribute Query or final query in the chain.
- During policy configuration, map the rule parameter referenced in the authorization policy to the Attribute query in the Policy Configuration window. Specify the return attribute section and name of the value from the database query.

After your policy has been configured and deployed, the policy decision point (PDP) will invoke the database PIP during policy enforcement. Multiple Database Attribute Queries can be created, allowing the use of multiple attributes from the same or different external databases in an authorization policy. These attributes can be added as multiple rules in a single authorization policy, a single rule in multiple authorization policies, or a combination of both, and can be combined with PIP chaining, further enhancing the policy.

7.2.3 Database integration using the policy life cycle model

When using databases as PIPs, we are not necessarily implementing the complete policy life cycle model, as PIP database integration is a facilitator of user based attributes that are used to make authorization decisions. The process for evaluating the need for, and use of database PIPs, is observed in some of the phases of the policy life cycle model.

Policy modelling and simulation

In this phase, the notion of the actual PIP implementation or the number of PIPs required is not significant, as the focus here is to understand the business rules and the data available or required to apply these business rules.

Understanding what attribute data is required to make the authorization decisions and where this data is stored leads into the next steps of the policy life cycle model, which is when the use of PIPs will be realized.

Importing resources and metadata

The step of the policy life cycle model is not applicable to the use case of database PIPs, as it is in fact the records within the target database that will be used in Tivoli Security Policy Manager using a PIP to enable attribute based authorization decisions in the security policies.

Policy authoring

In this step, you do not write policy for the PIP; rather, your authorization policies contain Rules that, during policy configuration, are mapped to the database PIP.

Policy configuration

Most of the implementation of PIPs is completed in this phase. You define PIPs during policy transformation, that is, transforming an abstract policy to add binding information so that policies can be communicated to PDPs in terms of real IT infrastructure. This generally occurs during the configuration stage of transformation.

It is in this phase that the PIPs, Rule Parameters, and Rules for the authorization policy are defined. During the Policy configuration action, the mapping between the Database Attribute Query and Rule Parameters are mapped, allowing their use in the Rules of the authorization policy.

Policy distribution

The database that the PIP is configured to execute the queries against does not require the security policy to be distributed to it, because is the source of attribute data for authorization decisions.

Policy enforcement

During policy enforcement, when an authorization decision is required that relies upon additional attribute context data, the PDP invokes the PIP for each of the Rules that use mapped Rule Parameters that are mapped to Database Attribute Queries. When the Database Attribute Query is executed, it returns attribute values from the configured external database source, allowing the PDP to use the attributes in the authorization decision.

After the PDP has evaluated whether to permit or deny access, the policy enforcement point (PEP) is responsible to enforcing the decision.

Database security: You need to ensure that the database that holds your additional attribute context data is properly secured and cannot be hacked or impersonated by a faked database.

Auditing and reporting

When the database PIP is invoked, it executes the appropriate SQL query against the configured database. Permissions to execute the query must already be configured in the database and as such, auditing and reporting is achieved by monitoring the appropriate access control records on the target database.

The Tivoli Security Policy Manager Runtime Security Service (RTSS) logs contain invocation details of the query and errors encountered during the execution.

7.2.4 Conclusion

Many organizations store application or process specific data spread across multiple disparate systems. When evaluating business decisions, it is often valuable to use attributes and context from more than one source, which is outside the realms of traditional role or group based security concepts.

Tivoli Security Policy Manager enables business rules to be implemented and evaluated using multiple sources of data by exploiting database policy information points and potentially chaining multiple PIPs when authoring and evaluating authorization policies.

7.3 Database policy enforcement point

There is an ever increasing need to store, retrieve, and analyze electronic data in almost every organization while driving value and reducing costs. When you consider a database, it is not only the hardware and software that makes it expensive, but the skilled professionals needed to manage the database as well.

The cost of database administration can greatly exceed the cost of the database software and hardware, so it is critical that the database administrator's time be used effectively and efficiently. When you consider the number of types of databases, such as active, analytical, cloud, data warehouse, distributed and document oriented, it soon becomes apparent that the management of these databases consumes considerable time and therefore costs.

Security management of databases and the resources they protect is a critical component of an organization's overall database management process. A security breech, theft of information, or accidental deletion of data might not only impact the business from a brand image or trust perspective, but there may be regulatory compliance requirements that impose significant fines for not adhering to required practices.

In this section, we explore the capabilities of Tivoli Security Policy Manager to manage native database security and provide central security policy management, distribution, and enforcement.

7.3.1 Foundation for integration

Typically, most databases provide their own security implementation to protect the integrity of the data stored in them. These implementations may not be consistent or compatible between different database vendors and the tools and utilities for these databases may not be able to be used across products. Generally, after a user has been authenticated, an authorization check is performed. Given the high volume of transactions that databases need to perform, the processing time for authentication and authorization process must be minimal.

It is not uncommon for an organization to maintain database management systems (DBMS) from multiple vendors because, generally, each business application they use in the workplace includes its own particular database type or version. Having a heterogeneous database environment requires database administrators (DBAs) to gain skills, and a significant portion of their time is spent ensuring that consistent database security is applied to all of the DBMSs they manage.

As the number of DBMSs increases and the amount of data in them increase, so too does the time and cost. In any environment, as the workload increases, the risk of error and inconsistency also increases, and centrally controlled and automated processes can help drive compliance, efficiency, and reduce error, risk, and cost.

7.3.2 Integration with Tivoli Security Policy Manager

Tivoli Security Policy Manager reduces the impact of maintaining database security by providing centralized policy management and distribution, which allows DBAs to spend less time administering the system and more time focusing on other activities that benefit the business.

The database security policy management provided by Tivoli Security Policy Manager is achieved by executing Structured Query Language (SQL) queries to the databases that have been configured and imported. Tivoli Security Policy Manager applies coarse-grained authorization policy and permissions to databases by executing GRANT and REVOKE queries, which correspond to Permit and Deny evaluations in authorization policies.

To ensure that the transactional capabilities of databases are not compromised, during authorization the database does not make a local or remote call to RTSS; rather, it uses the native GRANT and REVOKE permissions applied by Tivoli Security Policy Manager.

When managing a database with Tivoli Security Policy Manager, no local or remote RTSS is included in the system landscape; the DBMS itself acts as the PDT, the PDP, and the PEP. Using a PIP in the context of database security policy is not possible, as this would require a connection to an RTSS, which may impact performance within a high volume transactional database.

Support for DB2 is included when Tivoli Security Policy Manager is installed; other databases may require database specific plug-ins.

7.3.3 Database integration using the policy life cycle model

Now that we have explored database security policy management within Tivoli Security Policy Manager, we can follow the deployment within the policy life cycle model.

Policy modelling and simulation

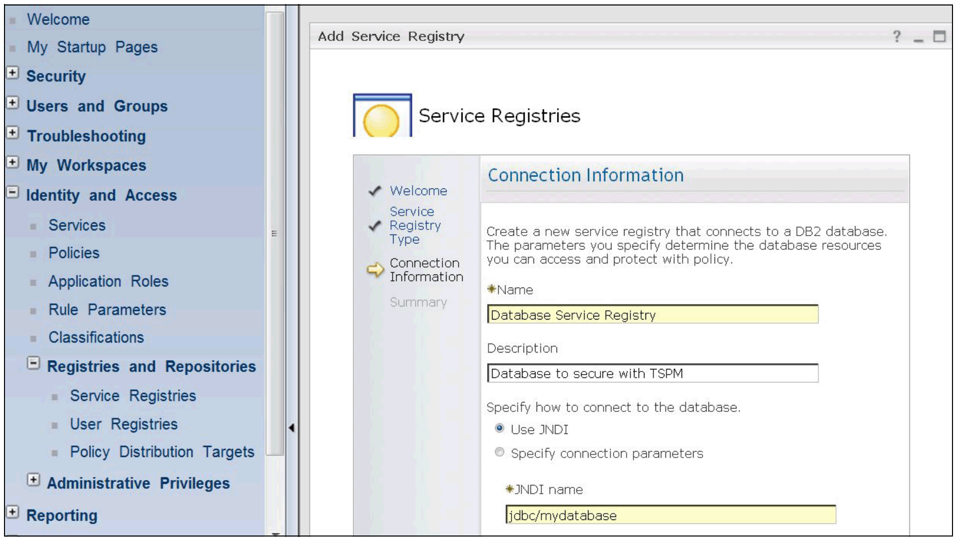
Before a database schema is built, the structure is typically *normalized* to ensure the data is stored in the most efficient layout for storage, retrieval, and maintenance. This process is quite similar to the policy modelling step of resources in the policy life cycle model. It is unlikely the database schema is able to be changed, because the available resources are already defined; however, user access control should be modelled and simulated.

As part of this stage, the available security constraints for the database should be reviewed to ensure that this is the most efficient and effective way to meet the needs of the organization and can be represented in Tivoli Security Policy Manager. For authorization decisions, the security policy can only contain application roles that map to users and groups provisioned into the database. When modelling, fine-grained attribute based authorization decisions should not be considered, only those that equate to GRANT and REVOKE permissions on the database resources, including tables, views, functions, and procedures.

Importing resources and metadata

With the modelling and simulation of the database resources and security constraints completed, the resources must be imported into Tivoli Security Policy Manager so that security policy authoring can be achieved. Databases that have their security policy managed by Tivoli Security Policy Manager are represented as a Service Registry in Registries and Repositories, and all individual databases must be imported individually. When importing a database, you can either use a data source on WebSphere Application Server or specify the connection parameters for the JDBC driver directly.

The use of a JNDI source is shown in Figure 7-1. The data source definition must contain a user that has the appropriate permissions to the resources you need to discover with the registry.



Connection Information

Create a new service registry that connects to a DB2 database. The parameters you specify determine the database resources you can access and protect with policy.

*Name: Database Service Registry

Description: Database to secure with TSPM

Specify how to connect to the database.

[✓ Use JNDI]

Specify connection parameters

*JNDI name: jdbc/mydatabase

Figure 7-1 Importing a database as a Service Registry using JNDI parameters

Now that you have created the database service registry, you must discover the available resources from the database by using the Import function of the Services window. You must select the Service Type **Services from a service registry** and then select the source from the list of service registries you have added. Next, you can select which resources you want to discover, including Tables, Views, Procedures, and Functions, and you must specify which Schema to search and, optionally, specify the resource name and whether to exclude system tables from the query.

Next, a connection is opened to the database to discover the resources you have specified in your search. During the discovery, each of the discovered resources are displayed as a service that you can individually select, allowing you to specify which resources are imported into the database service in Tivoli Security Policy Manager. You optionally change the Service Name that is displayed in the service hierarchy in Tivoli Security Policy Manager and you can also change the description, which, by default, displays the metadata for the resource type that the service represents.

After selecting which resources to import, you can optionally classify the service. By classifying all your databases in the same classification, you can easily apply the security policy across multiple databases.

When you have completed the import process, each of the resources discovered will be displayed as a child service of your database. An example of a resource structure is shown in Example 7-1, which includes Tables and their columns, Views, Functions, and Procedures.

Example 7-1 Database resource structure

```
Databases
   JKInsurance
      calculatePremiumFunc
      processClaimProc
   AccountsTbl
      accId
      accNum
      accTypeId
   AccountTypeTbl
      accTypeId
      accTypeDesc
   PoliciesTbl
      policyId
      policyNum
      customerId
      agentId
   CustomersView
      customerId
      customerName
      stateOfResidence
      accId
```

The imported resource example as shown in Example 7-1 includes multiple resource types from the database, each represented as a child service. Table 7-1 shows the mapping of each resource type to the corresponding child service.

Table 7-1 Database nodes in the services listing

Nodes in services listing	Example resources
Function	calculatePremiumFunc
Procedure	processClaimProc
Table	AccountsTbl, AccountTypeTbl, PoliciesTbl
View	CustomersView

After you import your database resources into Tivoli Security Policy Manager, you can author the policy to be applied to those database.

Policy authoring

Due to the high-speed transactional nature of databases, it is undesirable to perform an external call during processing to establish additional attributes or to have an external PDP perform the authorization decision. For database services, this means that only Application Roles that map to local users and groups for the database are supported. If rules are added during policy authoring, the conditions of the rules have no effect, and the authorization decision of the first rule will be applied.

When authoring your policies, you can only attach a security policy to the individual discovered resources, not the top level database. This situation occurs because of how the authorization policy is transformed to GRANT and REVOKE statements. You can select multiple services from the same or different databases, allowing you to consistently apply the same native permissions.

For the Services to which the policy will be attached, you can configure the SQL actions that the security policy manages and applies. The available SQL actions that can be managed by Tivoli Security Policy Manager are DELETE, EXECUTE, INSERT, SELECT and UPDATE, but you must ensure that the action you select is appropriate to the resource type. For example, the EXECUTE action is not applicable for table based operations, but can be selected during policy authoring.

During the Application Roles selection, you can add a new role as part of the authoring process or use roles that have been created previously. Although the notion of role in Tivoli Security Policy Manager is generally agnostic to the PDT and is determined at authorization time, for databases, the user or group to which the role is mapped must be local to the database itself. For some databases, this may be the local operating system accounts, or users and groups defined in an access control table, managed by the database itself.

The built-in application role of Any user equates to PUBLIC or ANONYMOUS, depending on the database implementation, and represents users who have not authenticated to the database. The Any authenticated user role represents PUBLIC access for users who have authenticated but where an individual user or group has not been explicitly added.

The last step in policy authoring is to configure the Authorization Decision for the security policy in which the Rules decide whether the evaluation of the rule permits access or denies access. For a database, a Permit policy acts like a GRANT query in SQL and a Deny policy acts like REVOKE query.

To ensure that the transactional capabilities of databases are not compromised, rules are not applied when the policy is configured and distributed.

Policy configuration

With your authorization policies attached to your database resources, you can now configure the policy specific to database resources. Before configuring your policy, you must add your database as a PDT using the add process for PDTs in Registries and Repositories. You then select your database as the PDT and also as the PDP authorization type.

The status for the Authorization configuration target will be listed as No configuration required. During policy configuration in Tivoli Security Policy Manager, an authorization policy normally requires a configuration of application roles and rule attributes. However, for an authorization policy attached to a database resource, the standard policy configuration is not applicable, because there are no valid rule attributes in a database GRANT statement, and the default application role mapping is achieved by querying an LDAP group registry, of which a database would have no integration capability. Calling external sources during authorization decisions may also compromise the transactional nature of a database.

Instead, an additional configuration target for your database will be listed that provides an alternative user interface for user and group application role mapping. When you select this configuration target and select *Configure*, the list of application roles that have been added in authorization policies attached to the database are displayed. You can now specify the mapping of the role names to users and groups known to the database.

Typically user repositories for a database are internal to the database itself, or may use known sources such as the local operating system. As such, it is not possible to provide a search capability for all the potential repositories due to database installation and configuration options. You must manually specify the name of the user or group that maps to the role referenced in the policy. The users and groups that you map must be valid within the database's authentication and authorization context and depend on the database installation.

If your policy includes rules as part of the policy evaluation, these rules will be ignored and therefore no option will be displayed to configure the rule mapping attributes. The Access Decision for the policy will only evaluate to the decision of the first rule.

Policy distribution

During the policy distribution process, the policies and role mappings for the database PDT are converted into database queries and then executed, causing native SQL permissions to be applied to the database resources.

The distribution is achieved through the use of Java Database Connectivity (JDBC) calls to the database using the connection parameters you specified when adding the PDT for the database. During distribution, you can view the resultant effective authorization policy converted to GRANT and REVOKE statements, as shown in Example 7-2.

Example 7-2 Sample effective authorization policy

```
GRANT DELETE ON TABLE DSRDBM01.ACCOUNTS TO GROUP administrators
GRANT INSERT, UPDATE, SELECT, DELETE ON TABLE DSRDBM01.ACCOUNTS TO USER dbadmin
GRANT SELECT ON TABLE DSRDBM01.ACCOUNTS TO USER dbadmin, PUBLIC
REVOKE INSERT, UPDATE, DELETE ON TABLE DSRDBM01.ACCOUNTS FROM PUBLIC
```

After the policy has been distributed, it comes into effect immediately. Existing connections to the database may need to be recycled to ensure that permissions are reloaded for the resource in the database being accessed.

Policy enforcement

After you have distributed the policy to your database, the effective GRANT and REVOKE SQL permissions are in effect. When database security policy is managed by Tivoli Security Policy Manager, the database itself acts as the PDP and PEP using the native database permissions. If no Tivoli Security Policy Manager security policy has been distributed, the existing database access control is used.

When a database client attempts to execute a query, stored procedure, or access view, the native database permissions that have been transformed from the eXtensible Access Control Markup Language (XACML) security policy from Tivoli Security Policy Manager evaluates whether access is granted. Notification to the client about the success or failure of the action they performed is not changed when using Tivoli Security Policy Manager, allowing the use of existing client error handling with no need to modify existing applications or database consumers.

Auditing and reporting

When a database security policy is managed by Tivoli Security Policy Manager, the database itself acts as the PDP and the PEP. Because there are no external components from Tivoli Security Policy Manager included in the authorization policy evaluation, all audit and reporting capabilities rely upon the configured database.

In most enterprise DBMSs, the deployment tools include an audit facility that allows you to monitor data access and provides information needed for subsequent analysis. Auditing can help discover unwanted, unknown, and unacceptable access to the data, as well as keep history records of the activities on the database system. Refer to the auditing and reporting capabilities of your DBMS to review the authorization policy evaluation results for the policy you have distributed.

Any messages or errors encountered during the import of database resources and metadata, policy authoring, policy configuration, or policy distribution continue to be available in the console message and error logs.

7.3.4 Conclusion

The data organizations store in their databases not only represents significant value to the business in terms of operational readiness and continuance, but the data can also be used in analytics and reporting. Many organizations maintain multiple data sources and require efficient systems to manage the security policy and access control, allowing DBAs to spend less time administering the system and more time focusing on other activities that benefit the business.

Tivoli Security Policy Manager reduces the manual and potentially error prone tasks of defining and applying database security policy by providing a centralized database security policy management solution, which has the effect of reducing security management costs and ensures that the required policies are applied consistently to all your database resources.

7.4 Enterprise content management databases

Most companies operate with strict content management requirements to help them control content and automate content related processes. As unstructured content grows exponentially, you need content management to capture, store, manage, integrate, and deliver all forms of content across your enterprise.

Having a Enterprise Content Management (ECM) system in place assists companies in mining for the right information at the right time to make the right decision.

Deployment of an ECM helps organizations meet regulatory and legal obligations associated with records and establish retention periods for all information. As the amount of content grows, Information Lifecycle Governance (ILG) becomes an important factor when defining business processes and security requirements to lower both cost and risk.

In this section, we describe the integration with the IBM Enterprise Content Management solution to manage the security policy of resources and content stored within the ECM.

7.4.1 Foundation for integration

An ECM system includes a number of features to store, archive, and dispose of electronic resources and data. These resources are generally document oriented databases and represent a high value to organizations, and are typically captured through manual or automated document imaging processes. Processes such as enterprise report management, standardization, and consolidation, as well as content analytics, are common features of an ECM. These processes enable you to search, assess, organize, and analyze large volumes of content.

A significant consideration for these systems is the security of the data. This security is not just the physical security, but is user centric, including who can access what data and under what special conditions may someone gain access to which they would not normally be entitled.

Consider a business rule within the health care industry where doctors on duty have access to Electronic Health Records of an inpatient, but only during their working hours. This situation may exist for a number of regulatory or privacy reasons, but generally makes logical sense: You do not want someone looking at your medical records if they are not treating you.

Any person in the role of Doctor on duty should be granted access to documents and folders classified as Electronic Health Records for patients whose status is inpatient, where the time of the day matches the working hours for the doctor. This business requirement for accessing content is based not only on Doctors (the role), but also the content in the ECM (the resources), the time of day (the rule), and could also include the level of access (the action).

The business rule in this case requires dynamic security policy using contextual attributes for authorization decisions. Tivoli Security Policy Manager provides the capability to manage security policy for an ECM with attribute based policy constructs.

7.4.2 Integration with Tivoli Security Policy Manager

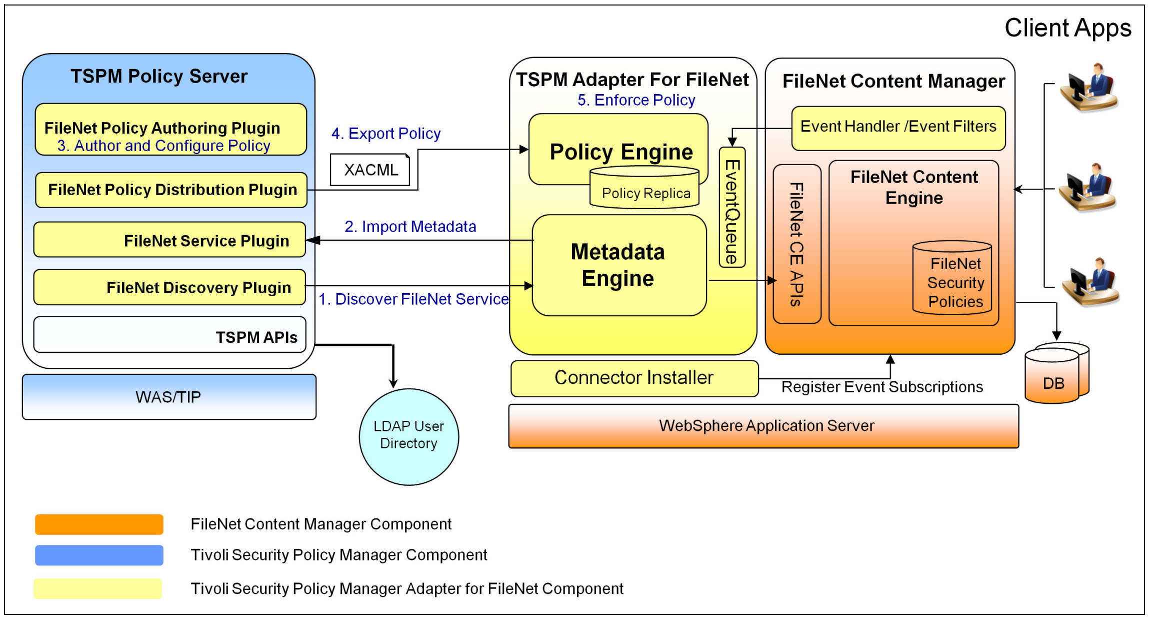
Tivoli Security Policy Manager provides native security and access control list (ACL) integration with the IBM FileNet® Content Manager, simplifying the maintenance and management of FileNet security. This setup enables the authoring of policy within Tivoli Security Policy Manager that is meaningful within an IBM FileNet ECM environment and provides support for security constructs that represent specific permissions, content resource classification, and access control.

Integration with IBM FileNet is achieved by using two components.

- IBM FileNet Plug-ins for Tivoli Security Policy Manager
- Tivoli Security Policy Manager Adapter for IBM FileNet

The Tivoli Security Policy Manager Adapter for IBM FileNet is a J2EE application, which is deployed on the IBM FileNet Content Manager. The adapter acts as the PDT and the PEP and is able to consume and transform XACML policy from Tivoli Security Policy Manager and enforce authorization decisions without the direct use of a runtime security services client or server.

An overview of the integration components is illustrated in Figure 7-2.



Client Apps
TSPM Policy Server
FileNet Policy Authoring Plugin
3. Author and Configure Policy
FileNet Policy Distribution Plugin
FileNet Service Plugin
FileNet Discovery Plugin
TSPM APIs
WAS/TIP
4. Export Policy
XACML
2. Import Metadata
1. Discover FileNet Service
LDAP User Directory
TSPM Adapter For FileNet
5. Enforce Policy
Policy Engine
Policy Replica
Metadata Engine
EventQueue
FileNet Content Manager
Event Handler /Event Filters
FileNet CE APIs
FileNet Content Engine
FileNet Security Policies
DB
Connector Installer
Register Event Subscriptions
WebSphere Application Server
FileNet Content Manager Component
Tivoli Security Policy Manager Component
Tivoli Security Policy Manager Adapter for FileNet Component

Figure 7-2 FileNet integration architecture

In Figure 7-2, Tivoli Security Policy Manager uses the IBM FileNet Discovery Plug-in (1) to connect to the IBM FileNet Content Manager and to discover applications that are available in the content repository. After the author selects the desired application, the IBM FileNet Service Plug-in (2) connects with the Metadata Engine in the Tivoli Security Policy Manager Adapter for IBM FileNet to import the application model and metadata. The IBM FileNet Policy Authoring Plug-in (3) uses the application metadata to help author content entitlement policies. Finally, the IBM FileNet Policy Distribution Plug-in (4) distributes the policy in XACML to the Policy Engine component of the adapter. The Policy Engine (5) is ultimately responsible for enforcing the policy.

The Tivoli Security Policy Manager Adapter for IBM FileNet contains the following integration components:

- Discovery Plug-in

  The Discovery Plug-in (not explicitly shown in Figure 7-2) exposes application programming interfaces (APIs) to discover IBM FileNet applications. The IBM FileNet plug-ins for Tivoli Security Policy Manager use these APIs for discovery of hosted applications.

- Metadata Engine

  The Metadata Engine exposes the metadata extraction as APIs. The IBM FileNet plug-ins for Tivoli Security Policy Manager use these APIs to import metadata for content.

- Policy Engine

  The Policy Engine exposes APIs to accept XACML policies. The IBM FileNet plug-ins for Tivoli Security Policy Manager use these APIs to distribute policy.

The IBM FileNet adapter uses the IBM FileNet Content Engine (CE) APIs to update the FileNet Security Policies' access control lists (ACLs) for content stored in the IBM FileNet Content Manager.

To dynamically update ACLs based on changes in the content repository, the adapter exposes an EventQueue API. The EventQueue is populated by the Event Filter deployed on the IBM FileNet Content Manager. The Event Filter listens for change events in the IBM FileNet Content Manager, and forwards the relevant events to the EventQueue. The Policy Engine evaluates the change events to determine if the ACL of the event source object must be updated as a result of the change.

The Tivoli Security Policy Manager Adapter for IBM FileNet is an example of how you can use the Tivoli Security Policy Manager Software Development Kit (SDK) to build custom endpoint integration components. Check with your Tivoli Security Policy Manager representative for the availability of plug-ins for your ECM product, including IBM FileNet.

7.4.3 ECM integration using the policy life cycle model

Now that we have explored the integration components of the Tivoli Security Policy Manager Adapter for IBM FileNet, we can prepare to deploy and use the integration within the policy life cycle model.

Policy modelling and simulation

When deploying an ECM, typically the structure of the content resources, their classification, and required security levels are implemented within the system itself.

When Tivoli Security Policy Manager is used to manage the security policy of IBM FileNet, the modelling of roles, resources, and classifications is typically already normalized, allowing us to prepare to import the resources and metadata. Before implementing any new components into the system architecture, review the existing modelling to ensure policy optimization.

Importing resources and metadata

The IBM FileNet Discovery Plug-in provides the mechanism to discover applications on the IBM FileNet Content Manager. The Metadata Engine running on the IBM FileNet server exposes the services as a set of URL endpoints that the IBM FileNet Discovery Plug-in uses to import the Application Model and metadata.

The Metadata Engine uses the Model extraction services to extract the model and the metadata from the IBM FileNet Content Manager using the IBM FileNet CE API, as shown in Figure 7-3. These services are used by the Tivoli Security Policy Manager IBM FileNet Discovery Plug-in to import the models and metadata.

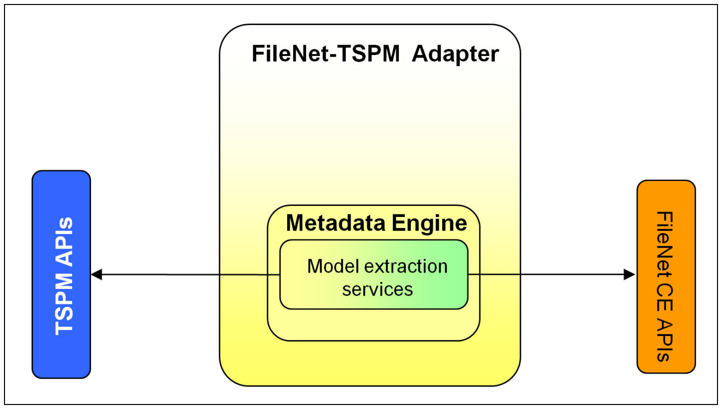


Figure 7-3 Model extraction services

After an application is selected, the available metadata and model information for the application is imported as a service into the Tivoli Security Policy Manager console, as shown in Figure 7-4.

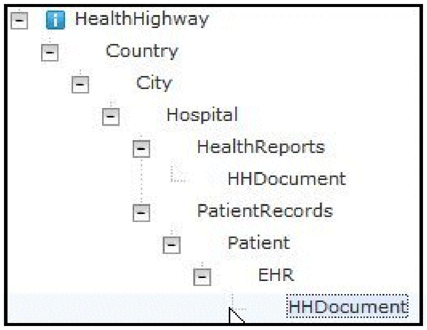


Figure 7-4 Metadata hierarchy within Tivoli Security Policy Manager

After the models are imported into Tivoli Security Policy Manager, the hierarchy appears as a service within the console. The metadata provides instance level values for authoring policy.

Policy authoring

Authoring a policy using IBM FileNet application semantics involves the following steps.

- ► Identifying resources
- ► Creating application roles
- ► Specifying conditions

To assist in authoring meaningful policy for IBM FileNet, the imported application model and metadata enables the creation and maintenance of policy using application domain semantics rather than XACML semantics. This task is achieved by using the custom policy authoring plug-ins.

Identifying resources

The resource selection widget lets you refine the resource selection and is provided at both the policy and rule levels. Resources are modelled as services and can be specified globally or manually. In the IBM FileNet plug-ins, a resource represents a service and the allowable actions in a standard authorization policy.

For example, a policy can be authored for all (*) cities or only Boston, as shown in Figure 7-5.

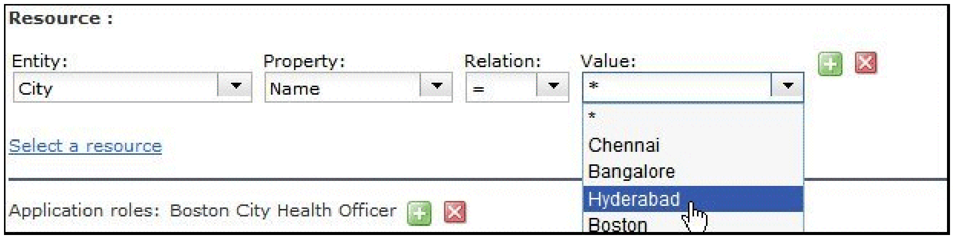


Figure 7-5 Resource selection widget

The Entity drop-down menu specifies the resource name, Property specifies the property of the resource, Relation specifies the operator, and Value specifies the value of the property. These values are populated from the metadata imported from IBM FileNet Content Manager.

Creating application roles

For IBM FileNet, application roles are created using the same logic as when authoring a standard authorization policy. Roles are mapped to users or groups defined in the LDAP registry during the policy configuration stage.

Following our previous example of a health care context, application roles that may be appropriate for the health care industry may be doctor, nurse, lab technician, and so on.

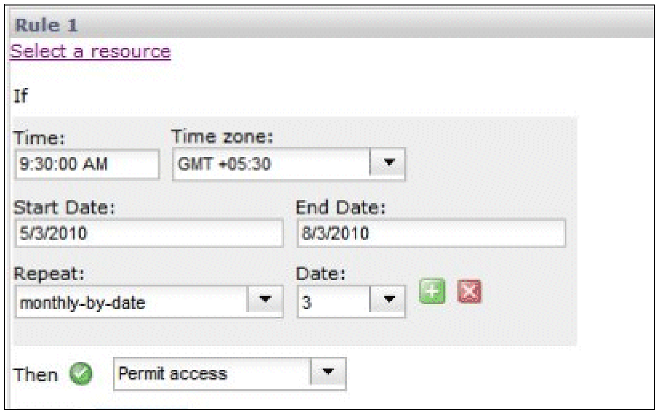
Specifying conditions

The IBM FileNet plug-ins include support for additional rule types, including the capability to repeat a time of day access policy with a timer widget. You can also author a policy for specific resources that were discovered during the policy import.

In our healthcare example, additional conditions that could be added to a policy in the form of rules include:

- Resource restriction conditions for a specific city name. For example, City Name = Austin.
- Environmental conditions for a specific time when the policy should be enforced. For example, between close of normal business hours for the organization.

An example of a time based environmental policy that is activated on the third day of every month is shown in Figure 7-6.



Rule 1

Select a resource

If

Time: 9:30:00 AM

Time zone: GMT +05:30

Start Date: 5/3/2010

End Date: 8/3/2010

Repeat: monthly-by-date

Date: 3

Then Permit access

Figure 7-6 Configure environmental conditions

With policy authoring completed, you can now proceed to configure and distribute the policy to IBM FileNet.

Policy configuration

When configuring policy for IBM FileNet, your application roles and rule parameters must be mapped from the abstract references to the specific physical resources.

During application roles mapping, any new or existing roles must be mapped to the users or groups provisioned in the directory server. The directory server must already be configured in the User Registries for Tivoli Security Policy Manager.

Any rule parameters defined during the policy authoring must be mapped to attributes from the XACML request. In our healthcare example, the city name attribute could be populated in the Resource XACML attribute.

Policy distribution

With the policy configured, it can now be distributed by the IBM FileNet plug-ins to the IBM FileNet adapter. The adapter must already be installed, configured, and running on the IBM FileNet Content Manager.

During distribution, the XACML policy is distributed to the Policy Engine component of the IBM FileNet adapter using a secured HTTP Post mechanism as a batch mode event. The Policy Engine stores policies in its Policy Cache, where it then becomes available for use during authorization decisions.

All policies are distributed in a single XACML file. When a new policy is added or an existing policy is modified, the policy in Tivoli Security Policy Manager must be redistributed to the adapter. When a new version of the policy is transferred, the adapter discards old policies and refreshes the cache with the distributed policies.

Policy enforcement

Policy enforcement is facilitated by the IBM FileNet adapter using the following components:

- Policy Engine
- Event Handlers and Filters

Policy Engine

The primary responsibility of the Policy Engine is to parse the incoming XACML policy from Tivoli Security Policy Manager and store it in the cache. It also transforms the XACML policy into IBM FileNet ACLs for the target object (or group of objects) in the IBM FileNet Content Manager, based on an event. An event can be one of the following items:

- A property change event sent by IBM FileNet Content Manager, when the property of an object or content in the Content Manager has changed, which may require an ACL update.
- A timer event raised by a scheduler in the Policy Engine, resulting from a time-based, environmental context definition in the policy. For example, a repeat policy that occurs daily at a specified time, such as 9:00 a.m., when the organization opens for business.

Event Handlers and Filters

The adapter contains custom Event Handlers plug-ins written specifically for IBM FileNet. Event Handlers register for specific event types originating from the IBM FileNet Content Manager. These events are filtered at the source to ensure that authorization events from an application discovered in Tivoli Security Policy Manager are passed to the Event queue of the IBM FileNet adapter.

Property Change Events and Metadata Change Events originating from the IBM FileNet Content Manager are used to enforce policies dynamically. In our healthcare example, two events that may trigger an authorization decision include:

1. An ACL of a patient folder must change when the patient's state property changes from normal to admitted.
2. An ACL of a document must change when the document is classified as a daily treatment record.

When an event enters the queue, the adapter processes the authorization decision request using the event data and the cached policy in the Policy Engine to allow the FileNet Content Manager to either permit or deny the request.

Auditing and reporting

The Tivoli Security Policy Manager Adapter for IBM FileNet implements its own logging facility, which is used primarily for error tracking and resolution and recording general event history from the adapter. The adapter writes messages of varying severity levels from information notification to fatal errors to its own logs directory.

The logging severity can be configured and should be adjusted to the environment where the adapter is installed and configured as well as organizational requirements. For example, if the Event Handlers are regularly updating native ACLs as a result of content changes, and the threshold severity is too low, the log size may grow to an unmanageable size. If the severity threshold is configured too high, messages that may indicate an impending failure might not be recorded. Reporting and auditing on authorization decisions for content access is performed through the analysis capabilities of the ECM itself.

A potentially common critical error for the adapter is when the event queue is full., which indicates that the maximum size of the queue must be increased by modifying the EventQueue.size property. Monitoring the adapter logs for operational message such as the queue size enables the fine tuning of the adapter for each organization.

7.4.4 Conclusion

Tivoli Security Policy Manager Adapter for IBM FileNet provides several integration components for use in Tivoli Security Policy Manager and are used within the policy life cycle model. These integration components are used by Tivoli Security Policy Manager, allowing you to author content entitlement policies and enforce them on the IBM FileNet Content Manager by translating XACML policy into appropriate native security constructs.

The IBM FileNet adapter enables IBM FileNet to interoperate with XACML, adding the capability for rich security constructs, which can be adapted dynamically to meet the entitlements management complexities in industries such as healthcare.

By combining the powerful content management of IBM FileNet and the fine-grained policy authoring capabilities of Tivoli Security Policy Manager, the adapter offers a dynamic entitlements solution for challenging business environments.

8

Application level integration

In previous chapters, we discussed the business drivers and patterns for externalizing security policy enforcement and outlined intermediary, container, and database patterns. When you cannot successfully externalize the security policy enforcement with any of these patterns, the application integration pattern should be considered. This pattern involves developing applications that explicitly use an IT security policy management framework for security services.

Many business applications rely on custom-built authorization code, controlling what their users can do within the application. There are a number of disadvantages to this approach that Tivoli Security Policy Manager can help address:

1. Lack of visibility

   There is no way to inspect the policy driving the individual access decisions. The security logic is often embedded in the programming code itself, or in files that form part of the application.

2. Lack of control

   The personnel responsible for security across the organization need to liaise with each application team separately, rather than controlling policy directly. There is no single view of the security policy.

3. Lack of flexibility

   Updating the policy requires changes to application code, which is costly, inefficient, and can lead to an inconsistent security policy and gaps in the enterprise security solution.

Tivoli Security Policy Manager provides the capability for applications to externalize their security functions to a performant, XACML-based runtime engine driven by a standards-based policy framework. This policy can be centrally managed, is external to the application, and can be updated independent of application development cycles. This setup allows management of the security policy and associated security infrastructure to be removed from the application logic, allowing application developers to focus on delivering features that provide core business value.

In this chapter, we discuss the Tivoli Security Policy Manager interfaces for application integration and provide several examples of application integration.

8.1 Runtime security services interfaces

In this section, we discuss a summary of the interfaces provided by the Tivoli Security Policy Manager Runtime Security Services (RTSS) and guide you about where the interfaces are useful. We also outline a summary of the RTSS enforcement and extension points. The Tivoli Security Policy Manager RTSS Software Development Kit (SDK) provides detailed documentation on the interfaces and examples of how to use them.[1] The SDK is installed with the basic RTSS components.

8.1.1 Tivoli Security Policy Manager authorization API

The Tivoli Security Policy Manager authorization API allows a caller to make an authorization request to determine if access should be granted to a defined protected resource. The interface abstracts details of an authorization request and accepts context information, such as the currently authenticated user, the resource involved, and the action being performed. In response, the caller receives a Boolean authorization decision.

[1] For more information about the SDK, go to the following address:
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.tspm.doc_7.1/sdk/rtssSDKreadme.html

The Tivoli Security Policy Manager authorization API is defined in the `com.ibm.sec.authz.jaccplus` package. The following items are important concepts related to this API:

- Application identifier

  When performing an authorization decision, it is necessary to consider the resource in question. The Tivoli Security Policy Manager authorization API defines an *application identifier* (sometimes referred to as a *context identifier*) as a string value that uniquely identifies an application or service. It is a way to partition a policy space such that only a policy relevant to a particular application is considered during an authorization decision. The format of the string depends on the application or service being represented. In the case of a custom application, this identifier is used to represent the application within the Tivoli Security Policy Manager administration console. The value of this identifier is the "applicationID" used when defining the service structure.

- Evaluation context

  When a request to authorize access to a resource is made, various attributes of the user need to be considered in that decision. The information needs to be passed using a mechanism that is provided by the API. The *evaluation context* is the manner in which context information is passed to the decision engine. Context information includes the currently authenticated user, group membership, optional application-level roles, and other application-specific attributes. An evaluation context should be created once on initialization and then reused. An evaluation context should not be created on each request, as this action causes a significant performance impact. There are two forms of evaluation context:

  – ContainerEvaluationContext

    An evaluation context where subject data can be automatically retrieved from a supported container, such as WebSphere Application Server, for use in an authorization decision.

  – ApplicationEvaluationContext

    An evaluation context where subject information can be specified by the caller. This should be used when the application is not running in an environment supported by the ContainerEvaluationContext or the application is running in an environment that does support ContainerEvaluationContext, but where application security is disabled. This is often the case where another framework, such as Spring, manages authentication.

- ApplicationPermission

  The following attributes should be considered during policy enforcement:

  – The *service resource* to be considered in the authorization decision.

  – The *action* to be considered in the authorization decision.

  To represents a resource and action, the Tivoli Security Policy Manager API defines an ApplicationPermission class that takes string parameters to represent the service resource and action identifiers.

- ApplicationPolicy

  ApplicationPolicy is the primary object used to invoke the Tivoli Security Policy Manager API. It provides a method with a signature (String, EvaluationContext, Permission) that returns a Boolean decision. The string parameter represents the application identifier, which is the value specified when defining the service structure. This method evaluates the policy for the application specified by the context identifier to see whether the requested action is granted on the specified resource to any of the users and groups in the EvaluationContext.

Authorization using a container-authenticated subject

The Tivoli Security Policy Manager API can use the subject authenticated by the WebSphere container at run time for authorization decisions. The EvaluationContext implementation is `com.ibm.sec.authz.jaccplus.ContainerEvaluationContext`. An example is shown in Example 8-1.

Example 8-1 Authoring using a container-authenticated subject

```
//The context identifier normally the same within an application
public static String contextId = "applicationID";

//Create an EvaluationContext that gets data from the container.
//The context should be created once and re-used.
private EvaluationContext _evalContext = new ContainerEvaluationContext();

public boolean isAllowed( String resource, String action ) {
   Permission perm = new ApplicationPermission( resource, action );
   boolean retVal = ApplicationPolicy.getPolicy().implies( contextId, _evalContext, perm );
   return retVal;
}
```

Using an application-provided subject

If the application is not running in a container that supports the retrieval of subject data, it can specify that information itself by using the ApplicationEvaluationContext information defined by `com.ibm.sec.authz.jaccplus.ApplicationEvaluationContext`, as shown in Example 8-2.

The "handler data" map returned from `EvaluationContext#getHandlerData()` is a thread-local map used for passing context information to the authorization API. In this case, the constructed ApplicationSubject object in the map is using the appropriate key value.

Example 8-2 Example of using an application-provided subject

```
//The context identifier normally the same within an application
public static String contextId = "applicationID";

//Create an EvaluationContext that gets data from the application.
//The context should be created once and re-used.
private EvaluationContext _evalContext = new ApplicationEvaluationContext();

public boolean isAllowed( String user, String resource, String action ) {
   //First, convert our array of group identifiers to principals.
   List<Principal> groupPrincipals = new ArrayList<Principal>(groupIds.length );
   for ( String thisGroup : groupIds ) {
      groupPrincipals.add( new ApplicationGroupPrincipal( thisGroup ));
   }

   //Create our ApplicationSubject
   ApplicationSubject subject = new ApplicationSubject();

   //Specify the user identifier
   subject.setUserPrincipal( new ApplicationUserPrincipal( user ));
   subject.setGroupPrincipals(groupPrincipals.toArray( new Principal[] {} ));

   //Set it in the HandlerData map
   _evalContext.getHandlerData().put("java.security.auth.Subject.container", subject );

   Permission perm = new ApplicationPermission( resource, action ) ;

   boolean retVal = ApplicationPolicy.getPolicy().implies(contextId,  _evalContext, perm );

   return retVal;
}
```

Providing additional subject attributes

An application can provide additional attributes to be used when evaluating a policy. In the Tivoli Security Policy Manager console, these attributes are specified as being “in the request, in a standard location”. Attributes are added by implementing the `com.ibm.sec.authz.jaccplus.IAttributesHandler` interface, then registering this implementation with the `com.ibm.sec.authz.jaccplus.ApplicationAttributes` object for Subject, Resource, Action, or Environment.

The `com.ibm.sec.authz.jaccplus.ApplicationAttributes` object is pre-created for the Subject, as this is the most common use case. To register your handler against it, use the code in Example 8-3.

Example 8-3 Providing additional subject attributes

```
ApplicationEvaluationContext context = new ApplicationEvaluationContext();

((ApplicationAttributes)context.getContext("com.ibm.sec.auth.subjectx.SubjectAttributes.contain
er")).registerHandler(
    "[attributeIdentifier]", new IAttributesHandler() {

      public List<Object> getAttribute(String key, EvaluationContext evalCtx) throws
PolicyContextException
      {
          ArrayList<Object> retVal = new ArrayList<Object>();
          retVal.add( "[attribute-value]" );
          return retVal;
      }

      public String[] getSupportedAttributes() {
          return new String[] { "[attributeIdentifier]" };
      }

      public boolean supports(String key) {
          return ( "[attributeIdentifier]".equals( key ));
      }
    });
```

Providing additional resource, action, and environment attributes

Attributes can be provided to the Resource, Action, and Environment request sections in a manner similar to Example 8-3 on page 218. However, the `com.ibm.sec.authz.jaccplus.ApplicationAttributes` object must be created manually and registered against the appropriate key using the `com.ibm.sec.authz.jaccplus.EvaluationContext#registerHandler(String, com.ibm.sec.authz.jaccplus.IEvaluationContextHandler, boolean)` method, as shown in Example 8-4.

Table 8-1 shows the correct String keys to use. These keys are provided as constants of the ApplicationAttributes object as well.

Table 8-1 String keys for Resource, Action, and Environment

Request section	Context key
Resource	“`com.ibm.sec.auth.subjectx.ResourceAttributes.container`”
Action	“`com.ibm.sec.auth.subjectx.ActionAttributes.container`”
Environment	“`com.ibm.sec.auth.subjectx.EnvironmentAttributes.container`”

Example 8-4 Adding additional subject, resources, and environment attributes

```
ApplicationEvaluationContext context = new ApplicationEvaluationContext();

//The code for the AttributeContextHandler class is shown below.
IEvaluationContextHandler resourceContextHandler = new AttributeContextHandler(
    "com.ibm.sec.auth.subjectx.ResourceAttributes.container" );

//We could also use ApplicationAttributes.ATTR_RESOURCE_KEY as the first parameter here
// Passing "true" means to over-ride any existing handler for this key; if false and a handler
// already exists an exception is thrown.
context.registerHandler( "com.ibm.sec.auth.subjectx.ResourceAttributes.container",
resourceContextHandler, true);

//Now register IAttributeHandler implementations against the resourceAttrs object as above.
((ApplicationAttributes)context.getContext("com.ibm.sec.auth.subjectx.ResourceAttributes.contai
ner"))
    .registerHandler( "[AttributeId]", handlerImpl );

//This is the code that returns an ApplicationAttributes object in response to the getContext()
call.
private static class AttributeContextHandler implements IEvaluationContextHandler
{

    private final String _context;
    private final ApplicationAttributes _attrs = new ApplicationAttributes();
    public AttributeContextHandler( String context ) {
        this._context = context;
    }

    public Object getContext(String key, Map<String, Object> handlerData) {

        if ( _context.equals( key ))
        {
            return _attrs;
        }
        else
        {
            return null;
        }
    }

    public String[] getKeys() {
        return new String[] { _context };
    }

    public boolean supports(String key) {
        return _context.equals( key );
    }
}
```

Providing an XML fragment with the resource

An XML element can be provided with the resource at run time, allowing the policy to use XPath to extract data as part of the authorization decision, as shown in Example 8-5. The ApplicationPermission object has the capability to hold an XML Node object, which is then automatically serialized at run time.

Example 8-5 Providing an XML fragment

```
//The context identifier normally the same within an application
public static String contextId = "applicationID";

//Create an EvaluationContext that gets data from the container.  The context
// should be created once and re-used.
private EvaluationContext _evalContext = new ContainerEvaluationContext();

public boolean isAllowed( String resource, String action, Node xmlContent ) {
   Permission perm = new ApplicationPermission( resource, action ) ;
   perm.setContent( xmlContent );

   boolean retVal = ApplicationPolicy.getPolicy().implies( contextId, _evalContext, perm );
   return retVal;
}
```

Providing a WS-Security authentication token with the subject

A WS-Security token can be added to the ApplicationSubject, as shown in Example 8-6, which is serialized as part of the XACML Subject. Tivoli Security Policy Manager can then send this token to Tivoli Federated Identity Manager for validation or exchange. This token can be used in conjunction with, or instead of, setting user and group identifiers, depending on the use case.

Example 8-6 Providing a WS-Security authentication token

```
//The context identifier normally the same within an application
public static String contextId = "applicationID";

//Create an EvaluationContext that gets data from the application.
//The context should be created once and re-used.
private EvaluationContext _evalContext = new ApplicationEvaluationContext();

public boolean isAllowed( Element authnToken, String resource, String action ) {
   //Create our ApplicationSubject
   ApplicationSubject subject = new ApplicationSubject();

   //Add a WS-Security token
   subject.setAuthenticationToken( tokenToken );

   //Set it in the HandlerData map
   _evalContext.getHandlerData().put("java.security.auth.Subject.container", subject );

   Permission perm = new ApplicationPermission( resource, action ) ;
   boolean retVal = ApplicationPolicy.getPolicy().implies( contextId, _evalContext, perm );
   return retVal;
}
```

Using DelegatedPermissionCollection for batch authorization requests

Rather than making a series of implies() calls for a known set of ApplicationPermission objects, the `com.ibm.sec.authz.jaccplus.DelegatedPermissionCollection` object can be used to make a single implies() call, as shown in Example 8-7. Depending on the deployment scenario, this action may result in higher performance due to fewer remote calls being made, for example, to check for access to three known resources.

Example 8-7 Batch authorization requests

```
//The context identifier normally the same within an application
public static String contextId = "applicationID";

//Create an EvaluationContext that gets data from the container.  The context
// should be created once and re-used.
private EvaluationContext _evalContext = new ContainerEvaluationContext();

DelegatedPermissionCollection dpc = new DelegatedPermissionCollection();
dpc.add( new ApplicationPermission( "resource-1", "read" ));
dpc.add( new ApplicationPermission( "resource-2", "read" ));
dpc.add( new ApplicationPermission( "resource-3", "read" ));

//If all the Permissions in the collection are allowed, then the return value is
// true.
boolean allPermit = ApplicationPolicy.getPolicy().implies( contextId, _evalContext, dpc );

//Individual access results can be checked by calling DelegatedPermissionCollection#implies().
boolean resource1Permit = dpc.implies( new ApplicationPermission( "resource-1", "read" ) ));
boolean resource2Permit = dpc.implies( new ApplicationPermission( "resource-2", "read" ) ));
boolean resource3Permit = dpc.implies( new ApplicationPermission( "resource-3", "read" ) ));
```

Using getPermissions() for application entitlements

The `com.ibm.sec.authz.jaccplus.ApplicationPolicy#getPermissions(String, com.ibm.sec.authz.jaccplus.EvaluationContext, Class)` method can be used to retrieve a collection of Permissions that the currently authenticated Subject can access, as shown in Example 8-8.

Example 8-8 Retrieving application entitlements using getPermissions()

```
//The context identifier normally the same within an application
public static String contextId = "applicationID";

//Create an EvaluationContext that gets data from the container.  The context
// should be created once and re-used.
private EvaluationContext _evalContext = new ContainerEvaluationContext();
```

```
public List<String> getEntitlement( String action ) {
   //Get the list of resources that the current user can perform the specified action on
   List<String> retVal = new ArrayList<String>();

   //Call the getPermissions() API. Please note that only ApplicationPermission is supported.
   PermissionCollection entitledPerms = ApplicationPolicy.getPolicy().getPermissions(
contextId, _evalContext, ApplicationPermission.class );

   //You can loop over the returned Permissions using an Enumeration
   Enumeration<Permission> e = entitledPerms.elements();
   while ( e.hasMoreElements() )
   {
        ApplicationPermission thisPerm = (ApplicationPermission)e.nextElement();
      if ( action.equals( thisPerm.getActions() ) ) {
         retVal.add( thisPerm.getName() );
      }
   }

   return retVal;
}
```

A Permission class must be passed to this API to determine the type of Permissions extracted from the policy. The only supported class at this time is `com.ibm.sec.authz.jaccplus.ApplicationPermission`.

8.1.2 JSP tag library

The Tivoli Security Policy Manager tag library is provided by the RTSS SDK. The tag library is used to annotate authorization requests within a JSP page. The library provides an XML-like programming interface that allows tags to define tag attributes to construct an authorization request using the RTSS client.

The tag library supports two types of enforcement:

- Authorization

  Requests a decision using a specific subject, action, and resource combination. The return of this call is a Boolean response.

- Entitlement

  Requests all of the authorized resources of a given subject and action. The returned entitlement is stored in a cache, and a subsequent authorization request for the same action and subject uses the cached data, thus avoiding a call to the RTSS client.

The JSP library is included in a JSP page by including the following line:

```
<%@ taglib uri=”http://ibm.com/rtss/taglibs/1.0/authz” prefix=”tspm” %>
```

In the rest of this section, we discuss some of the tags available in the Tivoli Security Policy Manager tag library.

Context tag

The <context> tag contains the serviceId used by the RTSS client to determine the policies of a service defined in Tivoli Security Policy Manager. The context tag may also contain a set of user-defined attributes, and whether to retrieve entitlements.

Multiple context tags can be defined on the page to obtain entitlements from different serviceId namespaces, or to set different user defined attributes for the same serviceId namespace (Table 8-2).

Table 8-2 Supported attributes for the context tag

Attribute	Description
ServiceId	Identifier of the service that must match the namespace of a service defined in Tivoli Security Policy Manager.
id	A string used to identify a context that can be referenced by the authorize / deny tags.
entitlement	A Boolean value used to designate whether to retrieve the entitlement of a service specified by the serviceId.

Example 8-9 shows a sample of how to define a context tag. This context definition can be used by the other tag examples in this section.

Example 8-9 Example context tag

```
<tspm:context id="ctx1" serviceId="http://mydomain.com/myapp" />
```

Authorize tag

The <authorize> tag is used to dynamically hide or show parts of a JSP or portlet based on policy authored in Tivoli Security Policy Manager. If the current user is authorized, then the content of the tag is shown. If the user is not authorized, then the content is not rendered. The authorization decision is based on the subject, the specified action, and the specified resource ID. If the authorization is denied, the elements under the authorize tag are not evaluated and displayed.

The authorize tag inherits its serviceId, user-defined attributes, and whether to retrieve entitlements or not from the referenced context. If the referenced context contains an entitlement, then the authorize tag uses the cached entitlement to make the authorization decision. If the referenced service identifier contains no entitlement, then the authorize tag calls the RTSS client for an authorization decision. Table 8-3 shows the supported attributes for the <authorize> tag.

Table 8-3 Supported attributes for the authorize tag

Attribute	Description
action	An action name defined in Tivoli Security Policy Manager for the service.
resourceId	The resource ID of an element in a service structure defined in Tivoli Security Policy Manager.
contextId	The identifier of the context to use.

Example 8-10 shows how an <authorize> tag is used to determine if a subject has access to the data within the authorize tag. If the subject is authorized, the data is displayed. If the subject is not authorized, the data is not displayed.

Example 8-10 Example authorize tag

```
<tspm:authorize action="viewPolicy" resourceId="myapp:rbac:policies"
contextId="ctx1">
   <table name="policies">
   <tr><th>Name</th><th>Description</th></tr>
   <tr><td>policyA</td><td>Allow access to all</td></tr> </table>
</tspm:authorize>
```

Deny tag

The <deny> tag is the opposite of the <authorize> tag. It is used to display content when a request is not allowed. If the request is denied, the elements under the deny tag are evaluated and displayed. Table 8-4 shows the supported attributes for the <deny> tag.

Table 8-4 Supported attributes for the deny tag

Attribute	Description
action	An action name defined in Tivoli Security Policy Manager for the service.
resourceId	The resource identifier of an element in a service structure defined in Tivoli Security Policy Manager.
contextId	The identifier of the context to use.

Example 8-11 shows a sample of how a <deny> tag can be used to display content in an unauthorized use case. This tag can be used in conjunction with the <authorize> tag to display content informing a user whether they are authorized to access a resource or not.

Example 8-11 Authorization using the deny tag

```
<tspm:deny action="viewPolicy" resourceId="myapp:rbac:policies"
contextId="ctx1">
   You are not authorized to view the policy myapp:rbac:policies!
</tspm:deny>
```

User provided context attributes

By default, the tag library uses the Tivoli Security Policy Manager API ContainerEvaluationContext for policy evaluation. In addition to the container-provided context, the application developer may inject additional subject, resource, action, and environment attributes to the evaluation context (Table 8-5).

Table 8-5 Supported attributes

Attribute	Description
subject	A set of subject attributes that can be used in an authorization decision.
resources	A set of resources attributes that can be used in an authorization decision.
environments	A set of environment attributes that can be used in an authorization decision.
actions	A set of action attributes that can be used in an authorization decision.

Example 8-12 shows a sample of how additional context attributes can be defined.

Example 8-12 Definition of additional context attributes

```
tspm:context id="ctx1" serviceId="http://mydomain.com/myapp" >
   <tspm:subjects id="attrOId">
      <tspm:attribute value="a"/>
      <tspm:attribute value="b"/>
   </tspm:subjects>
   <tspm:resources id="attr2Id">
      <tspm:attribute value="a"/>
   </tspm:resources>
   <tspm:actions id="attr3Id">
      <tspm:attribute value="a"/>
   </tspm:actions>
   <tspm:environments id="attr4Id">
      <tspm:attribute value="a"/>
   </tspm:environments>
</tspm:context>
```

8.1.3 Custom authorization solutions for external systems

There are many situations where flexible authorization capabilities are required to integrate systems into a policy management infrastructure, such as:

- Custom devices requirements

  Applications are being deployed on distributed smart devices, such as power meters, industrial sensors, and so on. As these capabilities become more wide spread and are embedded in mission critical devices, security is becoming an important consideration. There are unique challenges that do not exist in other policy enforcement scenarios, such as requirements to run with a minimal memory footprint, limited communications protocols, limited availability of cryptographic technology, and so on. A custom solution may be required to work around technical restrictions imposed by technology in these cases.

- Legacy application

  Some applications may not easily integrate with the technology supported by Tivoli Security Policy Manager. This may be due to applications using technology that is not based on open standards, legacy or proprietary technology, and so on.

In such cases, there are several options for integrating with Tivoli Security Policy Manager. These can be divided into two categories:

- Custom authorization client implementation of the Tivoli Security Policy Manager authorization web service
- Custom authorization application using the Tivoli Security Policy Manager API

Authorization web service interface

Tivoli Security Policy Manager provides an authorization web service, which authorization clients can use. The authorization web service is available as a way to query the runtime security services server for authorization decisions. This model can be used in any environment where a web service client can be deployed.

The authorization web service allows you to send XACML requests to your runtime security services server using the SOAP protocol. While IBM does not provide a pre-built web services client, the Web Service Description Language (WSDL) and associated files are provided as an attachment so you can generate your own client.

The Tivoli Security Policy Manager authorization web service is defined using a WSDL file. An authorization client can implement the Tivoli Security Policy Manager authorization WSDL to call the Tivoli Security Policy Manager authorization web service.

The authorization web service provides the following operations:

- evaluateXACML

  This operation receives an `evaluateXACMLRequest` message that is the standard XACML authorization request and produces a standardized authorization decision in the form of a `valuateXACMLResponse`.

- evaluateEntitlements

  This operation receives a proprietary `evaluateEntitlementRequest` message containing the required information to produce a list of entitlements for the designated subject.

Authorization requests are sent to the runtime security services server using the following URL, customized to your environment, as shown in Example 8-13.

Example 8-13 Web service client URL

```
http://<rtss host>:<web container port>/rtss/authz/services/AuthzService
```

The protocol for a simple permit or deny authorization decision is an XACML request over SOAP. The attributes contained within this request should follow a few guidelines. Four elements are required for authorization: a set of *subjects*, a set of *resources*, an *action* performed against the resources, and the *environment*, or application context, which scopes the context for evaluation. Attributes are included in each section in the XML format, as shown in Example 8-14.

Example 8-14 XACML authorization request

```
<xacml-context:Attribute AttributeId="\<AttributeId\>" DataType="\<DataType\>" >
    <xacml-context:AttributeValue>value</xacml-context:AttributeValue>
</xacml-context:Attribute
```

The tables below reference the AttributeId and DataType fields.

Subject

An XACML subject consists of a set of attributes that describe the principals involved in the authorization request. For example, one subject can be the user requesting access to the resource while another subject represents the policy enforcement point (PEP) or code sending the request to the policy decision point (PDP). The subject-category defines the type of subject. In an XACML request, at least one *access-subject* must be defined.

The user's subject-id and a set of group-ids are usually specified here as distinguished names.

Table 8-6 Subject AttributeId and DataType

AttributeId	DataType	Value
urn:oasis:names:tc:xacml:1.0:subject:subject-id	http://www.w3.org/2001/XMLSchema#string	The user's login name as a string
urn:oasis:names:tc:xacml:1.0:subject:subject-id	urn:oasis:names:tc:xacml:1.0:data-type:x500Name	The user's login name as an X500Name
urn:oasis:names:tc:xacml:1.0:subject:group-id	http://www.w3.org/2001/XMLSchema#string	The groups the user is a member of as a string
urn:oasis:names:tc:xacml:1.0:subject:group-id	urn:oasis:names:tc:xacml:1.0:data-type:x500Name	The groups the user is a member of as an X500Name

To add custom attributes for the user, add more elements in the Subject block. Use a custom AttributeId that describes the attribute (it must be a URI) and an appropriate data type.

Resource

The XACML resource elements describe the resource to which the subject is requesting access. Examples of resources are web services operations, a URL, or custom service IDs (if using custom applications). The contents of this element depends on the type of service the policy is authored for, such as *Application* (Table 8-7) or *Web Service* (Table 8-8 on page 230).

Table 8-7 Application AttributeId and DataType

AttributeId	DataType	Value
urn:oasis:names:tc:xacml:1.0:resource:resource-id	http://www.w3.org/2001/XMLSchema#string	The element name of the resource being accessed

Table 8-8 Web Service AttributeId and DataType

AttributeId	DataType	Value
urn:oasis:names:tc:xacml:1.0:resource:resource-id	http://www.w3.org/2001/XMLSchema#anyURI	The HTTP URL of the web service being accessed
urn:ibm:xacml:profiles:web-services:1.0:wsdl:1.1:service	http://www.w3.org/2001/XMLSchema#string	The service name in the WSDL of the service being accessed, in the format {namespace}Service
urn:ibm:xacml:profiles:web-services:1.0:wsdl:1.1:port	http://www.w3.org/2001/XMLSchema#string	The port name in the WSDL of the service being accessed, in the format {namespace}Port
urn:ibm:xacml:profiles:web-services:1.0:wsdl:1.1:operation	http://www.w3.org/2001/XMLSchema#string	The operation name in the WSDL of the service being accessed, in the format {namespace}Operation

Action

The action element describes the action the subject is trying to perform on the resource. The XACML standard dictates that an action element needs to provide one urn:oasis:names:tc:xacml:1.0:action:action-id attribute.

Table 8-9 Action AttributeId and DataType

AttributeId	DataType	Value
urn:oasis:names:tc:xacml:1.0:action:action-id	http://www.w3.org/2001/XMLSchema#string	The action being performed on the resource

Environment

The environment element is required to have an attribute contextId that specifies the application ID, as shown in Example 8-15.

Example 8-15 Environment example

```
<xacml-context:Environment>
    <xacml-context:Attribute AttributeId="ContextId"
DataType="http://www.w3.org/2001/XMLSchema#string"
    Issuer="http://security.tivoli.ibm.com/policy/distribution">
        <xacml-context:AttributeValue>test-app</xacml-context:AttributeValue>
    </xacml-context:Attribute>
</xacml-context:Environment>
```

The following rules apply when using the environment element:

- If the full DN is being provided for the subject-id and group-id attributes, then they *must* be included as both DataTypes http://www.w3.org/2001/XMLSchema#string and urn:oasis:names:tc:xacml:1.0:data-type:x500Name.
- For the Application service type in Tivoli Security Policy Manager, the resource-id and action-id should be of type http://www.w3.org/2001/XMLSchema#string.

Tivoli Security Policy Manager API with REST-based interface

A useful pattern for implementing authorization with external systems is to use the capabilities of the Tivoli Security Policy Manager API to build a custom authorization interface. The Web 2.0 paradigm provides a range of options that should be considered when implementing such an interface.

Representational State Transfer (REST) defines an architectural style for transferring representations of resources between a client and server, and managing resource states. It is a commonly used method in Web 2.0 technology to implement a RESTful facade against web services or applications.

In a RESTful approach, HTTP methods can be used to interact between a client and server. It has become popular to use a RESTful methodology to transfer representations of resources over HTTP using technologies, such as JavaScript Object Notation (JSON).

Figure 8-1 shows how a custom authorization interface can be exposed using a RESTful facade and JSON.

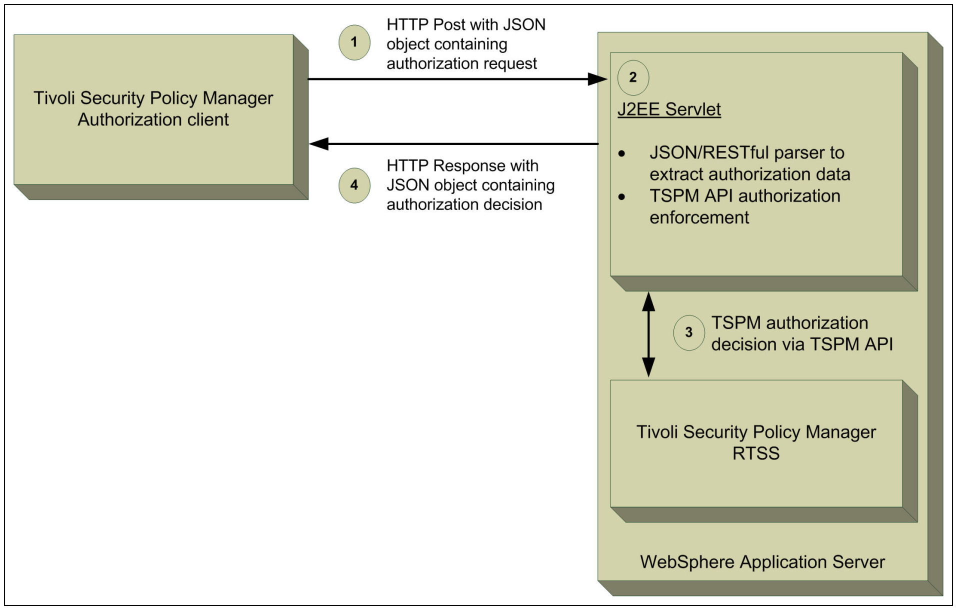

Tivoli Security Policy Manager Authorization client
1
HTTP Post with JSON object containing authorization request
4
HTTP Response with JSON object containing authorization decision
2
J2EE Servlet
• JSON/RESTful parser to extract authorization data
• TSPM API authorization enforcement
3
TSPM authorization decision via TSPM API
Tivoli Security Policy Manager RTSS
WebSphere Application Server

Figure 8-1 Custom RESTful based authorization solution

The server side solution consists of the following steps:

1. A RESTful authorization client submits an HTTP post request to a WebSphere Application Server running a custom authorization solution. The request contains a JSON object that includes details about the authorization request, such as subject, target resource, and context.
2. A custom authorization solution that enforces Tivoli Security Policy Manager policy is hosted in a WebSphere J2EE servlet. The authorization request is processed by a J2EE servlet doPost method. A JSON package can be used to capture the authorization data in the post request.
3. The subject, target, and context information extracted from the RESTful request can be used in a Tivoli Security Policy Manager API authorization decision, as defined in 8.1, "Runtime security services interfaces" on page 214.
4. The authorization result captured by the Tivoli Security Policy Manager API can be encapsulated in a JSON object and returned to the client in an HTTP response.

8.1.4 Policy information point

Business rules often have complex logic and pre-conditions that must be met to permit a particular operation as opposed to a simple allow or deny decision based on group or role membership in a typical user repository. In the context of authorization and entitlement management, it may be necessary to query external data sources to retrieve additional information to assist in making the decision, or to retrieve specific values against which to compare pre-conditions.

To facilitate authorization against business context information from external systems, Tivoli Security Policy Manager provides the *policy information point* (PIP) interface. The PIP interface provides a mechanism to query external information sources for attribute data to be used in an authorization decision. The PIP's function is to look up, compute, or make arbitrary values available for use during policy evaluation, based on request data from Tivoli Security Policy Manager. There are several types of PIPs that allow Tivoli Security Policy Manager to interface with external systems. In this section, we focus on the Java PIP.

The Java PIP allows custom written Java code to act as a Tivoli Security Policy Manager PIP, effectively enabling the system to interface with any technology that supports the Java language.

PIPs are implemented as an OSGi[2] extension plug-in to the RTSS server or client. The SDK provides an example PIP, which should be used as a starting point for any custom Java PIP development. A Java PIP consists of:

- A plug-in implementation class

  The Java PIP should implement the `com.ibm.tscc.rtss.authz.api.IExternalFinder` interface. The interface defines several methods that require implementation. Once defined, the implementation class is referenced in the plug-in configuration file.

- Plug-in configuration

  An OSGi plug-in file should be created to extend the `com.ibm.tscc.rtss.authz.custom` plug-in point. This plug-in defines the PIP. Both the class file and the `plugin.xml` file form part of the jar file that make up the deployable PIP. Example 8-16 shows a sample OSGi extension point plug-in file.

  Example 8-16 Example PIP plugin.xml

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <?eclipse version="3.2"?>
  ```

[2] To learn more about the Open Services Gateway initiative, go to the following address: http://www.osgi.org

```
<plugin>
   <extension id="com.ibm.demo.authz.finder.sample"
            name="Sample Attribute Finder"
            point="com.ibm.tscc.rtss.authz.custom">
      <custom class="com.ibm.demo.authz.finder.sample">
         <ConfigParameter
              Name="usernameAttributeId"
              Required="true"/>
      </custom>
   </extension>
</plugin>
```

- An entry in the Tivoli Security Policy Manager configuration

  The `security-services.xmi` file has to be updated with an entry that configures the PIP into Tivoli Security Policy Manager. This file is located in the following directory for the RTSS server:

  `<PROFILE_HOME>/config/cells/<CELL_NAME>/rtss/`

  The `security-services.xmi` file is located in the following directory for the RTSS client:

  `<PROFILE_HOME>/config/cells/<CELL_NAME>/ertss`

  In Example 8-17, we show the configuration entry for a sample Java PIP. Note that osgiCfgInit.sh needs to be run if the PIP is used on the RTSS client.

Example 8-17 Java PIP configuration entry

```
<components name="Authz">
   <subComponents name="AttributeRetrievalServices">
      <items name="Sample external rule">
         <properties>
            <values name="type" value="custom" type="java.lang.String"/>
            <values name="id" value="com.ibm.demo.authz.finder.sample"
                     type="java.lang.String"/>
            <values name="enabled" value="true" type="java.lang.String"/>
            <values name="usernameAttributeId" value="userIdentifier"
                     type="java.lang.String"/>
         </properties>
      </items>
   </subComponents>
</component>
```

Policy information point deployment

To deploy a PIP into an RTSS, the following steps are required. Note that the RTSS client paths are used in this example.

1. Stop the WebSphere Application Server where the RTSS client is installed.
2. Copy the PIP plug-in .jar file to the following location:

   `<WEBSPHERE_SERVER_HOME>/plugins`
3. Make a backup copy of the following file:

   ```
   <WEBSPHERE_PROFILE>/config/cells/<WEBSPHERE_CELL_NAME>/ertss/securit
   y-services.xmi
   ```
4. Add the PIP configuration to the following file:

   ```
   <WEBSPHERE_PROFILE>/config/cells/<WEBSPHERE_CELL_NAME>/ertss/securit
   y-services.xmi
   ```
5. Start WebSphere Application Server.

8.1.5 External rules

Tivoli Security Policy Manager provides a flexible authorization framework by using *external rules*. An external rule provides a pluggable callout to an external authorization service that returns a permit or deny response. External rules are implemented as an OSGi plug-in extending a Tivoli Security Policy Manager RTSS extension point.

An external rule consists of the following parts:

- Implementing class

  To implement an external rule, a Java class must be written that implements the `com.ibm.tscc.rtss.authz.api.IExternalRule` interface. This interface consists of startup, shutdown, and evaluate methods. The evaluate method is passed as a `com.ibm.tscc.rtss.authz.api.RequestContext` object that contains the context of the request.
- Plug-in configuration

  The plug-in configuration file defines the OSGi extension point, id, name, and configuration parameters required for an external rule. In Example 8-18, we show a sample `plugin.xml` configuration file for a risk classification rule.

  Example 8-18 Sample plugin.xml file for external rule

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <?eclipse version="3.2"?>
  <plugin>
     <extension
  ```

```
            id="RiskClassificationRuleSample"
            name="Sample External Rule Implementation"
            point="com.ibm.tscc.rtss.authz.erule">
         <erule class=.RiskClassificationRule">
            <ConfigParameter
                  Name="Hostname"
                  Required="true"/>
            <ConfigParameter
                  Name="Username"
                  Required="false"/>
            <PolicyParameter
                  Description="Risk classification"
                  Name="RiskLevel"
                  Required="false"/>
         </erule>
      </extension>

   </plugin>
```

- Tivoli Security Policy Manager configuration

  Tivoli Security Policy Manager is configured to use an external rule using the `security-services.xmi` file. This file is located in the following directory for the RTSS server:

  `<WASPROFILE>/config/cells/<CELL_NAME>/rtss`

  The `security-services.xmi` file is located in the following directory for the RTSS client:

  `<WASPROFILE>/config/cells/<CELL_NAME>/ertss  (RTSS Client)`

  This file contains the properties used for the external rule as defined in the external rule `plugin.xml`. These parameters are used at startup and are passed in a `java.utils.Properties` object to the startup method of a class that implements the IExternalRule interface. These properties are then accessible to the Java code implementing the external rule. In Example 8-19, we show an excerpt from a `security-services.xmi` file that defines the configuration for an example risk classification rule.

Example 8-19   Example security-services.xmi file

```
subComponents name="ExternalRules">
         <items name="server1-rules-engine">
           <properties>
             <values name="id" value="RiskClassificationRuleSample"
                               type="java.lang.String" />
             <values name="enabled" value="true" type="java.lang.String" />
             <values name="Hostname" value="server1.customer.com"
type="java.lang.String" />
             <values name="Username" value="foo" type="java.lang.String" />
```

```
      </properties>
    </items>
```

8.2 Policy management API

Tivoli Security Policy Manager provides a range of basic features to manage supported types of services, policies, service registries, policy distribution targets, and so on. To support complex environments, the product has been designed to use an extensible and pluggable architecture that allows for various aspects of the product to be customized. The ability to extend the capabilities of Tivoli Security Policy Manager allows custom solutions to be developed.

The Tivoli Security Policy Manager Software Development Kit[3] (SDK) provides details about the interfaces, classes, and examples of each of the major plug-in types. When considering a project to customize Tivoli Security Policy Manager plug-ins, these examples can be used as a starting point to guide the implementation. We discuss the following aspects of the APIs:

- Plug-in structure
- Data model
- Plug-ins

8.2.1 Plug-in structure

Tivoli Security Policy Manager uses a pluggable OSGi based framework that allows plug-ins for various types of objects to be defined.

Custom Tivoli Security Policy Manager OSGi plug-ins extend defined extension points to create new types of objects in the system. The product provides several basic plug-ins for each extension point. Examples include:

- Service type plug-ins, such as the WebService service plug-in, the J2EE service plug-in, and the database service plug-in
- Policy type plug-ins, such as the authorization policy type plug-in, and the message protection policy type plug-in
- Service registry types, such as the WebSphere Service Registry and Repository registry plug-in and the Sharepoint Server registry plug-in

[3] For more information about the SDK, go to the following address:
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.tspm.doc_7.1/sdk/tspmSDKreadme.html

Additional customized plug-ins can be defined for each extension point to extend the capabilities of the product to support the required services, policies, service registries, and so on. A custom plug-in may be required if:

- Integration with a non-supported service registry is required.
- Policy distribution is required for a non-supported policy distribution target.
- A custom service type is required.
- A custom policy type is required.

The process of developing plug-ins involves implementing the following components:

- Plug-in configuration
- Plug-in relationships
- Java implementation

Plug-in configuration

The `plugin.xml` file of the `com.ibm.tspm.extpts package` defines the available Tivoli Security Policy Manager extension points and the schema file for each plug-in type. The schema file associated with each type of plug-in defines the Tivoli Security Policy Manager interfaces required by each type of plug-in and the structure of the plug-in configuration file.

Each plug-in consists of its own `plugin.xml` configuration file that defines the plug-in. The `plugin.xml` file must conform to the schema definition for the appropriate plug-in type defined in the `com.ibm.tspm.extpts` package. The recommended approach for developing a plug-in is to follow an example from the Tivoli Security Policy Manager SDK. Example 8-20 shows the DB2 service registry plug-in.

Example 8-20 DB2 service registry plug-in

```
<plugin>
   <extension
         id="com.ibm.tspm.registry.db2"
         name="DB2 Service Definition Registry"
         point="com.ibm.tspm.extpts.serviceDefinitionRegistryType">
      <serviceDefinitionRegistry
            UIProviderExtensionId="com.ibm.tspm.registry.db2.ui"
            UIProviderPluginId="com.ibm.tspm.registry.db2.ui"
            factoryImpl="com.ibm.tspm.registry.db2.DB2ServiceRegistryFactory"
            serviceDefinitionRegistryImpl="com.ibm.tspm.registry.db2.DB2ServiceRegistry"
            typeImpl="com.ibm.tspm.registry.db2.DB2ServiceRegistryType">
      </serviceDefinitionRegistry>
   </extension>
</plugin>
```

Commonly used attributes within this plug-in are shown in Table 8-10.

Table 8-10 Common attributes for the DB2 service registry plug-in

Attribute	**Description**
id	Identifies the plug-in.
point	The extension point that this plug-in extends. The `com.ibm.tspm.extpts plugin.xml` file defines the serviceDefinitionRegistryType extension point and associates it with the `serviceDefinitionRegistryType.exsd` schema. This file defines the format of the plug-in configuration file and the interfaces required.
UIProviderExtensionId	A plug-in that provides the UI for creating or modifying an object of this type.
factoryImpl	A class that implements the factory interface for the db2 service definition registry plug-in. In this case, the implementing class implements the IServiceDefinitionRegistryFactory interface defined in the `serviceDefinitionRegistryType.exsd` schema file from the `com.ibm.tspm.extpts` package.
serviceDefinitionRegistryImpl	A class that implements the service definition registry interface for the DB2 service definition registry plug-in. In this case, the implementing class implements the IServiceDefinitionRegistry interface defined in the `serviceDefinitionRegistryType.exsd` schema file from `com.ibm.tspm.extpts`.
typeImpl	A class that implements the service definition registry type interface for the DB2 service definition registry plug-in. In this case, the implementing class implements the IServiceDefinitionRegistryType interface defined in the `serviceDefinitionRegistryType.exsd` schema file from `com.ibm.tspm.extpts`.

Plug-in relationships

When defining custom Tivoli Security Policy Manager plug-ins, constraints need to be applied that define how objects relate to each other. Some objects in Tivoli Security Policy Manager are not compatible with certain other objects. These relationships are captured in the following file:

```
<WAS_PROFILE>/config/tspm/etc/com.ibm.tspm.servicePluginsConf.xmi
```

The following categories of object relationships can be defined within this file:

- ServiceTypeID-WorksWith-PolicyTypeIDs

  Not all service types can be attached to each policy type. For example, message protection policies will not work with all service types, such as J2EE applications. This relationship is not valid and this is defined using this mapping.

- ServiceTypeID-WorksWith-ServiceDefRegIDs

  Not all service types work with all service definition registries. For example, the WebSphere service type works with the WebSphere Service Registry and Repository service definition registry and the Universal Description Discovery and Integration servers (UDDI) service definition registry, but the J2EE application service type only works with the RTSS service definition registry.

- PDPTypeID-CanEval-PolicyTypeIDs

  Not all policy decision points can evaluate all policy types. For example, the DataPower policy decision point can evaluate both message protection policy and XACML policy, but the DB2 policy decision point can only evaluate XACML policy.

- PDTTypeID-CanStore-PolicyTypeIDs

  Not all policy distribution target types can store all policy types. For example, the RTSS can only process XACML policy, and the DataPower policy distribution point can store both message protection policy as well as XACML policy.

- PDPTypeID-OverridesConfig-PolicyTypeIDs

  Some policy decision point types can override the configuration of a policy type. For example, the DB2 policy decision point can override the configuration of XACML authorization policies.

Java implementation

Interfaces define the expected behavior of individual plug-ins. The Tivoli Security Policy Manager interface for each plug-in type is discussed later in this section. Each type of plug-in has its own set of interfaces that are defined by the `com.ibm.tspm.extpts` package. The appropriate interfaces for each plug-in must be implemented to build a plug-in that functions correctly.

8.2.2 Data model

The `com.ibm.tspm.datamodel.vo` package contains data model interfaces that are used throughout the product and are exposed for use in Tivoli Security Policy Manager plug-ins. Classes that implement the value object interfaces represent objects that are persisted in the Tivoli Security Policy Manager repository.

- IClassificationVO

  The IClassificationVO interface defines a Tivoli Security Policy Manager classification. A classification is a grouping mechanism for services and policies that allow for more effective management of these objects. Classifications are used to create an association between zero or more policies and zero or more services. A classification provides a convenient way to allow a collection of policies to be associated with a collection of services.

- IEffectivePolicyVO

  An effective policy represents the combined policy for a service. When multiple policies are defined for a service, they are consolidated into an effective policy that can then be used by a policy decision point to enforce authorization decisions. Effective policy can contain one or more effective policy data objects defined by the IEffectivePolicyDataVO interface. IEffectivePolicyVO value objects are created through `com.ibm.tspm.datamodel.vo.impl.PolicyVOFactory`.

- IEffectivePolicyDataVO

  The IEffectivePolicyDataVO interface defines effective policy data that applies to a given service scope. Effective policy, presented by the IEffectivePolicyVO interface, can contain one or more effective policy data objects where the collection of generated effective policy data is the effective policy for a service. The effective policy data includes the generated policy and the MIME type of the generated policy for exporting to a file. New IEffectivePolicyDataVO value objects are created through `com.ibm.tspm.datamodel.vo.impl.PolicyVOFactory`.

- IPolicyDistributionTargetVO

  A policy distribution target (PDT) is the endpoint for a policy. Examples of PDTs include DataPower and WebSphere Application Server. The policy distribution target might enforce the policy, use the policy to make an access decision for enforcement elsewhere, or simply store the policy.

- IDistributionMappingVO

  A distribution mapping is a representation of an association between a policy type, a PDT, a PDP type, and a service. This association is compose of information that is necessary to attach, configure, and distribute policy to an endpoint. This object also includes status information captured during policy configuration and distribution. A distribution mapping must exist in order for policy to be distributed.

- IPolicyVO

  A representation of a set of guidelines, rules, regulations, or requirements to be enforced on a target. This set of conditions defines the circumstances under which a request for access to the target is permitted or denied.

- IRoleVO

  A representation of an abstract role used to represent users or groups that can perform actions in a policy. The role remains abstract until it is mapped to one or more users or groups using the IRoleMappingVO object.

- IRoleMappingVO

  A representation of a mapping of an abstract role to one or more users and groups for a given service. This mapping defines the users and groups used with policy generation for a given service.

- IRuleParameterVO

  A rule parameter represents an unbound piece of information referenced in an IPolicyVO. For example, an authorization policy might require the authenticated user's “department” to grant or deny access. An IRuleParameterVO references this piece of information. During policy configuration, an IConfiguredRuleParameterVO is created that specifies how to get the value of “department” from the IT environment.

- IServiceDefinitionRegistryVO

  A representation of a service definition registry. A service definition registry is a repository of data and descriptor information regarding services in the IT environment. Such service registries include UDDI and WebSphere Service Registry and Repository. Tivoli Security Policy Manager uses the service definition registry abstraction represented by this class to support these and other service definition registry types.

- IServiceScopePointVO

  A service scope point is usually the top of the service or resource tree and it represents the service as a whole.

- IPolicyDistributionResultVO

  The status of a policy distribution operation (including policy removal) is captured in a policy distribution event object. A policy distribution event object contains one or more policy distribution records, each of which holds the distribution status for one type of policy, configured for one type of PDP, and targeted to one endpoint, also called a PDT. Policy distribution to a given PDT might consist of multiple steps (notification and policy retrieval, for example).

  The IPolicyDistributionResultVO interface represents information associated with a policy distribution event, including policy removal, for a single step in policy distribution to a single PDT. A step might be notification of a policy update, or policy retrieval. Policy distribution result information includes the time and status of the distribution step, and any information that might have been returned by the PDT.

- IExportedPolicyDataVO

  The IExportedPolicyDataVO contains policy data in an un-configured state. An instance of the IExportedPolicyDataVO object is used to export policy data to a file.

8.2.3 Plug-ins

In this section, we discuss the following plug-ins:

- Policy type plug-in
- Policy distribution plug-in
- Service registry plug-in
- Service plug-in

Policy type plug-in

Tivoli Security Policy Manager ships with many supported policy types, such as the message protection policy type. It may be necessary to define your own policy type to represent a custom type of policy. To implement a custom policy type plug-in, the following interfaces from the `com.ibm.tspm.datamodel.plugins.api.policy` package should to be implemented:

- IPolicyAuthor

  An interface for creating policies suitable for distribution. The interface defines methods for converting a policy to an effective policy and converting a policy to an exportable format. This interface must be implemented by all policy plug-ins that require a security policy to be authored. To generate an effective policy, an IPolicyPack data structure is provided as an input and contains the basic data required to build an effective policy. Policy plug-ins should use the policy pack rather than retrieving policies directly from the database.

The effective policy object can be exported to an external representation, such as an XACML or WS-MessageProtection policy, depending on the policy type. The IExportedPolicyDataVO, IEffectivePolicyDataVO, and IPolicyVO interfaces from the data model are also used.

- IPolicyConfigurator

  An interface for optionally transforming policy into a format that is suitable for PDPs. Policy is authored in a standard format during policy authoring and configurators may optionally modify the generated policy to be appropriate for the PDP evaluating the policy. The interface is also called by Tivoli Security Policy Manager to determine whether a policy has been configured and is ready for distribution. The interface defines a method that allows the policy configuration flow to determine if any additional configuration is necessary to allow policy generation and distribution, and a method to transform an effective policy to add additional configuration specific to a policy decision point.

- IPolicyFactory

  An interface for creating, modifying, deleting, and parsing policies, and validating policy associations. It is responsible for returning a policy value object, performing any outside operations needed for effective use of the policy, setting the extension identifier, and setting the plug-in identifier. The plug-in returns a policy value object in the case of the createPolicy and modifyPolicy methods. The server is also responsible for removing the requested value object from the repository in the case of the deletePolicy method.

  The `com.ibm.tspm.datamodel.vo.impl.PolicyVOFactory` class can be used to create an object that implements IPolicyDistributionTargetVO.

- IPolicyType

  An interface to return static information about a policy type. The interface defines methods to exposed details such as name, description, and dialect of a policy type. These methods are used to display information in the user interface.

Policy distribution plug-in

Policy distribution plug-ins allow Tivoli Security Policy Manager to manage the distribution of effective policies to PDTs. A PDT plug-in can be written to customize the mechanism for distributing and removing a policy to and from an endpoint. These plug-ins are an important feature of the product, as they allow Tivoli Security Policy Manager to distribute policy to any custom endpoint for which a PDT can be written. For example, a PDT plug-in may be required to facilitate the distribution of policy to an endpoint using a particular proprietary API.

To define a policy distribution target plug-in, the following interfaces should be implemented from the `com.ibm.tspm.datamodel.plugins.api.dist` package:

- IPDTDistributor

  An interface for distributing and removing policy to a PDT. It uses interfaces defined in the data model, such as IPolicyDistributionTargetVO, to represent the destination for a given policy, IServiceScopePointVO to represent a target service, and IPolicyDistributionResultVO to represent the result of policy distribute or remove operations.

- IPDTFactory

  An interface for creating, modifying, and deleting PDT objects. It is responsible for returning a PDT value object, performing any outside operations needed for the effective use of the distribution target, setting the extension identifier, and setting the plug-in identifier. An implementation of the interface must return a policy distribution value object in the case of the createDistributionTarget and modifyDistributionTarget methods. The server is also responsible for removing the requested value object from the repository in the case of the deleteDistributionTarget method.

  The `com.ibm.tspm.datamodel.vo.impl.DistributionVOFactory` class can be used to create an object that implements IPolicyDistributionTargetVO.

- IPDTType

  An interface to return static information about a PDT type. A class that implements the IPDTType interface can return name and description details of a PDT type plug-in.

Service registry plug-in

Service definition registries are the existing repositories from which service definitions can be retrieved. Examples are WebSphere Service Registry and Repository, a database, WebSphere Application Server, and so on. To define a service registry plug-in, the following interfaces can be implemented from the `com.ibm.tspm.datamodel.plugins.api.registry` package. Several interfaces from the data model are used, including IServiceDefinitionRegistryVO, IServiceScopePointVO, IClassificationVO, and IRoleVO.

- IServiceDefinitionRegistry

  An interface that represents a service definition registry and defines methods to retrieve a list of services from a service registry, retrieves a specified service definition from a service registry, validates service definitions from a service registry, and retrieves roles and classifications associated with a specified service in a service registry.

- IServiceDefinitionRegistryFactory

  Interface for creating, modifying, and deleting service definition registry objects. The plug-in is responsible for returning a service definition registry value object, performing any outside operations needed for effective use of the policy, setting the extension identifier, and setting the plug-in identifier. The plug-in returns a service definition registry value object in the case of the createServiceDefinitionRegistry and modifyServiceDefinitionRegistry methods. The server is also responsible for removing the requested value object from the repository in the case of the deleteServiceDefinitionRegistry method.

  The `com.ibm.tspm.datamodel.vo.impl.RegistryVOFactory` class can be used to create an object that implements IServiceDefinitionRegistryVO.

- IServiceDefinitionRegistryType

  An interface for returning static information about a service definition registry type, such as name, description, and version of the service definition registry. These methods are used to display information in the user interface.

Service plug-in

Tivoli Security Policy Manager ships with many supported services, such as the J2EE service type or the SharePoint service type. It may be necessary to define your own service type to represent a custom type of service. The interfaces defined in the `com.ibm.tspm.datamodel.plugins.api.service` package should be implemented to define a service type plug-in.

- IServiceFactory

  An interface for creating, modifying, deleting, and importing services. It is responsible for returning a service scope point value object, performing any outside operations needed for effective use of the policy, setting the extension identifier, and setting the plug-in identifier. The plug-in returns a service scope point value object in the case of the createService and modifyService methods. The server is also responsible for removing the requested value object from the repository in the case of the deleteService method.

  The `com.ibm.tspm.datamodel.vo.impl.ServiceVOFactory` class can be used to create an object that implements IServiceDefinitionRegistryVO.

- IServicePoint

  An interface for defining a service point for a service type plug-in. A service point defines a node in a service and can be a service scope point, a policy attachment point, or a structure point.

- IServiceType

  An interface to return static information about a service type. The interface defines methods to exposed details such as a name and description of a service type. These methods are used to display information in the user interface.

8.3 Application integration

We have discussed several interfaces that can be used to integrate applications with Tivoli Security Policy Manager, such as the Tivoli Security Policy Manager authorization API, the Tivoli Security Policy Manager web services authorization interface, and a pattern for implementing authorization using a RESTful pattern.

In this section, we examine three application level integrations, one based on the Java based authorization API, a WebSphere Application Server portlet integration example that uses the Tivoli Security Policy Manager Tag library, and one that allows applications to integrate with C# and .NET based applications.

8.3.1 Integration with Java technology

Tivoli Security Policy Manager provides integration to support technology based on the Java Platform, Enterprise Edition (Java EE), such as the container level integration outlined in 6.2, “WebSphere Application Server” on page 161, or the JAX-RPC and JAX-WS integration outlined in 5.2, “Java Web Application Servers” on page 127. If a Java based application has security requirements that cannot be implemented using any of the above security mechanisms, Tivoli Security Policy Manager provides the Tivoli Security Policy Manager authorization API to allow security services calls from within Java based applications that support the interface.

Integration with WebSphere Application Server

Tivoli Security Policy Manager provides a mechanism to model Java EE application resources and actions, attach policy controlling the conditions under which those resources can be accessed, and evaluate that policy by invoking an authorization API. The Tivoli Security Policy Manager authorization API is built on two standards: *Java Authorization Contract for Containers* (JACC) and *eXtensible Access Control Markup Language* (XACML).

The authorization API allows applications to pass context information for use in an authorization decision. This context information, such as the authenticated user and resource being accessed, is converted into an XACML request and sent to the RTSS to be evaluated against the policy as shown in Figure 8-2. Context handlers are used to populate additional information that may be needed for the policy to reach a decision.

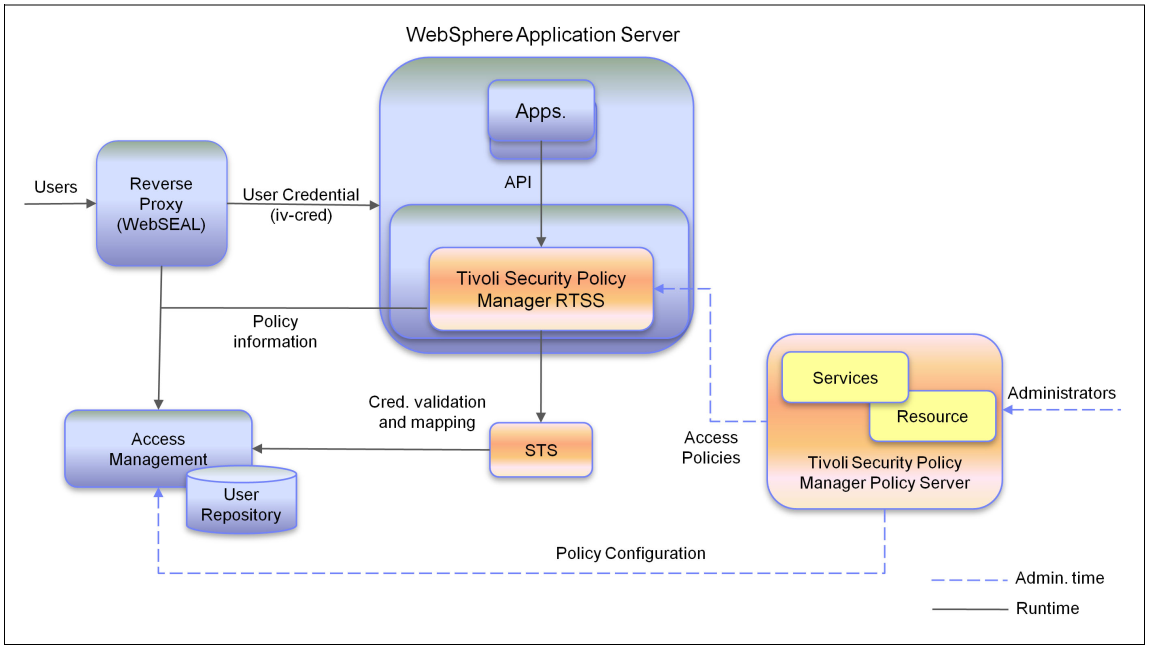


Figure 8-2   WebSphere Application Server integration

In the rest of this section, we study how an application is developed to use Tivoli Security Policy Manager authorization decisions in the context of the security policy management life cycle.

Policy modelling and simulation

The first step to externalize security is to model the resources in the application and the actions that are to be performed on them. These resources, known as *services* in Tivoli Security Policy Manager, and actions will be created in Tivoli Security Policy Manager, and a policy describing the conditions under which these resources can be accessed can be attached.

Using a healthcare provider organization as an example, let us consider an application that allows access to patient information. Various personnel have access to this application, but access needs to be restricted based on their role and the specific data being accessed.

In the healthcare scenario, the application resources have been modeled as follows:

- Patient information (such as billing address and phone numbers)
- Patient test results
- Patient prescription history
- Patient genomic data

After the resources have been modeled, unique identifiers can be created for each resource. These identifiers are what the application itself should pass to Tivoli Security Policy Manager at run time when performing a security check.

- Patient information could be allocated the identifier *patient-info*.
- Patient test results could be allocated the identifier *patient-test-results*.
- Patient prescription history could be allocated the identifier *patient-medications*.
- Patient genomic data could be allocated the identifier *patient-genomic*.

The set of actions to be performed on these resources can also be identified. Sometimes, a single action such as invoke is enough. In this use case, consider the use two actions: read and edit.

Importing resources and metadata

After the resources and actions have been modeled, the service structure can be created in Tivoli Security Policy Manager. An application service type should be created in the Tivoli Security Policy Manager console with the following attributes:

- The Application name is a human-readable name that is shown in the user interface. In this scenario, we use Patient Data Application for the application name.
- The Application ID is a context string that the application must pass at run time when performing a security check. In this case, patient-application is the appropriate value to use.
- Define resources in Tivoli Security Policy Manager console using the structure modeled above. The first step is to provide a unique identifier for the root service in the hierarchy. This is usually the same as the application ID from the previous step.
- When defining the service structure, the two fields for Element Name and Element ID are similar to the Application Name and Application ID fields from earlier. The Element Name is a human-readable name that is displayed in the Tivoli Security Policy Manager console, and the Element ID is what the application must pass at run time when making an authorization decision using the authorization API.

Policy authoring

In the next step, we need to define the policy or conditions under which access to the resources should be granted or denied, and attach them to the specific service definitions. For this example, let us consider the following policies:

- Doctors can edit prescriptions and test results for their own patients.
- Doctors can read all patient data.
- Nurses can read all patient data.
- Administrative assistants can read and edit patient information.

From this simple list of policies, the following roles can be identified:

- Doctor
- Nurse
- Administrative Assistant

Rule parameters can also be identified from this information. A rule parameter is a piece of information required to make a decision, which is provided by the calling application or retrieved dynamically from an information source in the runtime environment.

In this case, a rule is required that expresses the following constraint: Doctors can edit prescriptions and test results for their own patients.

Two pieces of information are needed to model this rule: the identifier of the currently logged in user, and the doctor who is the primary care physician for that patient, which we assume is the currently authenticated user. This gives us the following rule parameters:

- Patient's physician
- Currently authenticated user

These roles and rule parameters can be created under the appropriate panels in Tivoli Security Policy Manager.

Policy configuration

After the relevant policy has been created and attached, this policy needs to be configured. The configuration step is where the abstract role definitions are mapped to groups in the a directory, and the rule parameters are bound to a concrete information source.

The most important piece of configuration for this information is the binding of the rule parameters to information sources. Remember that there are two rule parameters:

- Patient's physician
- Currently authenticated user

Retrieving the patient's physician requires the identifier of the patient whose data is being accessed. This is the first instance of a piece of data the application itself must pass when making an authorization decision. It is important to define the exact identifiers under which information like this is passed. In this instance, the application will use the identifier patient-id.

Depending on the exact application environment, this patient ID could use a search parameter for an LDAP or JDBC search. The Tivoli Security Policy Manager product documentation covers the necessary steps that are required to set up these types of searches. The important point to remember is that the patient-id attribute should be specified as the Key attribute in the decision request.

Policy distribution

After a policy is configured, it can be distributed to the appropriate RTSS PDT.

Policy enforcement

Now that the services and policies have been modeled and configured, the information that must be passed in by the enforcement point becomes clear. To summarize, the following data is required:

- The application ID of “patient-application”
- A resource identifier in the set [“patient-info”, “patient-test-results”, “patient-medications”, “patient-genomic”]
- An action identifier in the set [“read”, “edit”]
- The identifier of the patient whose data is being accessed as the attribute “patient-id”

The mechanism by which a customized PEP is written depends on the exact environment on which the application runs. It could be an EJB if running within a J2EE application, a Spring bean if running in Spring,; or a Java class called by application code.

Section 8.1, “Runtime security services interfaces” on page 214 outlines the Tivoli Security Policy Manager API. There are several design decisions that need to be considered when writing the code that calls the Tivoli Security Policy Manager API, such as determining how the identity of the currently authenticated user is passed to the decision engine. This determines the implementation of the EvaluationContext object that is used. The EvaluationContext is the object that maintains references to the objects required to extract information from the running environment.

There are a number of possible scenarios:

- The application is deployed in WebSphere Application Server, with application security enabled, and using WebSphere authentication. In this case, the API has the ability to automatically extract the authenticated user from the WebSphere Application Server container. In this case, the calling application should use a ContainerEvaluationContext implementation, as shown in “Authorization using a container-authenticated subject” on page 216.
- The application is deployed in WebSphere Application Server, but application security is disabled and a custom authentication method is being used. This scenario is common when a framework, such as Spring, is being used. For this case, the application must create the user objects, so the ApplicationEvaluationContext implementation, as shown in “Using an application-provided subject” on page 217, should be used instead.

  Considering re-use: It is important that the EvaluationContext is created once and re-used across multiple calls. It is thread-safe, so sharing across threads is encouraged as well. In many use cases, object state and helper code is initialized on creation of the context object, and re-using the instance ensures this initialization happens only once. Re-creating the EvaluationContext instance on every request can result in a significant performance impact.

- If the ApplicationEvaluationContext is being used, then you first need to set up the user context, as shown in “Using an application-provided subject” on page 217.
- All attributes are provided through implementations of the IAttributesHandler interface. These implementations are registered with the ApplicationAttributes mapped to predefined keys in the EvaluationContext's handlers, as discussed in “Providing additional resource, action, and environment attributes” on page 219. In the sample use case, passing the patient identifier requires setting up a callback mechanism. Attributes can be provided to the Resource, Action, and Environment request sections.

Integration with other platforms based on Java

If the Java platform does not support Tivoli Security Policy Manager, a RESTful approach can be used to implement a custom authorization interface, as discussed in “Tivoli Security Policy Manager API with REST-based interface” on page 231.

8.3.2 Integration with WebSphere Portal

Portlets are special JSPs, and in theory the portlet support applies to other types of JSPs as well. However, we only consider portlets listed in this section. Tivoli Security Policy Manager can enforce authorization for contents within a JSP page through a custom tag library, so this is a form of programmatic security rather than declarative security.

The tag library is installed as part of the RTSS client. To use this library, the following steps must be completed:

1. The application developer must identify the actions and resources within a page content that can be governed by Tivoli Security Policy Manager policies.
2. The application developer implements the page using the tag library to customize the visibility of page content.
3. A security administrator imports the actions and resources into Tivoli Security Policy Manager. This information is used to author policies that govern user privileges.
4. Finally, at run time, the JSP tag library invokes the RTSS client to obtain the authorization decision and entitlements.

We now discuss these steps one by one.

Identifying portlet authorization resources

The portlet and its associated resources need to be defined to Tivoli Security Policy Manager as a service to enable policy attachment. Tivoli Security Policy Manager provides an Eclipse plug-in for this purpose. It is called the Tivoli Security Policy Manager Interchange Editor. The Interchange Editor allows you to define a portal service and store it in an interchange file.

The service description for portlets can contain one or more services, which in turn consist of portlets, actions, and resources within the portlets. The interchange file may also contain roles. It is good practice to use the Interchange Editor side by side with the actual portal source code and use copy and paste to avoid typing errors when entering the matching resource names.

The resulting interchange file is stored for later use by the Tivoli Security Policy Manager administrative console.

Implementing authorization using the tag library

The tag library is discussed in 8.1.2, “JSP tag library” on page 223 in detail. The application developer adds the tags required to implement the authorization calls.

Importing the interchange file

The XML interchange file, which was created using the Eclipse plug-in, must be imported into Tivoli Security Policy Manager using the administrative console. The components need to exist on the same system where your web browser is located. Chapter 3. “Managing services”, of *Tivoli Security Policy Manager Version 7.1 Administration Guide*, SC23-9476 has a sub-section named “Importing services from an interchange file” that provides details about how to import the portlet as a service.

After the service is imported, a policy can be attached to it. The policy must then be distributed to the local RTSS on the portal server.

Runtime enforcement

Runtime integration is accomplished by providing a WebSphere Application Server JACC based container integration. The tag library must exist in the correct place. It calls the RTSS transparently. The resulting authorization decision determines the content that gets rendered by the portlet.

8.3.3 Integration with Microsoft technology

Tivoli Security Policy Manager provides several integration options to support Microsoft based technology. The .NET runtime security service client integration can be used to externalize security from C# and .NET based applications. The container level integration, which is discussed in 6.3, “Microsoft environment” on page 169, uses the .NET runtime security service client integration to provide the IIS integration with the SharePoint Authorization HTTP Module, ASP.NET integration with the .NET Role Provider, and MOSS integration with the SharePoint Item Event Receiver. If your application does not support the use of the .NET Role Provider, you can use the .NET integration with the .NET runtime security service client to externalize security policy management from your applications.

C# and .NET application integration

Using the .NET runtime security services client, you can integrate Tivoli Security Policy Manager authorization and entitlements decisions directly into your .NET based applications. The Tivoli Security Policy Manager .NET Role Provider and the Enforcement Point for Microsoft SharePoint (which includes the SharePoint HTTP Authorization Module and SharePoint Item Event Receiver) are examples of using the .NET runtime security services client to provide integration with Tivoli Security Policy Manager.

The .NET runtime security services client provides a programmatic framework for calling the Tivoli Security Policy Manager runtime security services web services for authorization and entitlements decisions in your .NET applications. The client uses the Windows Communication Foundation (WCF) to transport XACML over SOAP. Your .NET application is then able to use the returned decisions to enforce access requests.

A typical usage of the .NET runtime security services client includes custom .NET applications or specialized extensions, such as a *SharePoint Webpart*. It may be necessary to use the .NET runtime security services client when the Enforcement Point or Role Provider do not provide the require level of integration, or they cannot be configured into your application. In both usage scenarios, the configuration and code required to initialize and call the .NET runtime security services client are the same.

When using the .NET runtime security services client directly in your code, an accompanying custom service and associated security policy must be created manually in the Tivoli Security Policy Manager console. The structure of the custom service and the attached security policy is dictated by the outcome from the *policy modelling and simulation* phase of the policy life cycle model discussed in previous chapters.

The .NET runtime security services client can be used to implement in or migrate security policy to Tivoli Security Policy Manager for your .NET applications, or it can be implemented with the initial stages of application development. It is not necessary to create the custom service before coding begins, but there is a fixed relationship between the application's authorization and entitlements requirements and the custom service created in the Tivoli Security Policy Manager console.

Service creation and policy authoring

The structure of the Tivoli Security Policy Manager custom service to be created will be driven by the security requirements for the application determined during the *Policy Modelling and Simulation* phase of the policy life cycle model. With the service defined, the required security policy elements must be created and attached within the custom service. If your security policy requires custom attributes from your .NET application or from an external source such as a directory server, they can be added here and mapped during the *Policy Configuration* phase.

Figure 8-3 shows an example of a simple custom service that represents a business system of inventory management. We use this example to demonstrate the use of standard attributes from the request, such as action-id, the use of custom attributes populated by the .NET application, and also external attributes from a directory.

```
Inventory Management
   |
   + ==> Current
   |
   + ==> On Order
   |
   + ==> Future Demand
   |
   \/
```

Figure 8-3 Sample custom service for .NET applications integration

The custom service shown in Figure 8-3 was derived from the following business requirement:

> "The ability to view (action) any of the inventory management system categories (resource) has to depend on whether you are an employee or supplier (application role), and which widget you order or supply (external attribute)."

The following security policies have to be created and attached to meet these requirements:

- Current Inventory Access policy, which permits all subjects to use the view action on the current widgets on hand. This policy is attached to the Current element.
- Employee Order Access policy, which permits employees to use the view action on all widgets that are on order. This policy is attached to the On Order element.
- Supplier Order Access policy, which permits suppliers to use the view action on all widgets that are on order for the item that they supply. This policy is attached to the On Order element.
- Future Demand Access policy, which permits supervisors to use the view action on widgets that may need to be ordered to fulfill future demands and avoid back orders.

With our custom service and security policies defined, we now explore how the .NET runtime security services client can be used in the .NET applications to integrate with Tivoli Security Policy Manager. The code used in this example comes from a sample SharePoint Webpart, but the same Tivoli Security Policy Manager integration can be applied in any .NET application.

Namespace references

To use the .NET runtime security services client in your .NET application code, you must reference the namespace of the client, which also provides access to the interfaces required to populate attributes in the request:

```
using IBM.Tivoli.SecurityPolicyManager.RTSS;
```

The initialization of the .NET runtime security services client and the interfaces required to populate custom attributes in the request are discussed in later sections.

Custom handlers

The .NET runtime security services client populates standard XACML attributes of action-id and resource-id as part of the request. If your security policy requires context specific or custom attributes, such as resource, session, or action, your .NET application must populate those attributes for the .NET runtime security services client.

Populating additional attributes is achieved by using *handlers*, which provide the necessary attribute-ids and associated values, which are populated in the XACML request. Handlers are custom coded modules that implement either the Subject or Attribute handler interface. A number of ready to use subject and attribute handlers are provided with the .NET runtime security services client.

Figure 8-4 shows the relationship between a custom .NET application, the .NET runtime security services client, and custom subject and attributes handlers.

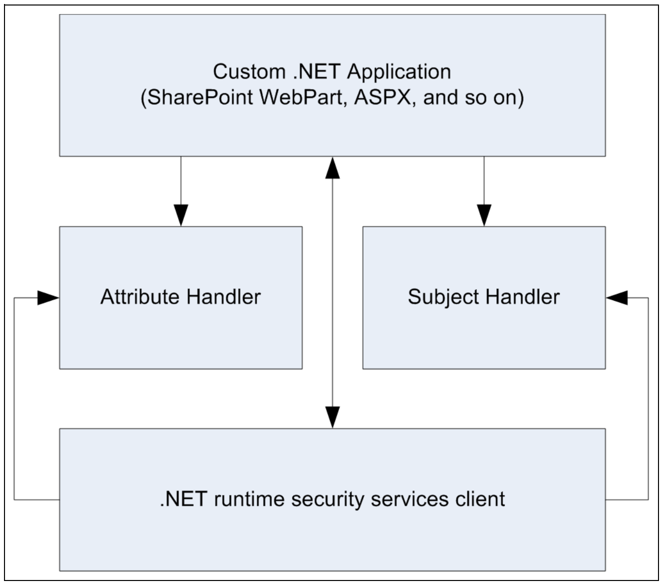

Custom .NET Application
(SharePoint WebPart, ASPX, and so on)
Attribute Handler
Subject Handler
.NET runtime security services client

Figure 8-4 Using .NET runtime security services client with custom handlers

Subject handler interface

Referring back to our security policies, our Current Inventory Access policy requires the subject-id from the XACML request to determine view access. The subject-id must be populated by a Subject Handler, which the .NET runtime security services client calls to populate the subject XACML attribute.

To build your own Subject Handler, you must implement the SubjectHandler interface, as shown in Example 8-21.

Example 8-21 Subject Handler interface

```
public interface SubjectHandler
    {
        string GetSubjectId();
        string[] GetGroupIds();
        string[] GetRoles();
        XmlElement GetAuthenticationToken();
    }
```

If a particular method is not applicable to the .NET application or subject handler, it should return null. The subject-id and group-ids should generally contain the full distinguished name (DN) of the subject so they can be used to map Application Roles during the policy authoring and policy configuration life cycle phases.

A number of ready to use subject handlers are provided with the .NET runtime security services client, or you can write your own subject handler by implementing the SubjectHandler interface to have full control over populating the attributes from .NET application.

Which subject handler the .NET runtime security services client uses can be configured in the `Web.config` file for your .NET application, or programmatically after initializing the client. If you need to use the .NET runtime security services client in multiple places in your .NET application and require each instance to use a different subject handler, you should configure the handlers programmatically.

The programmatic configuration of a subject handler is discussed in "Authorization client initialization" on page 262.

Attribute handler interface

To provide fine grained authorization decisions, our example security policies not only defined the inventory management systems that could be accessed, but also provided some specific context around which widget type was being accessed. This is an example of a custom resource attribute using session data.

Similar to the use of subject handlers, the .NET runtime security services client can be configured to call custom attribute handlers to provide custom subject, action, and resource attributes in the XACML request in addition to the action-id (view in our example) and resource-id (the inventory system in our example).

Subject, Action, and Resource Attribute handlers are built using the same interface and are also configured through the .NET application's `Web.config` file or programmatically. A Subject Attribute handler can be used to provide extended subject attributes, such as a Department code or Job ID, whereas a Subject handler is used only for subject-id, group, and role attributes, which is described in "Subject handler interface" on page 258.

To build your own Attribute Handler, you must implement the AttributeHandler interface, as shown in Example 8-22.

Example 8-22   AttributeHandler interface

```
public interface AttributeHandler
    {
        string GetIssuer();
        string[] GetAttributeIds();
        object[] GetAttributeValues(string attributeId, EvaluationContext
context);
    }
```

The .NET runtime security services client calls `GetAttributeIds()` for the array of attributes the handler provides and then calls `GetAttributeValues()` for each of the attribute IDs. If an attribute does not have a value for the context is it being called, the value returned should be null. You can optionally return a custom string representing the issuer of your attributes, or if this is not needed, return null.

The .NET runtime security client will make available to your code EvaluationContext, which provides you access to attribute data for the request, such as the action, resource, and subject, as these may be significant when determining values for the handler.

For our example, we build our own Resouce Handler (Example 8-23), which populates a custom attribute called widgetType in the Resource XACML element. The value for widgetType is set programmatically, but the handler could also determine this value by accessing session or environment data in which the .NET application is running.

Example 8-23 Custom ResourceHandler implementation

```
internal class WidgetAttributeHandler :
IBM.Tivoli.SecurityPolicyManager.RTSS.AttributeHandler
    {
        [ThreadStatic] private string widgetType_;
        string AttributeHandler.GetIssuer()
        {
            return null;
        }
        string[] AttributeHandler.GetAttributeIds()
        {
            return new string[] { "widgetType" };
        }
        internal string WidgetType
        {
            get
            {
                return widgetType_;
            }
            set
            {
                widgetType_ = value;
            }
        }
        object[] AttributeHandler.GetAttributeValues(string attributeId,
EvaluationContext context)
        {
            return new object[] { widgetType_ };
        }
    }
```

Now that we have added the custom handlers that populate specific attributes in the XACML request, we can complete the policy configuration. This phase of the policy life cycle model allows us to map attributes from the XACML request to Rule Parameters. We can also use values from external sources in Rule Parameters.

Policy configuration

During policy configuration, we are able to define from where our Rule Parameter values are mapped. In our example, we had a requirement to evaluate access decisions based on the widget that a supplier provided, which was attached at the On Order element.

The widget being accessed in our .NET application is populated by the custom resource handler. The value for this is contained within the widgetType custom attribute of the Resource XACML attribute in the request.

We need to compare this to the supplier's list of widgets that they provide, which are contained in a directory server. Using their subject-id from the Subject XACML attribute, we can retrieve their department attribute and determine if this value contains widgetType.

This action demonstrates using application context or session data from the runtime request to provide authorization against a predefined business system. Our Supplier Order Access policy attached to the On Order element defined an Application Role based on the group-ids contained within the Subject XACML attribute.

If these were not populated by the subject handler, this could also be achieved by using an additional Rule Parameter and determine a users group memberships at run time from the same or different directory server or database and use the subject-id to perform the lookup.

After our policy has been configured and deployed, we can call Tivoli Security Policy Manager for delegated authorization and entitlements from within our .NET application code.

Authorization client initialization

Before we can call out to Tivoli Security Policy Manager to make authorization and entitlements decisions in a .NET application, we must first initialize the .NET runtime security services client.

During initialization, the .NET runtime security services client will read its configuration from the calling application's `Web.config` file, which will add the RTSS web services endpoint, configure the security and authentication settings for the transport, and register any custom subject and attribute (action or resource) handlers. After initialization, handlers can be added or updated programmatically.

In the sample code shown in Example 8-24, you can see how line one instantiates a new AuthorizationClient object, which should be a local member and be reused without re-initializing it within your application's class.

Line two demonstrates how to register a custom subject handler programmatically. The handler must be accessible to your class, either by using a reference (`using`) or as an internal class.

The third and fourth lines add custom resource and action handlers. When adding a handler, you must define the handler type from the `EvaluationContext.Section` enumeration and an instance of the required handler. The subject and attribute handlers you register programmatically should also be local members so you can access them throughout your class to update the necessary values at run time.

Example 8-24 Authorization client initialization

```
_authzClient = new AuthorizationClient();
_authzClient.evaluationContext.SubjectHandler = new SubjectHandler();
_authzClient.evaluationContext.AddAttributeHandler(EvaluationContext.Section.Re
source, new ResourceAttributeHandler());
_authzClient.evaluationContext.AddAttributeHandler(EvaluationContext.Section.Ac
tion, new ActionAttributeHandler());
```

During initialization, any errors encountered by the .NET runtime security services client are output to its TraceSource, and any exceptions are passed back to your code to handle.

In our sample, shown in Example 8-25, we create a local member for our resource handler and register it with the authorization client.

Example 8-25 Attribute Handler initialization and registration

```
widgetHandler_ = new WidgetAttributeHandler();
_authzClient = new AuthorizationClient();
```

```
_authzClient.EvaluationContext.AddAttributeHandler(EvaluationContext.Section.Re
source, widgetHandler_);
```

Authorization request

Making authorization requests to Tivoli Security Policy Manager is a single line call after the .NET runtime security services client is configured and initialized:

```
bool accessAllowed = _authzClient.IsAccessAllowed(serviceName,
resource, action);
```

When creating a custom service, the Application Name field is what is displayed in the Tivoli Security Policy Manager console and the Application ID is used to reference the application in XACML requests. In our example, our Application Name was Inventory Management, so we assume our service name was entered as inventory-management, which we use in the .NET runtime security services client call as the serviceName.

Child elements created in the custom service require an Element Name and Element ID. Like its parent, the name is displayed in the Tivoli Security Policy Manager console and the ID is used to reference it. In our example, our element names were Current, On Order, and Future Demand. We assume the corresponding element IDs are current, on-order, and future-demand, which we use in the .NET runtime security services client call as the resource.

Finally, we must specify the action that corresponds to the function being performed in your .NET application. This action must be one the actions selected or defined when creating the custom service. For our example, the only action we specified was view.

Using our example, if we want to authorize a supplier, Supplier One, who manufactures widgetType1, the code shown in Example 8-26 is used. Remember that widgetHandler_ is our local instance of the custom resource attribute handler, which our application populates and the registered subject handler determines the subject at run time.

Example 8-26 Sample authorization call

```
widgetHandler_.WidgetType = “widget1”;
bool accessAllowed = _authzClient.IsAccessAllowed(“inventory-management”,
“on-order”, “view”);
```

The return value is true only for operations that evaluate to permit, with all other results being treated as false or unauthorized.

Entitlements request

If we are instead interested in what actions a user can do in our application, rather than authorizing a specific transaction, we can use an entitlements call. Entitlements differ from authorization calls in that they represent a collection of authorization decision evaluations at a given point in time using the available attribute data. As such, entitlements are best suited for decisions that do not rely on session specific or dynamic attributes.

When making an entitlements call, you must provide the serviceName (application ID of the custom service) and the action for the request. This request returns all the elements within the custom service that evaluate to permit for the given action.

```
string[] entitlements = _authzClient.GetEntitlements(serviceName, action);
```

In our example, if we want to determine all inventory management systems our supplier can access, we could use the entitlements call shown in Example 8-27. Remember that if we have a security policy that requires specific attributes from the original request, we must populate those attributes, or the policy may evaluate to false. The entitlement decision is a policy combination of all applicable policies attached to an element.

Example 8-27 Sample entitlements call

```
widgetHandler_.WidgetType = “widget1”;
string[] entitlements = _authzClient.GetEntitlements(“inventory-management”,
“view”);
```

Using our business rules, the entitlements returned should include the custom service element IDs of current and on-order.

If we did not populate the WidgetType in our custom handler, the policy for On Order would not have evaluated to permit and only current would have been returned to our application.

Conclusion

As you can see by following our example, the use of the .NET runtime security services client can provide your applications with flexible, context rich, and reusable security policy across all your .NET applications.

The client can be used for stand-alone and for web based applications, such as SharePoint WebParts and in ASPX pages. The client can also be used to build specific integration points, such as the SharePoint Authorization HTTP Module, SharePoint Item Event Receiver, and .NET Role Provider.

8.4 Conclusion

In this chapter, we discuss the application level approach, the Tivoli Security Policy Manager interfaces for application integration, and several examples of application integration, including integrations with Java and .NET technology.

Several Tivoli Security Policy Manager interfaces are discussed, including the RTSS authorization API, the policy management API, and a pattern for implementing a security interface using a RESTful approach.

9

Deployment considerations

In this chapter, we discuss a number of considerations for deploying Tivoli Security Policy Manager, including:

- Business considerations
- Deployment considerations
- Deployment architecture
- Application integration considerations

In each of these sections, we explore the topic in more detail and answer potential questions for each of the architectural decisions. Deployment considerations and approaches for implementing a successful project are discussed along with the architectural aspects of a deployment that should be considered.

9.1 Business considerations

There are a number of high level business drivers that motivate an organization to implement a security policy management initiative. These drivers are outlined in Chapter 1, “Business drivers and foundation for IT security policy management” on page 3 and include:

- Addressing governance
- Compliance and data security
- Risk management and cost containment

These business drivers should be clearly understood by all stake holders and a clear strategy for addressing them should be implemented at the business level. Addressing issues such as governance, compliance, data security, and risk management are not stand-alone and one-off projects, as they are continually evolving aspects of the business. Such initiatives require the continual support of the business and evolve as business needs change.

Projects to transform how policy is managed within an organization tend to touch many parts within the organization, including IT operations, the security teams, application teams, application development, executive management, and so on. It is important that a clear vision of the initiative is defined and all involved parties in the organization agree before such a project is attempted.

9.1.1 Business use cases

Let us now spend some time to discuss three business use cases:

- Cloud computing
- Portal deployment
- Application data

Each implementation has unique requirements, so a deployment can consist of one or more of these use cases. It is useful to align a project with at least one of these broad use cases for communication with stake holders, which can help to communicate the purpose of the project to all stake holders, both technical and non-technical, and ensure commitment.

Cloud computing

Organizations are continually seeking new ways, such as service-oriented architecture (SOA) and cloud computing initiatives, to deliver applications and services efficiently and cost effectively. These alternatives to traditional IT infrastructures can help reduce IT and application development costs, increase opportunities for collaboration, and drive business growth.

At the same time, however, they can create new vulnerabilities, exposing access to applications, data, and services beyond traditional organizational boundaries. Organizations therefore require more than traditional IT security to manage and protect access to these applications, data, and services.

Tivoli Security Policy Manager can help in cloud and SOA use cases to provide the following items:

- Secure access to on- and off-premise deployments of applications and services by reducing the risk of inconsistent security policies.
- Centrally managed user access in SOA deployments where the introduction of web services is transforming the IT environment.
- Secure access to hybrid cloud deployments, in which organizations use the benefits of the cloud while still protecting sensitive information that may be at risk.

The intermediary pattern outlined in Chapter 5, “Intermediary level integration” on page 121 discusses patterns that can be used to add security for cloud and SOA based implementations. This policy can be further enriched with data from external systems, such as user and data repositories, federation services, rules engines, and so on, as discussed in Chapter 4, “Integration with external systems” on page 93.

Portal deployment

Portal servers enable organizations to quickly consolidate applications and content into role-based applications, complete with search and personalization capabilities. This scenario provides users with an enhanced web experience, and organizations to more efficiently grow, extend, and reuse assets. Exposing this information brings security challenges about who, when, and where should have access to this data.

Tivoli Security Policy Manager can provide the following items:

- A personalized view of enterprise resources based on entitlements
- Additional authorization capabilities to presentation logic covering fine-grained controls to Portal applications
- Efficient centralized security policy management of portal resources

In Chapter 8, “Application level integration” on page 213, we discuss a pattern for how Portal resources can be personalized and protected based on a security policy.

Application data

Organizational data that was once stored securely on mainframe computers is now increasingly available through business services to a diverse set of internal and external users. This broader exposure creates unprecedented opportunities for better collaboration and service delivery, but it also puts critical data at greater risk for compromise. As composite applications become increasingly complex, and as organizations work to make information and services more accessible to more users, applications are potentially becoming more vulnerable.

Tivoli Security Policy Manager can provide the following benefits:

- Protect data by providing access on a need-to-know basis, particularly in collaborative environments where multiple types of users need access to different types of data.
- Enable security in data centers, where data volumes are growing at explosive rates and privileged users may pose particular risks for unintended access.
- Manage the risk of data disclosure across the organization, through a practical, life cycle based approach to data security.

The pattern outlined in Chapter 8, “Application level integration” on page 213 can be used to protect application based resources. The pattern outlined in Chapter 7, “Database level integration” on page 187 can be used to protect data stored in database resources. Using these to patterns together can allow an organization to centrally protect resources at the application and data layer, which provides a view of policy that cannot be achieved without using this approach.

Policy can be further enriched with data from external systems, such as user and data repositories, trust services, rules engines, and so on, as discussed in Chapter 4, “Integration with external systems” on page 93.

9.2 Deployment considerations

Before taking on a project to transform security policy management within an organization, all stake holders should have a clear understanding of the business motivations and drivers. Clear and achievable goals should be set that can be executed to show value to stakeholders in incremental steps adopting an *agile methodology*. As such, when a project touches many parts of an organization, it is useful to start with a small and well defined project scope and develop the project across several smaller iterations.

Figure 9-1 shows how a policy management project can evolve over several phases to continuously show value to an increasing number of stakeholders.

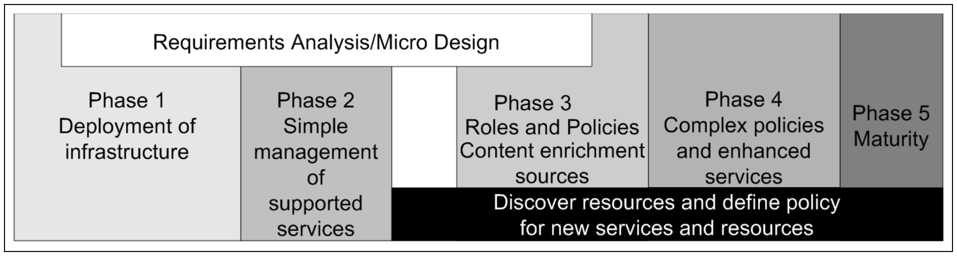

Requirements Analysis/Micro Design
Phase 1 Deployment of infrastructure
Phase 2 Simple management of supported services
Phase 3 Roles and Policies Content enrichment sources
Phase 4 Complex policies and enhanced services
Phase 5 Maturity
Discover resources and define policy for new services and resources

Figure 9-1 Phased implementation approach

The first phase focuses on the deployment of the basic components. In the second phase, simple capabilities that are delivered with the base product should be implemented to deliver basic functions. In the third phase, more complex policies and roles can be deployed and data from external sources can be integrated to create more meaningful and context aware policies. In the final stage, encompassing phases four and five, the system is moving toward a mature state, and integration with more complex systems can be implemented.

9.2.1 Identifying stakeholders

Before each iteration of the project, the involved stakeholders need to be identified. This action allows technical and organizational requirements to be planned prior to any implementation work. Typical stakeholders include, but are not limited to:

- Application owners (business line management)
- Policy owners (security team)
- IT architects
- IT operations
- Executive sponsors
- Business unit owners

Each of these stakeholders represent a significant role within any organization-wide project, and a successful implementation requires consensus and cooperation from all parts of the business.

9.2.2 Identifying policies, services, and data

To effectively plan a deployment, the required business level policies should first be determined. Because the policies involved can affect the scope of the project, it is important to determine these policies as soon as possible in the planning phase. From this information, the types of policies in scope can be formulated. A project may involve implementing message protection policies, authorization policies, or any other type of policies, which directly influences the level of effort required for this project, and understanding this situation early in the project phase can aid in the success of the project.

The services within the organization that policies may apply to need to be considered next. The number and type of services involved can affect the scope of the project as well. Specifically, the types of services that are required are an important consideration. A project may involve managing container security, web services security, application security, databases, and so on. By collecting and understanding information about your applications, you can better scope the integration options available and potentially discover the limitations.

Performing an inventory on your applications and data to scope their integration capabilities can help you find the most appropriate integration pattern, such as container, database, application, or intermediary. To start this process, answering some simple questions, like those below, can aid in this planning process:

- How does this application currently implement an authorization security model?
- What interfaces (if any) does this application provide to facilitate external authorization components?
- How will Tivoli Security Policy Manager be called into action?
- Can this application be hosted behind some other security device, such as Tivoli Access Manager WebSEAL or WebSphere DataPower?

By better understanding your services and their integration capabilities, you can then focus on what information can be used within your organization when evaluating access control decisions to your applications.

Sources of information that may be of use in adding business context information when managing a security policy should be identified as early as possible so that the potential options for expressing policy become clear. This situation may involve identifying user repositories, database repositories, security token services, identity and access management services, and other types of external systems. Having a clear understanding of this situation can help ensure that the correct design and implementation is developed.

When mining policies, services, or environmental data, identify the owners of the particular system or process. These owners need to be involved in the planning and deployment of a project.

9.2.3 Prioritizing services

After the policies, services, and data sources have been identified, the phases of the project can be planned. As Figure 9-1 on page 271 shows, services with standard support provide an accelerated return on investment (ROI) and should be implemented first. The requirements discovered in 9.2.2, "Identifying policies, services, and data" on page 272 should be mapped to the capabilities supported by the product.

The most strategic and least complex services, from both an organizational and technical perspective, should be implemented first. Avoid any complex integration work in the first stages of the project. This type of work should be added as the solution matures in later iterations of the project. Ensure that the implementation size is manageable and that applications are added one at a time to ensure the migration has been completed successfully before attempting to bring others online.

It is also important to ensure that applications continue to function as they previously did, using the same content enrichment sources as they previously did, such that the user experience or outputs are consistent. Use the capabilities of Tivoli Security Policy Server to integrate with this external sources rather than hardcoded implementations, so they can be easily modified as a security policy is reviewed and changed.

9.2.4 Identifying operational requirements

In any system architecture that relies on external components as part of the processing logic, it is essential to plan for and implement performance, availability, and continuity systems. Generally, such capabilities include *High Availability and Disaster Recovery* (HADR), *Service Level Agreements* (SLA), and *failover* (redundancy). These capabilities are typically non-functional requirements of the system, but are vital to the ongoing success of the implementation.

A Tivoli Security Policy Server solution is composed of a number of individual components, such as the Policy Server, runtime security services, and Management Console (TIP), which all require planning for their operational requirements. As with other infrastructure components to the broader systems, if these components become unavailable, access to any number of consuming services and applications may become unusable or unavailable.

Any system downtime ultimately leads to poor satisfaction levels and ongoing costs to bring the systems online again.

Many of the Tivoli Security Policy Server components use the underlying infrastructure of WebSphere Application Server, which includes the capabilities for clustering, load balancing, high availability, and failover. When deploying a large scale deployment of Tivoli Security Policy Server, it is vital that the capabilities of the underlying platform are installed and properly configured to facilitate HADR, meet SLAs, and provide redundancy for unplanned outages.

Planning for operational requirements and implementing these goals by using the underlying capabilities of the platforms such as WebSphere Application Server can assist in the ongoing availability and success of a Tivoli Security Policy Server deployment.

9.3 Deployment architecture

As mentioned before, Tivoli Security Policy Server consists of console, policy server, and runtime security services components. Depending on the IT environment and the use case required, there are different ways to deploy the components to meet those needs. We discuss some of the different and common deployment options in this section.

9.3.1 Single data center deployment pattern

A small to medium scale deployment is normally contained within one data center and the number of policy enforcement points are limited. For better performance and ease of management, use a simple pattern using one policy server and multiple runtime security services clients in local mode, as show in Figure 9-2.

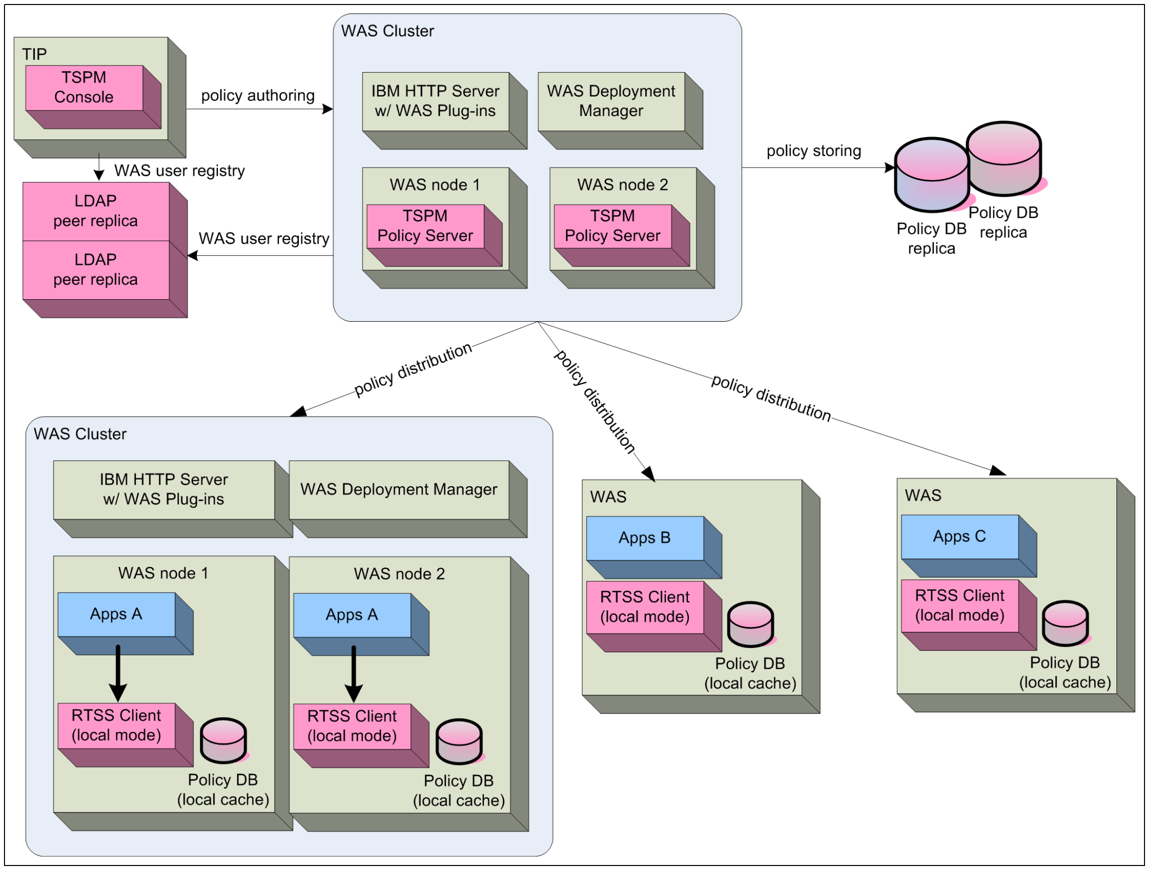


Figure 9-2 Single data center deployment pattern

The Tivoli Security Policy Manager console is the GUI interface used by security administrators to manage the product. Depending on the business requirements, multiple instances of the Tivoli Security Policy Manager console may be required to support high availability for administrative tasks. As the administrative components are separated from the runtime services, a highly available Tivoli Security Policy Manager console is not necessary to provide a high availability solution for the runtime components.

In a small to medium scale deployment, one Tivoli Security Policy Manager console is usually sufficient. However, regular system backups and a quick restoration capability is recommended in this case.

Multiple Tivoli Security Policy Manager policy servers can be set up within a WebSphere Application Server cluster, and multiple policy database replicas can be used to add high availability capabilities.

In this deployment pattern, an runtime security services client in local mode is used for both the policy decision point and the policy enforcement point. The runtime security services client in local mode is more efficient, because it maintains a local cache of the policy for its services. Authorization decisions can be made locally, avoiding the need to make a remote call over the network for an authorization decision.

The runtime security services local client is less dependent on the policy server. There is no need for continuous communication between the runtime security services local client and the policy server. It is also easier for security administrators to manage the runtime security services components, because there are no additional layers between the local client and the policy server.

9.3.2 Remote office deployment pattern

A larger organization has its IT typically deployed in multiple data centers or multiple regional offices. Using the Tivoli Security Policy Manager Runtime Security Service (RTSS) server in local mode can help reduce the network traffic over the WAN and simplify firewall settings between the remote offices and the corporate office. Figure 9-3 on page 277 shows a remote office deployment pattern. The policy server and policy database are set up in the corporate office, and a corporate LDAP server provides the necessary user and role definitions. RTSS local clients are deployed in the corporate office, as described in 9.3.1, “Single data center deployment pattern” on page 275.

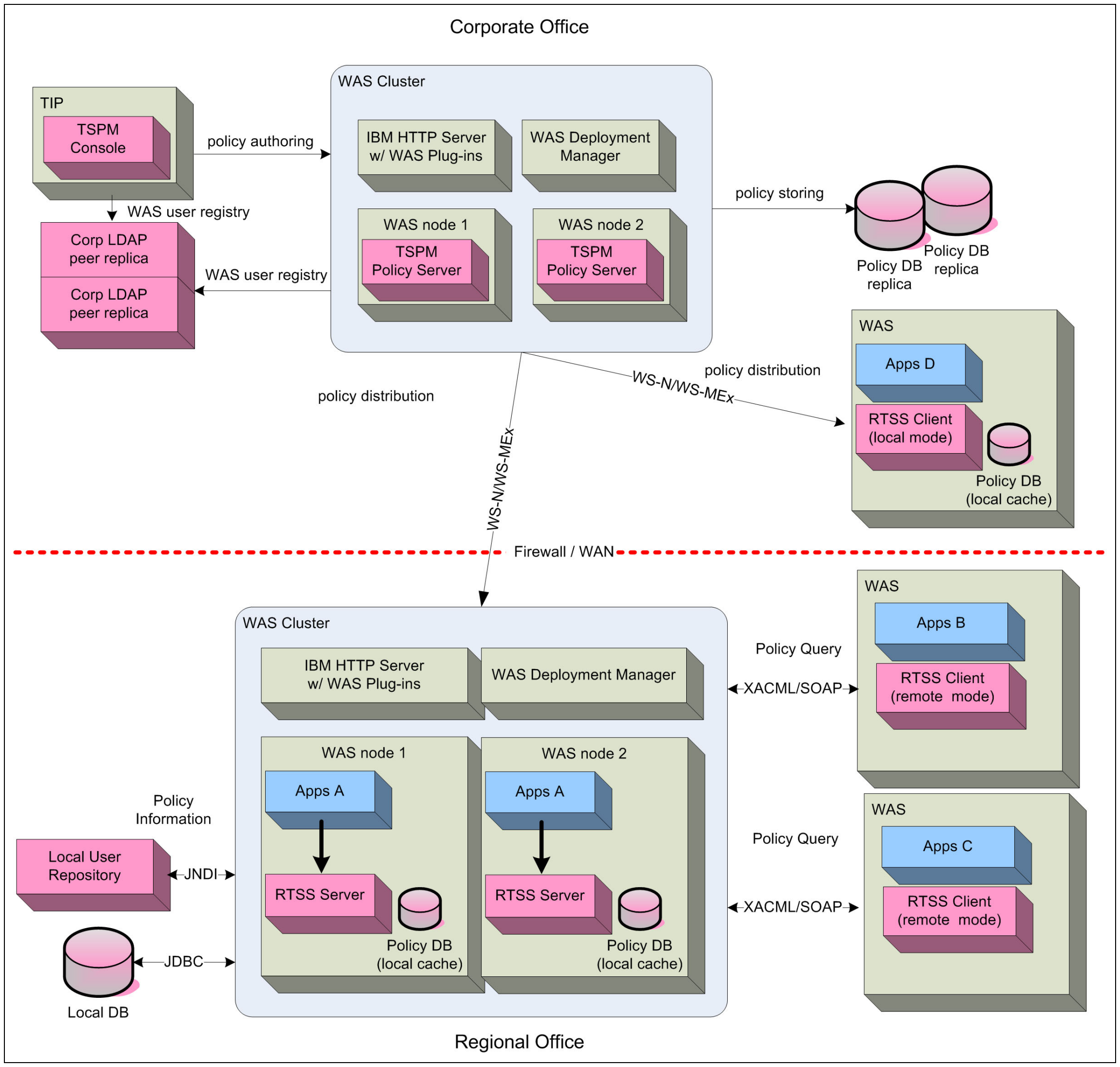


Figure 9-3 Remote office deployment pattern

A pair of clustered RTSS policy servers are set up at the remote office, and a WebSphere Application Server cluster provides system high availability and scalability. The RTSS server maintains a local cache of the policies that apply to the local services in the remote office. There could be some local user registries and databases that provide additional policy information; the RTSS server can use the policy information point (PIP) plug-in to retrieve this information. The RTSS server acts as the policy decision point (PDP) in this pattern.

For services and applications running in the remote office, an RTSS remote client is deployed to the WebSphere Application Server servers or clusters. The RTSS remote client acts as the policy enforcement point (PEP) component, it forwards the application's authorization queries to the RTSS server through the XACML over SOAP protocol, and then presents the authorization decision back to the application. Because the RTSS remote client does not maintain a local cache of the policies, the RTSS server must be highly available and able to handle the peak volume of the queries. If needed, additional RTSS servers (either standalone or clustered) can be set up to distribute the load; each server or server cluster handles a certain number of applications or services. The RTSS architecture is highly scalable to handle increased load.

9.3.3 Other considerations in determining deployment patterns

In the previous section, we used a network topology to design different deployment patterns. In reality, a network topology is just one of the many factors that can influence a Tivoli Security Policy Manager deployment pattern.

The total number of protected WebSphere Application Servers and clusters may determine whether we need additional runtime security services servers to share the policy distribution loads, and an runtime security services server is also needed if there are PEPs that need to perform authorization queries through XACML over SOAP.

Use runtime security services local clients for regular deployment. If there are a large numbers of WebSphere Application Servers and clusters to be protected, and if we observe performance degradation on the policy server, we can either add more policy server instances in the WebSphere Application Server cluster, or start to add runtime security services servers to share the load.

In an environment where both a policy server and runtime security services server are present, both runtime security services local clients and remote clients can be used. For systems with limited resources and a number of applications and services, it is a good idea to use an runtime security services remote client. For systems that have limited network access to the policy server, the runtime security services remote client might be the only choice.

In summary, an runtime security services server can be used to share the load of policy distribution, support runtime security services remote clients in a network environment with limited access, and support PEPs that do not have an runtime security services local client. The runtime security services local client is the preferred way for J2EE applications and web services, and it is more efficient, independent, and simple to manage. The runtime security services remote client works better for systems with limited resources or limited network access.

9.3.4 Operational considerations

Let us take a look at some operational considerations.

Performance

The key to runtime performance is to ensure that the *local policy cache* for policy distribution targets (PDT) is enabled. This action can reduce the level of processing that is required for subsequent requests. Depending on the type of application and the number of servers, it may be appropriate to use the local runtime security services instances, avoiding the requirement for a remote runtime security services client and server.

In environments that experience a high volume of transactions or peak load capacity transactions, clustering and load balancing of remote runtime security services instances may contribute to performance gains and reduce load on the application servers themselves. A single and centrally managed runtime security services cluster may also be more efficient and may require less ongoing management than a number of individually embedded local instances. Of course, this scenario relies upon your system architecture and available system resources, and in some respects depends upon the actual business deployment strategy, such as human resources, branch offices, and so on, of the organization itself.

As we mentioned earlier, the unit of the policies that are distributed to the PDTs is the service or application container. For each service or application container, there is one XACML file that contains all policies. It is important to keep this XACML file at a reasonable size so that it can be transferred and parsed more efficiently. Refer to the discussion in 9.4.2, “Application policy design considerations” on page 283 for more information about how to control the size of the policy file.

In 9.2.2, “Identifying policies, services, and data” on page 272, we briefly discussed how important it is to understand your applications to best determine the integration pattern with Tivoli Security Policy Manager. When considering the performance of an application, rather than the performance of the overall system architecture, the design of how the integration will be achieved is important.

Writing applications and directly calling an API each time an authorization decision is required may not be as efficient as using entitlements or potentially a different form of integration, such as a container or interceptor. Service design and policy structure can assist you in determining the best programmatic implementation to use Tivoli Security Policy Manager security.

When considering Java based applications, for example, a method called *interceptor model* from the Spring framework may yield improved performance over calling externally for all authorization decisions, inline in the application code. Especially for large codebases made up of multiple classes and methods, it is more appropriate to protect the resources being accessed, rather than the code itself, so that not every class and method requires a specific policy to be authored and attached.

Performance is always a key issue in the deployment of new systems, and ideally the new systems should outperform the previous one, while still adding additional capabilities to the environment. When deploying Tivoli Security Policy Manager, there are a number of optimizations and design choices that can all contribute to overall performance efficiencies.

Policy administration

Next, let us talk about some challenges to bulk load Tivoli Security Policy Manager policies, or to migrate the policies from one environment to another environment. The default Tivoli Security Policy Manager management interface console, Tivoli Integrated Portal (TIP), is GUI based, which is not an ideal platform to manage a significant number of policies, as the operations cannot be captured and repeated in an automated fashion.

The Tivoli Security Policy Manager V7.1 administration APIs can be used to assist the policy administration tasks. You can develop your own command-line applications to perform specific administration tasks, and then use scripts to call these commands. When using scripts, you may need to define the format of input files for policy definitions, and these input files and scripts can be modified and reused in different environments (for example, from a QA environment to a pre-production environment).

9.4 Application integration considerations

Tivoli Security Policy Manager provides multiple integration patterns for different applications and services. We discuss some of these integration considerations in this section.

9.4.1 Integration patterns

For the services or applications to be protected by Tivoli Security Policy Manager, there might be multiple integration patterns. Depending on the organization’s IT environment and business requirements, one or more integration patterns may be applicable to your deployment.

There are six service types available in the Tivoli Security Policy Manager console to represent your services and applications:

- Web services
- J2EE applications
- Portal applications
- Databases
- Microsoft SharePoint
- Applications

Although Tivoli Security Policy Manager differentiates between these different service types, from an integration perspective, these types can be more broadly categorized based on the method used to achieve integration. These patterns include:

- Web services using WS-Security and WS-Trust
- Java based applications including J2EE and Portal
- Structured Query Language (SQL) for Databases
- XML over SOAP by calling runtime security services

For a web services based pattern, Tivoli Security Policy Manager can provide authorization services using the WebSphere DataPower integration or using the WS-Trust interface. In addition, Tivoli Security Policy Manager can provide integration with Web Services Registry and Repositories (WSRR) for web services message protection policies. For an IT environment that uses WebSphere DataPower as a web services gateway, configure WebSphere DataPower as a Tivoli Security Policy Manager PDT to provide better performance and simpler configuration. However, if WebSphere DataPower is not used, and the web service is built on top of WebSphere Application Server, you may consider using the WS-Trust interface to make authorization service calls to Tivoli Security Policy Manager through the trust chain on the Trust Server. This configuration can be more complicated and requires additional software components, but it works well in some IT environments.

If the organization uses Java applications, there are also several Tivoli Security Policy Manager Java based integration patterns form which to choose. For applications using WebSphere Application Server J2EE container security services, Tivoli Security Policy Manager is shipped with a Tivoli Security Policy Manager JACC container that can be integrated with a custom J2EE application requiring configuration changes only. For applications that do not use J2EE container security, the Tivoli Security Policy Manager JACC Plus API can be used to make authorization calls to Tivoli Security Policy Manager. This API applies not only to J2EE applications, but also generic Java applications, such as the Plain Old Java Objects (POJO), or Java applications that use other containers, such as Spring.

In the database integration pattern, if the data is well structured within a supported database management system (such as DB2, Oracle, and so on), you can use Tivoli Security Policy Manager to provision the database access control policies directly to the database system using SQL. As an alternative, if the database management system supports it, another pattern is to set the access control over the particular database API or SPI calls to control access to the data.

For applications that cannot be integrated using the previous methods, or if you have multiple applications or services with similar service, role, and rule requirements, you can use XACML over SOAP to call runtime security services that provide a Tivoli Security Policy Manager integration point. Although the Tivoli Security Policy Manager console provides a discovery capability for Microsoft SharePoint sites to create the logic site hierarchy, integration is achieved by a Microsoft .NET runtime security services client. The remote client calls the web service interfaces of runtime security services to enable fine-grained authorization in Microsoft .NET environments. For all other applications that make use of the XACML over SOAP (remote client) pattern, you have the flexibility to create the service hierarchy in whatever format is most suitable for our system landscape.

Possible service hierarchies for custom applications could include:

- Hierarchical system environment roles, for example, a Microsoft .NET Role Provider
- Logical business functions, such as order taking, order processing, and dispatching
- User interface components within an application used to evaluate which controls should be visible or enabled, such as view, edit, or delete buttons
- A low level method of functions within the source code of an application requiring defined permission levels to invoke the call

When selecting the Applications service type as your integration pattern, you have the flexibility to model your requirements directly between your applications and the resultant Tivoli Security Policy Manager services. Appropriate planning, modelling, and simulation must be completed to ensure none of the other integration patterns are better suited for your environment. Due to the flexible nature of this pattern, it is easy to build your services in Tivoli Security Policy Manager around your existing applications, which creates a one to one mapping between application and authorization logic. This setup reduces the ability to optimize and centralize your service, role, and rule definitions, and requires application developers to understand the Tivoli Security Policy Manager deployment rather than the abstract business security model.

The application integration pattern is only limited by your organization's IT environments and can be used to complement complex business requirements and usage patterns. When it comes to choosing the most appropriate integration patterns, you should consider the following approaches:

- Avoid any overly complicated configurations, as these may become complex and expensive to deploy and maintain.
- Use existing functions from the product rather than developing your own code where possible. This action can reduce service and support costs during the application's lifetime.
- Investigate your integration options before implementing the final solution. Where possible, use prototypes and proof of concepts that allow you to validate the integration pattern prior to deployment.

9.4.2 Application policy design considerations

During the policy modelling and simulation phase of the policy lifecycle model, we must consider the system architecture and business requirements to assists us in selecting the most appropriate integration pattern. In this section, we focus on the customizable Applications service type available in Tivoli Security Policy Manager, which uses the XACML over SOAP (remote client) pattern. The Applications service type should be used where additional flexibility is required over the traditional service types available in Tivoli Security Policy Manager.

When you use a custom Applications service type, you have the flexibility to define the application or service hierarchy to represent the logical business and IT requirements. For web services, J2EE applications, Portal applications, Microsoft SharePoint sites, and databases, the nodes in the protected services are determined by the way they are defined in the corresponding service definitions and are imported to Tivoli Security Policy Manager rather than created manually.

Tivoli Security Policy Manager organizes application definitions in a tree, as shown in Figure 9-4. The top level *Application Container* represents the parent application level for all the protected *Elements* and the *Child Elements* below them.

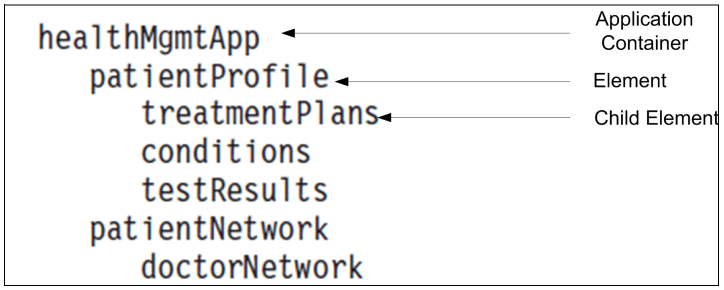


Figure 9-4 Application services tree structure

During the creation of your Applications service type, the Application Name you specify is visible in the services list in the Tivoli Security Policy Manager console. In Figure 9-4 `healthMgmtApp` represents the name and possibly also the Application ID. This value is used as the Context ID for the service to index into the policy store. Each of the Elements, such as `patientProfile`, and Child Elements, such as `treatmentPlans`, must also have a name and ID supplied so they may be referenced in authorization decisions on the client.

The granularity required for authorization in your application or the way in which you model your requirements determines the hierarchical Element structure defined within an Application Container. Although there are no restrictions on the number of elements that can be created in the one container, careful planning can improve the manageability and potential performance of authorization policy decisions for the container.

Before defining an Applications service type, several general considerations should be made to assist in creating an optimized and flexible service structure.

Service abstraction

When creating your custom service, if you have not considered the other integration patterns available or planned your service requirements appropriately, it is easy to build your service with a tight coupling tieing the implementations of the real application and your service structure together. In doing so, you reduce the benefits of centralized policy management and the ability to reuse your role definitions and business rules across your integrated systems and applications.

The Elements you define in your service structure represent the modelling of the real application, role, or process in your organization. We can use native names of the resources, such as Java class, methods, and so on, we can use metadata associated to the application, such as logical system components (each requiring different levels of authorization), or even a model of the roles for a preconfigured system, such as a Microsoft .NET Role Provider. In each case, it is important to maintain a level of abstraction between the real IT assets and the custom service.

For example, consider a custom web application deployed in WebSphere Portal that customizes the user interface elements based on a authorization level or entitlements of a user. Defining all of the user interface elements, such as buttons, check boxes, menus and so on, may work well for a single application deployment, but tightly couples the GUI design with service security policy design. If there is more than one web application, each one needs a corresponding service created in the Tivoli Security Policy Manager console, reducing scalability and increasing management.

One solution allowing the web application design to be maintained and abstract the service from the design is to use JSP tags to classify the GUI elements. For this solution, the security policy is based on JSP tags when the Application Developer assigns a unique JSP tag to each GUI element required. In the corresponding service and security policy, the *Security Administrator* need only be aware of the function of each JSP tag label and not the layout or design of the web application itself. With a common understanding and agreement of the JSP tags, both parties can continue to build and maintain their components without needing to understand the implementation details of the other.

With careful planning and modelling and an agreement about the abstraction details, you can create your service in the same way in which the TSPM service types abstract the design and layout from the actual IT implementation.

Service and authorization policy management

One of the significant features of Tivoli Security Policy Manager is the abstraction of business rules and authorization security policy from real IT assets and processes. To simplify security policy management and facilitate reuse across all services, it is important to understand and optimize all of your applications, roles, business rules, and so on. This can be achieved by following the policy modelling and simulation phase of the *Policy Lifecycle Model*, which minimizes the implementation specific security policies.

When undertaking policy modelling and simulation, one of the desired outcomes is to identify synergies between the various services and applications used within the organization. If a custom Applications service type is required, it is important to consider the breakdown of the service hierarchy required. For example, consider two different applications with similar security models requirements. In this scenario, much of the service hierarchy would be the same for both applications, and therefore a good candidate to reuse.

If one application had an extremely complex model and required an extensive hierarchy, it may be necessary to create a different Application Container for each application. Similarly, if all of the levels of hierarchy are added for all of the applications within the organization and no optimization has been performed, this may result in hundreds of Elements being created. Maintaining a large number of objects can be difficult and error prone simply due to the need for scrolling through the service to find the requirement element and attach security policies.

If your custom Applications service type contains a significant number of Elements and also Child Elements, the nesting effect of authorization policy inheritance can add unnecessary complexities during policy authoring. Because authorization policies are inherited by all children, you may need to author and attach a specific policy at individual nodes to compensate for the parental behavior. The maintenance and testing of complex and inherited policies can become expensive and error prone.

If policy modelling and simulation is not completed thoroughly and optimizations are not made, the number of resource elements and potential nested elements can increase the level of effort required for effective policy management.

Application Role reuse

Although Application Roles are defined for the whole Tivoli Security Policy Manager policy space, the mapping of an application role to user group is unique for each application container and needs to be mapped for each custom Applications service type that you create. Maintaining a large number of Application Containers requires them to be individually mapped and managed. If multiple applications share many of the same Application Role mappings, aggregating the Elements from each application into one Application Container makes it easier to reuse and manage the role mapping.

However, if a single application has not been optimized or has a significant number of Elements defined, to improve manageability and performance, we may need to consider maintaining multiple application containers. If there is any change in the Application Role to user group mapping in the organization, we must ensure that the change is propagated to all related Application Containers.

Although this change increases the management requirements by having to maintain more than one Application Container, the complexity is limited to just the one application and allows the benefits of the Element aggregation for the other applications to be maintained.

Service classification

Tivoli Security Policy Manager classifications are a collection of policies and service types that can be used to simplify the policy administration. Multiple classifications can be assigned to a single service type, and the union of the policies defined in those classifications will be applied. Classifications can help organize policies in a more efficient manner and can be used as building blocks for attaching complex policies.

Traditionally, policies are attached directly to a service or an Element in an Application Container, which requires that the security administrator understand the details of the service or application. Classifications remove this need by only requiring classifications with required policies to be created and then adding services into the appropriate classifications.

The policies contained within a classification are typically well established and represent defined actions or capabilities for of Application Roles. By using classifications, a security administrator does not need to know the details of the service when defining a classification. After classifications are defined, the service or application owner can determine which classifications should be applied.

Where possible, classification should be used to prevent the need for attaching individual security policies to Elements of each Application Container. This scenario assists in the reduction of the complexity of policy inheritance and consistent policy management across protected services.

Authorization policy distribution, storage, and enforcement

Tivoli Security Policy Manager distributes effective policy to PDTs, such as WebSphere DataPower, runtime security services, and so on. If you maintain a number of Application Containers, each with similar Elements and Application Roles and Rule Parameters, there is a significant amount of duplication in the traffic, although the network traffic generated by the distribution is small.

For example, consider an organization that is maintaining hundreds of Application Containers and a change in mapping is required for one of the configured applications. Each of the Application Container role mappings must be updated, configured, and then distributed, which results in additional security policy management and network traffic impact to distribute a similar policy to the configured PDTs. If *classifications* are used and similar Elements from different applications are aggregated into only a few Application Containers, the management and network traffic requirements are significantly reduced.

Using the example scenario above, if your Applications service type has not been optimized to reduce the number of Elements and the number of custom services required, the physical size of the policy and previous distributed policies increases. The storage requirements for each PDP continues to increase as additional policies are updated and distributed.

Conversely, some PDPs may only be responsible for a subset of the application elements, which means it is undesirable to distribute and store the additional policy elements. This situation can only be overcome by maintaining multiple Application Containers with different PDT and PDP configurations.

During policy enforcement, performance may be degraded if the policy store is extremely large and contains nested and complex policies. Instead, you should have optimized services and an authorization policy to reduce the overall footprint of the policy management, configuration, distribution, storage, and enforcement.

9.4.3 Conclusion

Tivoli Security Policy Manager provides a number of integration patterns that are suitable for a range of different applications and services. Each pattern is suited to particular integration scenarios, with the XACML over SOAP (remote client) pattern providing the most flexibility. With this pattern, you have the capability to create the service hierarchy in whatever format is most suitable for your system landscape, but a number of considerations must be explored to ensure optimization and performance is achieved.

9.5 Conclusion

An enterprise deployment of any new system always requires effective planning prior to any implementation. Defining and implementing a repeatable pattern for deployment not only assists in the migration of applications one by one, but also during migration from a test to production environment.

When deploying Tivoli Security Policy Manager, a number of significant planning steps should be completed and all levels of the organization undergoing the change should be included. By involving all functional departments of the business, a greater sense of ownership by all is realized and there will likely be fewer challenges, such as unknown applications and systems or missed rollouts to users.

There are a number of deployment patterns that can be applied to a range of organizational structures, each with specific implementation details, so it may be necessary to choose a particular pattern, but also use deployment options from another pattern.

By planning for your deployment thoroughly and gaining a clearer understanding of your organizational services, resources, roles, business rules, infrastructure, and operational requirements, you can experience a smoother implementation and accelerated time to value for the solution.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 292. Note that some of the documents referenced here may be available in softcopy only.

- *IBM Tivoli Security Policy Manager*, REDP-4483
- *Understanding SOA Security Design and Implementation*, SG24-7310
- *WebSphere Application Server V7.0: Technical Overview*, REDP-4482

Other publications

These publications are also relevant as further information sources:

- *IBM Tivoli Security Policy Manager Version 7.1 Administration Guide*, SC23-9476
- *IBM Tivoli Security Policy Manager Version 7.1 Configuration Guide*, GC27-2713

Online resources

These websites are also relevant as further information sources:

- Additional information about Tivoli Security Policy Manager:

  https://www.ibm.com/developerworks/mydeveloperworks/wikis/home?lang=en#/wiki/Tivoli%20Security%20Policy%20Manager/page/Home

- The IBM Tivoli Security Policy Manager Information Center contains information describing the installation and use of IBM Tivoli Security Policy Manager:

  http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=%2Fcom.ibm.tspm.doc_7.1%2Fwelcome.html
- The IBM WebSphere Application Server Information Center contains information describing the installation and use of IBM WebSphere Application Server:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.home.doc/welcome.html

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM

Redbooks

IT Security Policy Management Usage Patterns Using IBM Tivoli Security Policy Manager

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

IT Security Policy Management Usage Patterns Using IBM Tivoli Security Policy Manager

End-to-end security policy management for IT infrastructures

IBM Tivoli Security Policy Manager architecture overview

Solution patterns and deployment considerations

In a growing number of organizations, policies are the key mechanism by which the capabilities and requirements of services are expressed and made available to other entities. The goals established and driven by the business need to be consistently implemented, managed and enforced by the service-oriented infrastructure; expressing these goals as policy and effectively managing this policy is fundamental to the success of any IT and application transformation.

First, a flexible policy management framework must be in place to achieve alignment with business goals and consistent security implementation. Second, common re-usable security services are foundational building blocks for SOA environments, providing the ability to secure data and applications. Consistent IT Security Services that can be used by different components of an SOA run time are required. Point solutions are not scalable, and cannot capture and express enterprise-wide policy to ensure consistency and compliance.

In this IBM Redbooks publication, we discuss an IBM Security policy management solution, which is composed of both policy management and enforcement using IT security services. We discuss how this standards-based unified policy management and enforcement solution can address authentication, identity propagation, and authorization requirements, and thereby help organizations demonstrate compliance, secure their services, and minimize the risk of data loss.

This book is a valuable resource for security officers, consultants, and architects who want to understand and implement a centralized security policy management and entitlement solution.

SG24-7880-00 ISBN 0738436143