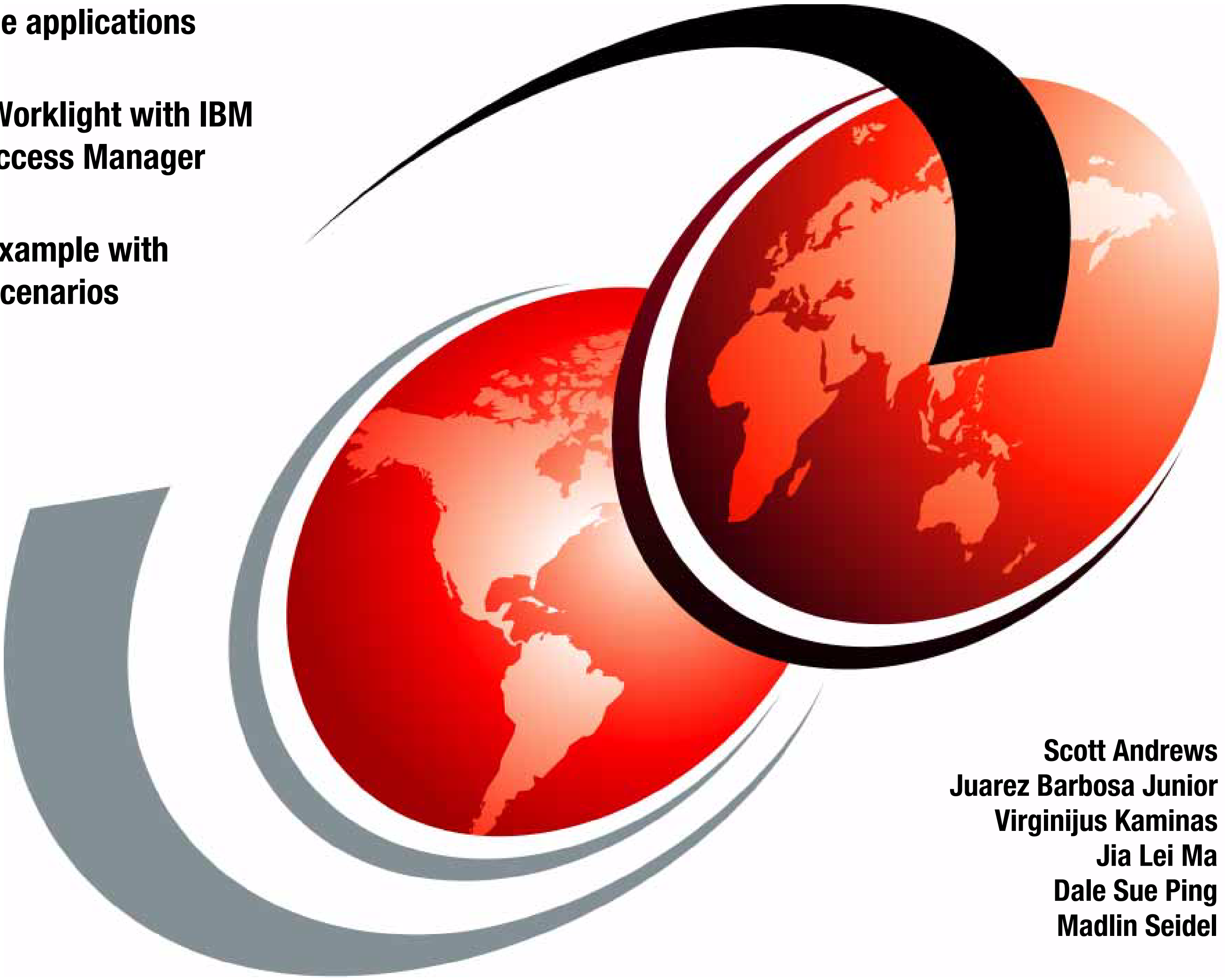
IBM

Securing Your Mobile Business with IBM Worklight

Apply Worklight security features to your mobile applications

Integrate Worklight with IBM Security Access Manager

Learn by example with practical scenarios



Scott Andrews
Juarez Barbosa Junior
Virginijus Kaminas
Jia Lei Ma
Dale Sue Ping
Madlin Seidel

Redbooks

ibm.com/redbooks

International Technical Support Organization

Securing Your Mobile Business with IBM Worklight

October 2013

SG24-8179-00

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

First Edition (October 2013)

This edition applies to Version 6 of IBM Worklight.

Contents

Notices

Trademarks

Preface

The IBM® Worklight® mobile application platform helps you to develop, deploy, host, and manage mobile enterprise applications. It also enables companies to integrate security into their overall mobile application lifecycle.

This IBM Redbooks® publication describes the security capabilities offered by Worklight to address mobile application security objectives.

The book begins with an overview of IBM MobileFirst and its security offerings. The book also describes a business scenario illustrating where security is needed in mobile solutions, and how Worklight can help you achieve it.

This publication then provides specific, hands-on guidance about how to integrate Worklight with enterprise security. It also provides step-by-step guidance to implementing mobile security features, including *direct update*, *remote disable*, and *encrypted offline cache*. Integration between Worklight and other IBM security technologies is also covered, including integration with IBM Security Access Manager and IBM WebSphere® DataPower®.

This Redbooks publication is of interest to anyone looking to better understand mobile security, and to learn how to enhance mobile security with Worklight.

Authors

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

Scott Andrews is an Advisory Software Engineer in the Australia Development Lab in Gold Coast, Australia. He is the Team Lead for the IBM Security Access Manager Integration Factory, specializing in IBM Security Access Manager, IBM Tivoli® Federated Identity Manager, and IBM Tivoli Security Policy Manager.

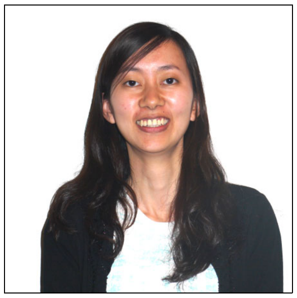
Scott has designed and delivered security integrations for many third-party products, including Worklight, JBoss, and Microsoft SharePoint. He has assisted clients with the architecture and deployment of these security integrations. He joined IBM in 2002 after graduating from Bond University, Australia with a Bachelor of Information Technology degree, and remains active today as an IBM University Ambassador.

Juarez Barbosa Junior is a Mobile Application Developer and Architect currently playing a role as IT Specialist for the worldwide Development Support Team with IBM Global Business Services®. He is a technology enthusiast who is passionate about Java and mobile computing, and he started working with mobile back in 2003.

He has 19 years of experience in IT, with 15 years using Java platforms and 10 years working with mobile projects including application development, software architecture, and systems integration. He is a former Nokia Developer Champion and an IBM Redbooks Thought Leader for IBM Mobile.

Virginijus Kaminas is a Software Engineer based in Ireland, and works for the Mobile Center of Competency for Industry Solutions. He joined IBM in 2012, was part of the Extreme Blue® program, and helped to develop Android and iOS applications for browsing IBM WebSphere Portal content.

His focus is on mobile solutions using Worklight, and he has contributed numerous articles regarding leading practices associated with mobile development. He holds a BSc degree in Computer Applications from Dublin City University, and is an IBM Redbooks Thought Leader.



Jia Lei Ma is an Information Developer at the Shanghai site of the China Development Lab. She joined IBM in 2005, and has worked as an element planner and quality assurance (QA) lead on various writing projects. These projects cover topics including IBM z/OS®, IBM i5/OS™, and IBM Rational® compilers.

She holds a BA degree in English from Fudan University, and is a certified Project Management Professional.

Dale Sue Ping is a WebSphere Technical Sales professional in Canada. He has 35 years of experience in the IT industry (technical and managerial), mostly with z/OS systems. For the past 12 years, he has specialized in the WebSphere product portfolio, such as Worklight and IBM WebSphere Application Server.

He works with technologies in the areas of connectivity, including service-oriented architecture (SOA) and enterprise service bus (ESB), in addition to business process management. He holds a BSc degree in Computer Science from the University of Toronto.

Madlin Seidel is an IBM employee with the corporate alternative work arrangement in Germany. She joined IBM in 2011, and has gained insight into various IBM divisions since then.

Madlin is currently studying International Business Administration in the corporate integrated degree program. She is going to graduate in 2014, and will hold a bachelor's degree from the Berlin School of Economics and Law.

This project was managed by the following IBM Redbooks team members:

- Martin Keen

  IBM Redbooks Project Leader
- Elise Hines

  IBM Redbooks Technical Writer

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

- Miku Jha

  IBM Product Manager for IBM Mobile
- Raj Balasubramanian

  IBM Product Design and Development, IBM MobileFirst
- Tom Mulvehill

  IBM Security Product Management
- Patrick Wardrop

  IBM Product Architect, IBM Security Access Manager
- Shane Weeden

  IBM Product Architect, IBM Security Systems
- Keisha McKenzie

  IBM Software Group Worldwide Market Segment Manager
- Tracy Clark

  IBM Software Group Worldwide Manager of Content and Messaging, Mobile Enterprise Marketing
- Michael Ortman

  IBM Software Engineer, Worklight Research and Development
- Ozair Sheikh

  Senior Managing Consultant, Software Services for WebSphere
- Shiu Fun Poon

  DataPower Security Architect, WebSphere DataPower Appliance Development

Now you can become a published author, too!

Here’s an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies.

Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Obtain more information about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- 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

1

Overview of IBM MobileFirst and its security offerings

Innovative enterprises gain opportunities by bringing all of their resources together to strengthen client engagement, whenever and wherever the client wants. IBM MobileFirst offers a complete solution for mobile products and services. Enterprises can use it to achieve efficiency and gain a competitive advantage.

This chapter introduces the IBM MobileFirst portfolio, including its application development platform, which integrates management, security, governance, and analytics capabilities.

Due to the increasing importance of security and risk mitigation in mobile enterprises, this chapter also describes common security challenges, and provides an overview of IBM Worklight and enterprise security products, including IBM Security Access Manager and IBM WebSphere DataPower.

1.1 Business value of mobile technologies

As technology advances, usage patterns change. In addition, new business tools, such as smartphones and tablets, emerge. BlackBerry was the de facto standard mobile device for enterprises for many years. However, iOS and Android smartphones have superseded the BlackBerry market share. *Mobile applications (apps)* are no longer side effects of the social media empire, but have become an essential tool for business productivity.

Every day, countless confidential transactions with financial institutions, online merchants, airlines, and various other retailers are performed on mobile devices. These transactional trends have been analyzed, and help to formulate innovative mobile-based advertising strategies, such as location-based advertising campaigns.

These strategies have paid off: 75% of mobile shoppers take action after receiving location-based messages[1]. There is increasing use of mobile devices as the primary means of interaction and communication with employers, customers, family, and friends. According to recent studies, 91% of mobile users keep their devices within reach 100 percent of the time[2].

Therefore, providing a highly available, innovative, and customer-focused mobile presence has become crucial to enterprise success and growth.

Take advantage of mobile technologies to achieve the following business benefits:

- Attract new customers, transform your value chain, and increase productivity.
- Engage users by delivering apps that are created with simplified multichannel development solutions.
- Turn interactions into an opportunity for return on engagement (ROE) and return on investment (ROI).
- Provide a consistent brand experience.
- Deliver innovation by using cloud technologies to meet increased IT demands, lower costs, and to gain insight, visibility, and control.

1.2 IBM MobileFirst solution overview

Whether your aim is to transform your customer acquisition strategies, to rationalize your business processes, or to boost your product and service innovations, IBM MobileFirst offers a broad mobile solution portfolio that fits your business needs. IBM MobileFirst enables you to encourage customer-building touch points, and deepen relationships with your customers, employees, and partners in real time.

IBM MobileFirst provides dedicated industry strategies, solutions, and services that can be designed, integrated, and delivered by IBM and its partners. These solutions are delivered using a common application and data platform with robust management, security, and analytics.

Furthermore, the IBM MobileFirst portfolio facilitates rapid integration between social media, cloud services, and back-end technologies that secure and manage strategic business processes. Figure 1-1 on page 3 depicts IBM MobileFirst's holistic approach to providing an end-to-end solution portfolio.

[1] IBM Mobile First, http://public.dhe.ibm.com/common/ssi/ecm/en/wss14156usen/WSS14156USEN.PDF
[2] http://public.dhe.ibm.com/common/ssi/ecm/en/wss14156usen/WSS14156USEN.PDF

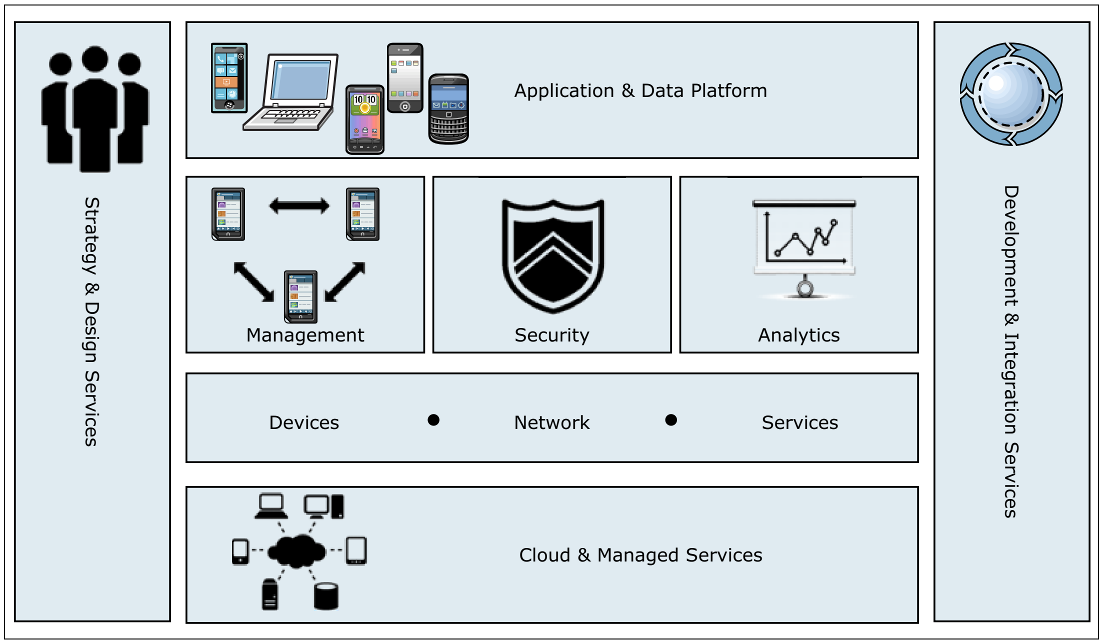

Application & Data Platform
Management
Security
Analytics
Devices
Network
Services
Cloud & Managed Services
Strategy & Design Services
Development & Integration Services

Figure 1-1 IBM MobileFirst overview

This section explores how the IBM MobileFirst offerings can contribute to your business success.

1.2.1 Mobile application development with IBM MobileFirst Platform

IBM MobileFirst Platform has key capabilities designed to build and deliver mobile solutions rapidly, with increased quality and lower cost, spanning multiple platforms.

IBM MobileFirst Platform provides the following highlights:

- Native, web, or hybrid application development
- Tools for building and testing high-quality applications for several devices
- Management, security, delivery, and distribution of applications
- Simplified connectivity to existing data and services
- On-premise or managed service delivery

The key offerings for developing and deploying mobile apps are IBM Worklight and IBM Rational Test Workbench.

IBM Worklight

IBM Worklight offers enterprises a secure platform and different tools to develop, test, and manage mobile apps (See 1.5.1, “IBM Worklight platform as the basis for mobile security” on page 14).

The following capabilities are new in IBM Worklight V6.0:

- Embedded functional testing. Worklight V6.0 replaces time-consuming manual test processes with the industry's first integrated mobile app testing capability.
- Embedded customer experience analytics. Developers can instrument their mobile apps with an analytics framework that collects data as the app is being used by the customer. The instrumentation framework can be easily extended with IBM Tealeaf® CX Mobile to analyze mobile apps for greater marketing insight.
- Geo-location toolkit. Geo-location data is one of the most powerful differentiators of mobile apps. The goal is to avoid exhausting resources, such as battery life and network bandwidth, by constantly polling the current location. The Worklight V6.0 toolkit enables developers to set policies for acquiring geo data, and to send it in batches only when it is needed to trigger business actions.

IBM Rational Test Workbench

IBM Rational Test Workbench is a comprehensive test automation solution that facilitates regression, performance, and scalability testing and integration technologies. IBM Rational Test Workbench enables your enterprise to build intelligent and interconnected applications, which can either be deployed in a traditional infrastructure or a cloud infrastructure.

IBM Rational Test Workbench provides enterprises with the following functionality:

- Significant reduction of test cycle times
- Test automation for all types of applications, using a physical device or a mobile emulator
- Simplification of test creation, with storyboard testing and code-free test authoring
- Rapid development of complex performance tests, with script-less, visual, and workload models
- Earlier end-to-end integration testing throughout hardware, software, and cloud-based dependencies
- Extensibility

For more information about IBM Rational Test Workbench, see the following website:

http://www-03.ibm.com/software/products/us/en/rtw/

1.2.2 Bring your own device with IBM MobileFirst Management

IBM MobileFirst Management deals with the challenges of mobile devices and applications, which have become increasingly valuable as enterprises continue to develop and implement *bring your own device (BYOD)* policies. For enterprises whose focus is on managing sensitive data, tracking and optimizing mobile expenses, and handling multi-platform complexities, IBM MobileFirst Management is a viable option.

IBM MobileFirst Management offers the following functionality:

- Unified management across devices
- Selective destruction of corporate data
- Configuration and enforcement of password policies, encryption, virtual private network (VPN) access, and camera use
- An integrated enterprise app store to streamline workflow between development and operations
- Detailed usage analysis for optimization of telecommunication expenses
- A user portal for management of mobile equipment, carrier plans, and usage tracking

As the key offering in this domain, IBM Endpoint Manager for Mobile Devices provides a single platform with complete integration. You can use this platform to manage, secure, and report on notebooks, desktops, servers, smartphones, tablets, and even point-of-sale terminals. The unprecedented benefit to the enterprise is visibility and control over all devices, cost reduction, productivity increases, and compliance improvements.

For more information about IBM Endpoint Manager for Mobile Devices, see the following website:

`http://www-03.ibm.com/software/products/us/en/ibmendpmanaformobidevi/`

1.2.3 Device and data protection with IBM MobileFirst Security

As day-to-day business becomes increasingly global, the topic of mobile security gains importance. Due to the existence of many threats and opportunities for hacking, companies must increasingly focus on security and prevention. IBM MobileFirst Security serves companies that aim to protect devices and data, ensuring secure network access and mobile apps.

IBM MobileFirst Security consists of the following products, which are discussed in more detail in the referenced sections:

- IBM Security Access Manager

  See 1.5.2, “User protection with IBM Security Access Manager” on page 16.
- IBM Security AppScan®

  See 1.5.3, “Application security testing with IBM Security AppScan” on page 16.
- IBM WebSphere DataPower

  See 1.5.4, “Delivery of services and applications with IBM WebSphere DataPower” on page 17.
- IBM Security QRadar® security information and event management (SIEM)

  See 1.5.5, “Security intelligence with IBM Security QRadar SIEM” on page 17.
- IBM Mobile Connect

  See 1.5.6, “VPN software with IBM Mobile Connect” on page 17.

This Redbooks publication also shows individual use cases with IBM security offerings in the following chapters:

- Chapter 6, “Integration with IBM Security Access Manager” on page 101
- Chapter 7, “Integration with IBM WebSphere DataPower” on page 139

1.2.4 Optimization of mobile experiences with IBM MobileFirst Analytics

The goal of IBM MobileFirst Analytics is to provide a qualitative experience for your clients. It analyzes mobile behaviors and quantifies the business impact of usage patterns. IBM MobileFirst Analytics can also improve customer service and satisfaction, which consequently drives customer loyalty.

IBM MobileFirst Analytics offers the following functionality:

- Automatic detection of client issues through user and mobile device data
- User ability to drill down with high fidelity replay and reporting of the user experience
- Correlation of client behavior through network and application data
- High conversion and retention rates with quantifiable business impact analysis

In the domain of analytics, the key offering is IBM Tealeaf CX Mobile. Organizations can use it to achieve the following benefits:

- Optimize customer experiences for all users with mobile web, HTML5 sites, hybrid apps, and native applications.
- Gain complete mobile visibility by capturing user information for mobile websites, including network and client interactions, and touchscreen gestures, such as pinching, zooming, scrolling, and device rotation.
- Build and manage early warning systems to detect mobile user problems. Provide proactive awareness about mobile app failures, usability issues, or other obstacles that lead to failed transactions, abandonment, poor application store rankings, and negative feedback.
- Quantify revenue impact and segmentation with real-time, drag-and-drop analysis of specific mobile user behaviors or device attributes.
- Quickly find and isolate problems within mobile customer sessions, with powerful ad hoc discovery and segmentation without predefining tags or beacons.

For more information, see the following website:

http://www-03.ibm.com/software/products/us/en/cx-mobile

1.2.5 Putting it all together

IBM MobileFirst offerings follow the eight stages of the mobile development lifecycle (Figure 1-2 on page 8):

1. Design and develop

   In the first stage of the mobile development lifecycle, the mobile architecture has to be designed and developed. This can be performed using IBM Worklight. Making use of open, flexible, and standardized cross-platform (iOS, Android, Windows Phone, and BlackBerry) development methodologies, Worklight provides a rich development environment maximizing code reuse.

   IBM Worklight helps you to develop mobile apps, but also to deploy, host, and manage mobile enterprise apps. With IBM Worklight Studio, organizations can push developed mobile apps straight to the IBM Worklight Server and to IBM Worklight Application Center for fast distribution.

2. Instrument

   In the next step, customized applications can get instrumented in favor of analytics, security, and management control. IBM Tealeaf collects, analyzes, and streamlines application usage, responsiveness, and device information, such as device or app IDs. This provides full client experience management (see 1.2.4, “Optimization of mobile experiences with IBM MobileFirst Analytics” on page 6).

3. Integrate

   Mobile cloud services can be integrated. Mobile cloud services are high-value middleware capabilities that are designed to be used by mobile apps. As an example, consumers increasingly purchase clothing on their mobile devices. A pass service enables companies to engage with those customers easily by sending them loyalty coupons, discounts, and so on. Worklight and IBM WebSphere Cast Iron® can process such an integration with back-end data, systems, and cloud services.

   For cloud services, a firewall controls inbound and outbound connections. To secure these connections, an enterprise’s internal IT infrastructure can be protected with a proxy or a security gateway, such as IBM Security Access Manager or WebSphere DataPower.

   Using a security gateway, such as IBM WebSphere DataPower, enables companies to control the parts of infrastructure that are shown to consumers. This functionality is especially relevant for push notifications (see 1.5.2, “User protection with IBM Security Access Manager” on page 16 and 1.5.4, “Delivery of services and applications with IBM WebSphere DataPower” on page 17).

   For more information about IBM Worklight, WebSphere Cast Iron, and WebSphere DataPower, see *Extending your business to mobile devices with IBM Worklight*, SG24-8117:

   http://www.redbooks.ibm.com/redpieces/abstracts/sg248117.html

4. Test

   To test the configured, enterprise-specific app, your company can use IBM Rational Test Workbench. It is a comprehensive test automation solution, which enables regression, performance, and scalability testing (see 1.2.1, “Mobile application development with IBM MobileFirst Platform” on page 3).

5. Scan and certify

   In addition to the customized app testing run by Rational Test Workbench, IBM Security App Scan is able to scan, evaluate, and certify an organization’s apps. It provides the user with enhanced vulnerability testing, scanning, and reporting for mobile web apps and web elements (such as JavaScript and HTML5), or for hybrid mobile apps (see 1.5.3, “Application security testing with IBM Security AppScan” on page 16).

6. Deploy

   The app is distributed to various channels, such as Worklight Application Center, devices managed by IBM Endpoint Manager for Mobile Devices, and external app stores. IBM Worklight Application Center distributes mobile apps inside of a company, and IBM Endpoint Manager for Mobile Devices addresses mobile device management.

7. Manage

   Using the application management functionality of the Worklight Application Console helps enterprises to manage mobile app releases based on pre-established rules, such as rules concerning all apps deployed, adapters, and push notification. The Worklight Application Console helps companies to manage authentication, unified push and Short Message Service (SMS) notifications, operational analytics, and app versions.

For more information about Worklight components, see 1.5.1, "IBM Worklight platform as the basis for mobile security" on page 14, and *Extending your business to mobile devices with IBM Worklight*, SG24-8117.

8. Obtain insight

   At the end of the cycle, insights can be derived from Tealeaf usage pattern analysis, which improves the effectiveness of your application. Consequently, this understanding will not just close the iterative mobile development circle, but can even influence the next iteration of the product design.

   Figure 1-2 shows the eight stages of the mobile development lifecycle.

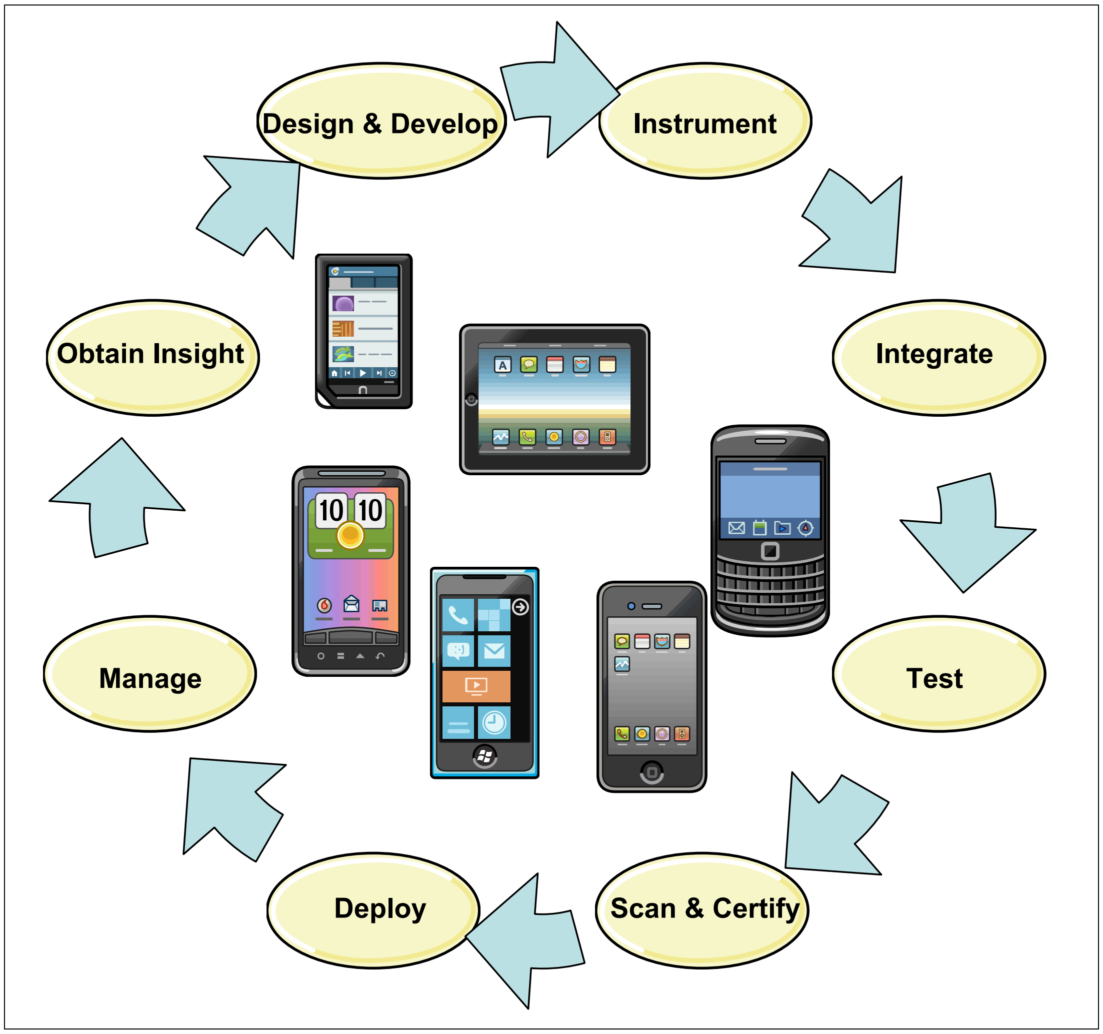


Figure 1-2 IBM mobile development lifecycle

So far, this chapter has shown that your organization can use IBM MobileFirst throughout the mobile app development lifecycle, making it a true end-to-end solution.

Because this book focuses on security issues, the next sections focus on the security aspects of IBM MobileFirst.

1.3 Mobile security threats

This section provides information about the various security threats that many enterprises face when operating a mobile business.

In a mobile world, there can often be little or no control over the device:

- When it is used
- Where it is used
- Who is using it
- What it is used for

The latest mobile devices are designed to offer abilities beyond voice and email, and to provide broad internet and network connectivity:

- 3G and 4G
- Long-term evolution (LTE)
- WiFi
- Bluetooth
- A wired connection to a personal computer

However, the increasing number of ways to transmit data also creates a number of potential ways that the device can be exploited.

When a device downloads a new mobile app from an online application store, there is the hazard that the software contains malware able to damage private and corporate data on the device. A device connected through WiFi or Bluetooth can be exploited and used in man-in-the-middle attacks. As shown in Figure 1-3, security threats can occur in multiple places along the varying paths of data transmission.

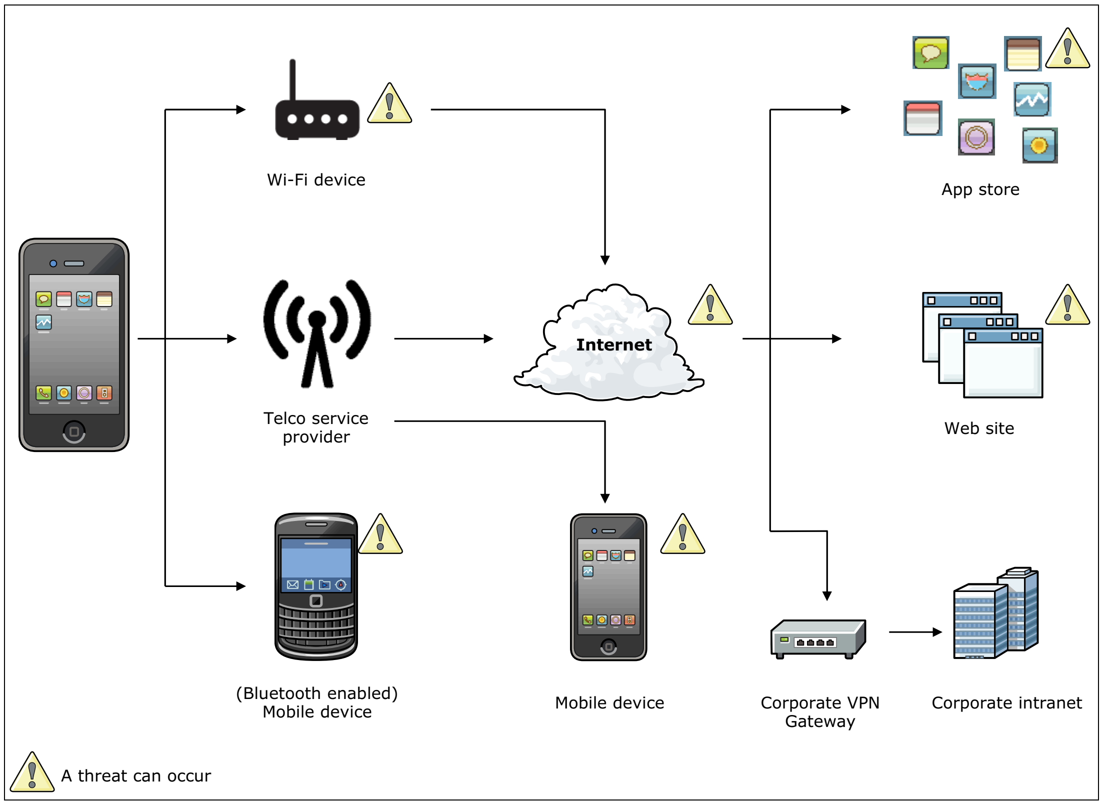


Figure 1-3 Flow of data transmission and potential locations of exploitation

The following hazards are among the most frequently seen mobile device security threats:

- Loss and theft
- Malware
- Spam
- Phishing
- Bluetooth and WiFi exploits

1.3.1 Loss and theft

Due to the small size and high portability of mobile devices, and the practice of BYOD encouraging the mix of private and business use, loss and theft are the most obvious security concerns for devices containing enterprise data or connecting to enterprise networks. According to a mobile threat study by Juniper Networks, 1 in 20 mobile devices has been stolen or lost in 2010[3], which reinforces the severity of this security threat.

So, what do you do if your device gets stolen? If the access to the device or the data is not effectively controlled, the data stored on the device can be compromised. The following techniques can reduce this risk:

- Using a complex password
- Locating the device remotely on a map using global positioning services (GPS)
- Locking the device to render it useless
- Wiping data remotely

1.3.2 Malware

Mobile device malware, such as *viruses*, *worms*, *trojans*, and *spyware*, has been on the rise, because most mobile platforms do not have native mechanisms to prevent or detect malware. Among others, malware can perform the following actions:

- Cause the loss of personal or confidential data.
- Incur additional service charges by sending premium SMS messages or initiating phone calls in the background.
- Render the device unusable.

No mobile platform available today is immune to malware. Primarily due to its recent popularity and open distribution model, Android is the leading platform exploited by new malware development. According to the research report from Juniper Networks, malware on Android has increased by 400 percent from June 2010 to January 2011[4].

Companies can significantly reduce the malware risk by adopting a mobile device security policy that is similar to the desktop and notebook environment. Furthermore, enterprises must run anti-malware software on each employee's device to detect malware in real time, and scan the entire device periodically.

1.3.3 Spam

With the growth of SMS, spam is also on the rise. Spam is a challenge for service providers, because it uses a significant amount of bandwidth, which reduces throughput for legitimate network traffic. It is also a growing security issue for mobile device users, because by clicking

[3] Securing mobile devices in the business environment,
`http://public.dhe.ibm.com/common/ssi/ecm/en/sew03027usen/SEW03027USEN.PDF`

[4] Securing mobile devices in the business environment,
`http://public.dhe.ibm.com/common/ssi/ecm/en/sew03027usen/SEW03027USEN.PDF`

a spam link, the user can be sent to a website hosting malware. According to the pilot of the Global System for Mobile communications Association (GSMA) Spam Reporting Services, 70 percent of reports of spam were for fraudulent financial services rather than the traditional advertising scenarios found in email[5].

The most effective way to deal with spam is to define blacklists to block spam messages, either by using the functions of an anti-spam solution, or by using an anti-spam feature on your device.

1.3.4 Phishing

Phishing attacks can include *email or SMS messages sent to trick users* into accessing a fake website, sending a text message, or making a phone call to reveal sensitive personal information. This information is often their Social Security Number, or credentials that enable the hacker to access financial or business accounts. Because of their small screen size, mobile devices often hide or collapse protection features, such as web address bars. Therefore, phishing attempts are likely to be more successful on mobile browsers.

An effective anti-phishing approach is to use two-factor authentication. First, a user enters a static password, then they enter a second authentication factor, such as a one-time password or a fingerprint, which is dynamically generated to further authenticate the user. Therefore, even if the user's static password is stolen, the hacker cannot log in to the genuine site.

1.3.5 Bluetooth and WiFi

Bluetooth and WiFi offer the convenience of increased connectivity for mobile devices within limited ranges, however *both can be easily exploited to infect a mobile device with malware*. For example, a mobile device can be lured to accept a Bluetooth connection request from a malicious device.

When mobile devices connect, the hacker is able to intercept and compromise all data sent to or from the connected devices. Therefore, setting the device to an undiscoverable mode and turning off the device's automatic Wi-Fi connection capability can reduce those risks. If you want to block incoming connection requests completely, install a local firewall.

1.4 Mobile application landscape

As a result of increased access and decreased control, security must be incorporated into the mobile app itself, the mobile app infrastructure, *and* the traditional network and server security infrastructure.

A mobile app strategy includes a complex landscape of technologies and devices, with no single perfect fit for every industry or enterprise. Companies must examine a number of key requirements to ensure the deployed applications, in addition to providing features and functions to make them useful, also offer secure access, targeted strategic platforms, and mobile app-type offerings to support the enterprise needs.

As such, security requirements and features must be incorporated directly into the entire mobile app solution, and not just on the server. Designing and building a robust, reusable security capability for the matrix of platforms and application types is a challenging and complex task. IBM Worklight provides a secure, reusable, and platform-neutral solution. Its

[5] Securing mobile devices in the business environment,
`http://public.dhe.ibm.com/common/ssi/ecm/en/sew03027usen/SEW03027USEN.PDF`

architectural composition is described in detail in Chapter 3, "IBM Worklight security overview" on page 27.

1.4.1 Mobile application platform

Security for mobile is not simply a virus scanner or firewall, and there is a range of mobile device platforms, such as *BlackBerry*, *Symbian*, *iOS*, *Android*, and *Windows Phone*, which *all need to be supported*. Each platform brings with it a unique security model, and other than the BlackBerry platform, most started as consumer platforms, and lack enterprise-strength security controls.

BlackBerry has been the de facto mobile device for business for many years, but the availability of other smartphones and tablets with broader consumer appeal, such as iOS and Android devices, is fundamentally changing the strategic platform for mobile app development.

Companies that provide mobile hardware to their employees are able to dictate the targeted platform to reduce the scope of application development and support. However, employees are now bringing their own mobile devices to the workplace, and asking companies to support them. These new devices offer improved hardware performance, a more robust platform feature set, and increased communication bandwidth, expanding their capabilities beyond voicemail and email.

In providing services and mobile apps to their customers, companies must consider the most popular platform usage pattern for their target demographic, which is not necessarily the one with the best integration or security features.

Companies can no longer rely on a single platform to deliver mobile apps. Worklight provides a rich development environment to develop mobile apps for a range of mobile operating systems using the Worklight security framework.

1.4.2 Mobile application types

Another key decision for mobile app development is what type of app to build. Traditionally, companies have provided corporate websites and pages designed to be viewed on a device with a large screen, keyboard, and mouse. With trends moving toward apps for on-the-go and mobile access, companies need to understand how their apps will be used, and what features and functions of the device are meaningful in the mobile app context.

The type of mobile app to deploy is often a topic of debate, and relies upon the type of functions that the app needs to perform. For example, LinkedIn was initially developed and deployed as a hybrid application, but has followed suit with Facebook, which moved to a native application approach.

The main mobile app types developed and distributed today are summarized in Table 1-1 on page 13.

Mobile web applications

A company providing a simple Contact Us page using a traditional web platform can opt for a mobile web app, *customized web pages accessed through a web browser on a mobile device*. There are no components deployed or installed on the device, and usability is limited to mostly viewing options, with restrictions on user input and resolution as a mobile device and not a traditional personal computer (PC).

These apps are optimized to display traditional web pages on mobile devices, and generally lack integration with device features, such as cameras with direct upload capability, GPS, near field communication (NFC), and other device capabilities.

They can be cost-effective to develop and deploy. However, they offer a limited user experience for starting the app and interacting with the device features.

Native applications

This type of mobile app is considered *a custom app written for a specific device and platform*. They provide full device capabilities and interaction, but require specialist app developers with skill sets for each intended hardware and software platform.

These apps are started from a list or home screen on the device itself, and are generally installed from trusted app stores or centers. Updating native apps requires a complete removal of the old version and installation of the new version.

Although this type of mobile app provides full control for the application, almost all of the app needs to be developed in-house, and typically is not portable or reusable across platforms. These apps represent a trade-off: full control and integration set against the cost to develop and maintain.

Hybrid applications

A popular intermediate approach is creating hybrid apps, which are mostly *developed using HTML5 and JavaScript with toolkits embedded in the app*. These toolkits can include Apache Cordova (PhoneGap), Dojo, JQuery, and so on.

This type of application provides a common platform to author apps for portability across platforms and devices, but does not include full device capabilities. Where a device integration feature is not provided by a toolkit, you need to write platform-specific native code.

Hybrid applications are started from a list on the home screen of the device, and are generally installed from trusted application stores or centers. Depending on the update required for a hybrid app, device-based HTML5, JavaScript, Cascading Style Sheets (CSS), and images can be updated in isolation. However, any native components require a removal of the old version and installation of the new version. Typically, most of the app capabilities use web technologies, making the update process quicker and more efficient.

The overall cost to develop can be lower, and the return on investment higher, than for native apps, because you do not require app developers for each hardware and software platform.

Table 1-1 provides a summary of the main mobile app types developed and distributed today.

Table 1-1 Mobile app type considerations

	Mobile web apps	**Native apps**	**Hybrid apps**
Development skills required, including languages and toolkits	Web technologies, such as HTML, JavaScript, and CSS	Device-specific and platform-specific languages, such as Java and Objective C	Web technologies, such as HTML and JavaScript
Device integration capability, such as camera, GPS, and NFC	Limited	► Complete ► Custom code	► Most ► Apache Cordova or Native
Code portability and reuse	Not applicable	Limited	High

	Mobile web apps	Native apps	Hybrid apps
Offline mode usage capability	No	Yes	Yes
Cost to implement and maintain	Low	High	Low

1.4.3 Putting it all together

No single mobile app platform and type will meet all business expectations for requirements and cost, and often a *combination* or *staged approach* is used. Each mobile app platform will appeal to different demographics or enterprises. Each mobile app type also has its own advantages and disadvantages, ranging from user experience, control, integration, and device capabilities.

Worklight can be used in all of these mobile app types to build apps for a range of platforms, which extends the security capabilities to the greatest number of mobile app use cases.

1.5 IBM MobileFirst Security solution outline

To understand how IBM Worklight helps organizations to integrate security into the overall mobile app lifecycle, this section provides details about the different components of IBM Worklight. Additionally, this section introduces the IBM security products that ensure security on mobile devices, over the network, in the enterprise, and within the mobile app itself.

1.5.1 IBM Worklight platform as the basis for mobile security

As shown in Figure 1-4 on page 15, the platform consists of five main components, providing a robust solution for mobile app security:

- IBM *Worklight Studio* is an Eclipse-based integrated development environment (IDE) that enables you to perform the coding and integration that is required to develop apps for various mobile operating systems.
- IBM *Worklight Application Center* is a web-based, internal enterprise application store for centralized application distribution, installation, and feedback. Using the application catalog service and application install service, internal users can download apps, track installed apps, and provide feedback by rating the app versions. During the development lifecycle, Worklight Application Center can also be used to streamline new app versions from development to test.
- IBM *Worklight device runtime components* consist of runtime customer application programming interfaces (APIs). These are essential libraries complementing the server by exposing APIs to access server functionality, implement customer-side portions of security features, use JavaScript and HTML for cross-platform development, and facilitate the interaction between, for example, JavaScript and native code.
- IBM *Worklight server* is a Java-based server that works as a security-rich and scalable gateway between applications, external services, and the enterprise back-end infrastructure. This server facilitates secure connectivity, multi-source data extraction and manipulation, authentication, direct updates of web and hybrid applications, analytics, and operational management functions.

- IBM *Worklight console* is a web-based administrative console supporting the ongoing monitoring and administration of the Worklight Server and its deployed apps, adapters, and push notifications. Additionally, based on configurable preset rules of the app version and device type, it helps to control and manage the access of apps to the enterprise network.

In a typical Worklight platform setup, the Worklight Server is installed behind a firewall or web reverse proxy, such as IBM Security Access Manager or WebSphere DataPower, and the mobile apps are installed and run on mobile devices, which at least partly exist outside the enterprise network. The Worklight Server acts as a gateway, mediating the communication and access of mobile apps to back-end systems. It can be further secured using IBM Security Access Manager or WebSphere DataPower.

Figure 1-4 shows the five main components of the IBM Worklight platform.

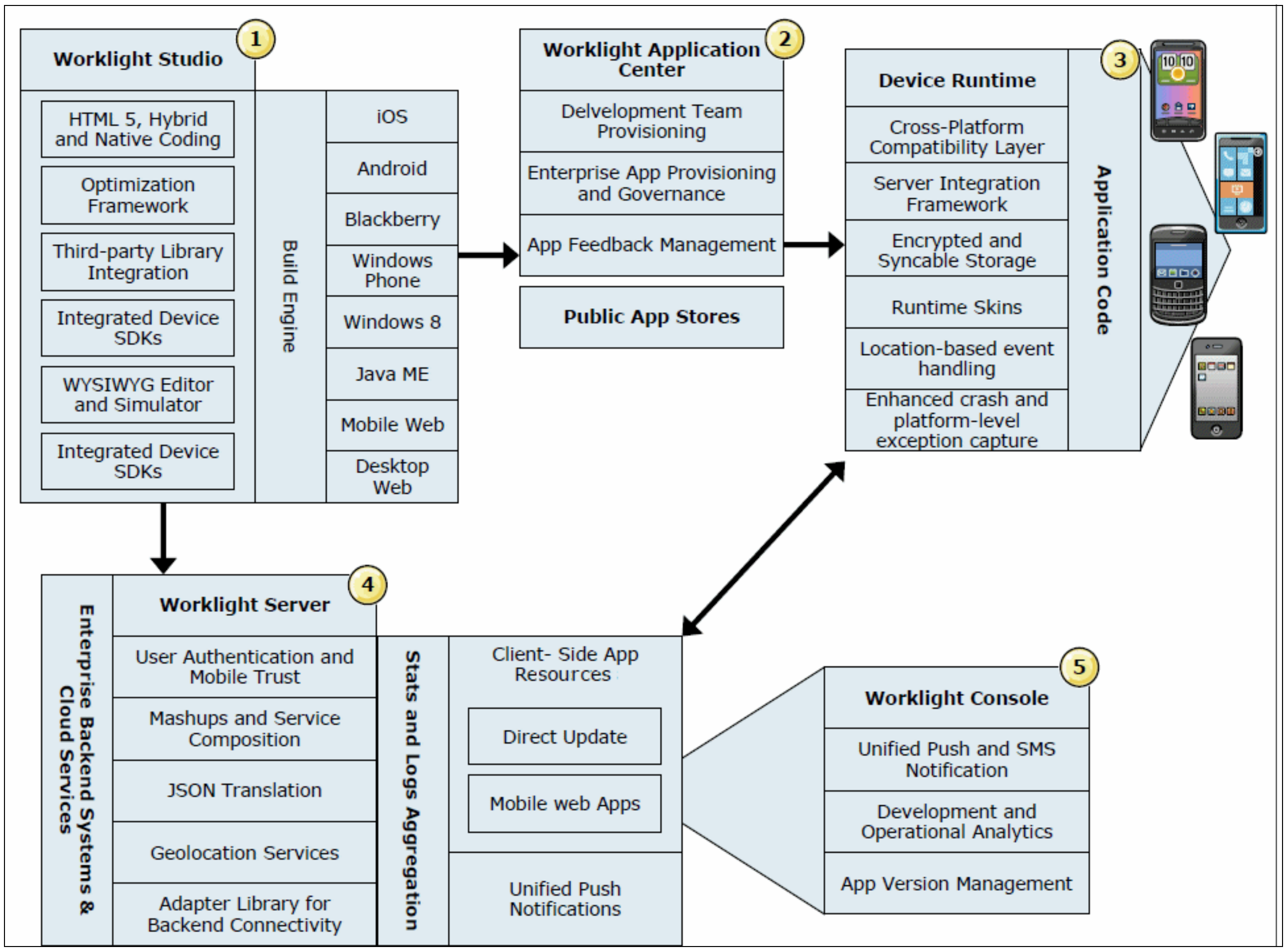

Worklight Studio
1
HTML 5, Hybrid and Native Coding
Optimization Framework
Third-party Library Integration
Integrated Device SDKs
WYSIWYG Editor and Simulator
Integrated Device SDKs
Build Engine
iOS
Android
Blackberry
Windows Phone
Windows 8
Java ME
Mobile Web
Desktop Web
Worklight Application Center
2
Delvelopment Team Provisioning
Enterprise App Provisioning and Governance
App Feedback Management
Public App Stores
Device Runtime
3
Cross-Platform Compatibility Layer
Server Integration Framework
Encrypted and Syncable Storage
Runtime Skins
Location-based event handling
Enhanced crash and platform-level exception capture
Application Code
Enterprise Backend Systems & Cloud Services
Worklight Server
4
User Authentication and Mobile Trust
Mashups and Service Composition
JSON Translation
Geolocation Services
Adapter Library for Backend Connectivity
Stats and Logs Aggregation
Client- Side App Resources
Direct Update
Mobile web Apps
Unified Push Notifications
Worklight Console
5
Unified Push and SMS Notification
Development and Operational Analytics
App Version Management

Figure 1-4 IBM Worklight platform components

To provide secure transactions with employees using their own mobile devices, accessing corporate resources from around the world, your IT team needs to gain close control over those devices to keep your networks safe and efficient. In the following sections, you can see an overview of what the IBM security products and appliances are able to accomplish.

1.5.2 User protection with IBM Security Access Manager

IBM Security Access Manager for Cloud and Mobile *extends user access protection to mobile and cloud environments* using federated single sign-on (SSO), user authentication, and risk scoring. IBM Security Access Manager includes the following features:

- Context-based access management for mobile end points, such as smartphones and tablets, to avoid inadvertent exposure of sensitive IT assets in an insecure environment
- Authentication and authorization of mobile app users and devices, with advanced session management and supported integration with IBM Worklight
- Centralized user access management to private and public cloud applications and services
- Risk-based access as a pluggable and configurable component
- Advanced web application protection from mobile devices
- A fast time-to-value and low total cost of ownership solution that makes minimal demands on the organization's IT staff
- Enhanced user productivity, better user experience, and reduced administration costs

For more information, see the following website:

http://www-03.ibm.com/software/products/us/en/samcm

Chapter 6, "Integration with IBM Security Access Manager" on page 101, provides information about IBM Security Access Manager and its integration with IBM Worklight.

1.5.3 Application security testing with IBM Security AppScan

IBM Security AppScan is designed *to manage vulnerability testing* throughout your company's software development lifecycle. It scans and tests all common web application vulnerabilities:

- SQL-injection
- Cross-site scripting
- Buffer overflow
- New flash and flex applications
- Web 2.0 exposure scans.

In addition, IBM Security AppScan V8.7 features a next-generation dynamic application security scanning engine and the innovative, all-new XSS-Analyzer. It provides the ability to identify and remediate code vulnerabilities by taking advantage of security insights from over 40,000 analyzed iOS and Android APIs.

IBM Security AppScan Enterprise provides the following functionality:

- A strategic approach to web application security
- Broad scanning capabilities to scan and test hundreds of applications simultaneously, and then retest them frequently
- Enterprise-level reporting that facilitates communication of security status and issues
- Remediation features that issue advisories to help guide developers in effective remediation

For more information, see the following website:

ftp://public.dhe.ibm.com/common/ssi/ecm/en/rab14001usen/RAB14001USEN.PDF

1.5.4 Delivery of services and applications with IBM WebSphere DataPower

WebSphere DataPower Appliances *simplify, govern, and optimize the delivery of services and applications*, and to enhance the security of XML and IT services. They extend the capabilities of an infrastructure by providing a multitude of functions.

As IBM has grown its line of WebSphere DataPower Appliances, the capabilities have expanded from the core business of service-oriented architecture (SOA) connectivity. WebSphere DataPower Appliances now serve areas of business-to-business (B2B) connectivity and web application proxying.

These appliances also support Web 2.0 integration with JavaScript Object Notation (JSON) and Representational State Transfer (REST), advanced application caching, rapid integration with cloud-based systems, and more.

For more information about the WebSphere DataPower Appliances including core functions and add-ons see *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327.

Chapter 7, "Integration with IBM WebSphere DataPower" on page 139, provides information about the integration of WebSphere DataPower with IBM Worklight.

1.5.5 Security intelligence with IBM Security QRadar SIEM

IBM Security QRadar security information and event management (SIEM) *consolidates log source event data* from thousands of devices, endpoints, and applications distributed throughout a network. It performs immediate normalization and correlation activities on raw data to distinguish real threats from false positives.

As an option, this software incorporates IBM Security X-Force® Threat Intelligence, which *supplies a list of potentially malicious IP addresses*, including malware hosts, spam sources, and other threats. IBM Security QRadar SIEM can also *correlate system vulnerabilities with event and network data*, helping to prioritize security incidents.

IBM Security QRadar SIEM performs the following functions:

- Provides near real-time visibility for threat detection and prioritization, delivering surveillance throughout the entire IT infrastructure
- Reduces and prioritizes alerts to focus investigations on a list of suspected incidents on which you can act
- Enables more effective threat management while producing detailed data access and user activity reports
- Supports easier, faster installation, and includes time-saving tools and features
- Produces detailed data access and user activity reports to help manage compliance

For more information, see the following website:

http://public.dhe.ibm.com/common/ssi/ecm/en/wgd03021usen/WGD03021USEN.PDF

1.5.6 VPN software with IBM Mobile Connect

IBM Mobile Connect *provides a full-featured, wireless virtual private network (VPN)*. This software employs data encryption to deliver security-rich access to enterprise applications over wireless and wired networks. IBM Mobile Connect enables access to enterprise applications and data from virtually any location, while protecting an organization's sensitive information.

IBM Mobile Connect (formerly IBM Lotus® Mobile Connect) provides the following features:

- Rich security features to protect transmission of sensitive data delivered through customer or customer-less access
- Fast, continuous network connectivity for uninterrupted access to enterprise information and applications
- Built-in controls that can help reduce costs through lower connection charges and transmission fees
- Simple maintenance for administrative ease and control
- Support for a variety of operating systems, mobile devices, and networks to help satisfy the varying needs of remote users

For more information, see the following website:

`http://www-03.ibm.com/software/products/us/en/mobile-connect/`

2

Business scenario used in this book

This chapter describes the mobile application (app) strategy and business requirements of a fictitious financial institution, *Banking Company A*. The fictitious company used in this scenario employed an enterprise mobile app lifecycle process to design, develop, and deploy secure mobile apps for their employees and clients.

Throughout this book, information about individual lifecycle phases is provided. These phases are demonstrated in the context of IBM Worklight, focusing on security concepts and integration.

This chapter explores the initial requirements-gathering and decision-making phases of the lifecycle that led to the adoption of Worklight and security integration.

After reading this chapter, you will have an overview of the decisions and strategy adopted by Banking Company A, and an understanding of how the security features of Worklight address their requirements.

2.1 Mobile strategy business drivers

Banking Company A is a progressive financial institution that embraces technology to streamline their business processes and provide enhanced customer service. As a financial institution, security across all areas of their business is paramount, with no exceptions made for ease of use or enablement.

Having completed an enterprise mobile app lifecycle review as shown in 1.2.5, “Putting it all together” on page 6, four key business drivers emerged for Banking Company A:

- A secure platform foundation
- Increased staff productivity
- Secure and easy access for customers
- Rapid development and deployment

By selecting IBM Worklight as their mobile app platform, Banking Company A is able to achieve accelerated deployment by taking advantage of its development and deployment tools, which are built on a secure and scalable infrastructure.

They chose to implement a hybrid mobile app, allowing them to best meet the varying platform matrix of their customers, provide dedicated support for enterprise-owned mobile devices, and (where approved) facilitate a bring your own device (BYOD) policy for their employees.

2.1.1 A secure platform foundation

Building on a stable and secure foundation is essential for any development project, whether it is physical construction and development or software development and engineering.

The existing infrastructure of Banking Company A is already highly secured, and exceeds the minimum industry and government regulations. When considering their mobile app strategy, taking advantage of the existing secure platform is key to ensuring that no new exploitable systems are introduced.

Integration with existing directories and data stores

Banking Company A maintains staff user accounts in a separate and isolated infrastructure from their customer user accounts and data. Employees do need access to customer account data to view transactions, create and approve apps, and perform day to day financial institution activities.

Banking Company A developed their identity management and authentication infrastructure, and are taking advantage of a web reverse proxy to reduce the complexity of authenticating to these systems. Because of this, they are able to configure Worklight to work with these security components using its flexible authentication integration framework.

Using Worklight, Banking Company A is able to do all of the following:

- Authenticate employees using their enterprise Lightweight Directory Access Protocol (LDAP) directory server logon.
- Authenticate customers from their database.
- Provide single sign-on (SSO) using their web reverse proxy.

The supported directory and data store authentication mechanisms are explored in detail in 4.1.5, “Security tests” on page 68.

Take advantage of existing authentication mechanisms

Limiting the number of new system deployments, and reducing change requests for existing systems, is highly desirable when you want to provide mobile connectivity while reusing as much of the existing capabilities as possible.

By not having to modify existing authentication mechanisms and protocols where existing secure access is established, enterprises reduce costs and maintain their security posture. Using existing access limits vulnerabilities introduced in making those changes.

Detailed information about supported authentication integration mechanisms is provided in 4.1.4, “Login modules” on page 61.

Banking Company A enabled further security enhancements by integrating with IBM Security Access Manager for Cloud and Mobile. Using this product, Banking Company A takes advantage of a single robust and scalable infrastructure to deliver centralized management of user privileges for all of their Worklight mobile apps.

IBM Security Access Manager's reverse proxy, deployed in the DMZ (a firewall configuration for securing local area networks), is used as a security enforcement point to validate a user's credentials on behalf of the Worklight Server before access is granted to the app's data or back-end systems.

Further details about the integration capabilities with IBM Security Access Manager are provided in Chapter 6, “Integration with IBM Security Access Manager” on page 101.

Banking Company A can also achieve directory integration and user propagation for SSO using IBM WebSphere DataPower, which is shown in Chapter 7, “Integration with IBM WebSphere DataPower” on page 139. The WebSphere DataPower Appliance can act as a powerful enterprise service bus (ESB) gateway, application firewall, and content accelerator when deployed as an intermediary between Worklight adapters and back-end systems.

In this scenario, IBM Security Access Manager is deployed to provide risk-based access and OAuth enablement to provide identity-aware mobile apps.

Application updates and authenticity testing

When providing a mobile app installed on employee and customer devices, Banking Company A must ensure their app is secure:

- It is not tampered with or modified in any way.
- It is not used to access restricted areas of their systems.
- It has not been modified and redistributed containing malware or exploits that can potentially compromise systems or capture confidential customer data.

The application authenticity feature of Worklight is used for this assurance, and is described in 5.3.4, “Control and confirm application authenticity” on page 96.

Further, if Banking Company A needs to add additional capabilities to their apps, or needs to resolve a critical flaw or vulnerability immediately (without relying on their users to download the latest version from the public or enterprise app store), they can take advantage of the Direct Update feature using the Worklight Console. This feature is described and demonstrated in 5.4, “Direct Update” on page 97.

Data integrity and encryption

All of the data accessed, transferred, and stored within the mobile app for Banking Company A requires encryption to comply with government and industry regulatory standards.

Using Worklight, Banking Company A is able to protect sensitive information from malware attacks and device theft. Using AES256 and PCKS#5 encryption for secure on device storage of app-generated information including random server-generated numbers for high security enables user authentication even when the Worklight server is offline.

Data encryption and integrity is demonstrated and described in 5.2, “Encrypted offline cache and JSONStore” on page 81.

To protect integrity and non-repudiation during the transmission of data, Secure Sockets Layer (SSL) with server identity verification is used. Worklight enables security-rich client and server communication over Hypertext Transfer Protocol Secure (HTTPS) to prevent data leakage, and uses automatic server certificate verification to thwart known attacks, such as man-in-the-middle attacks. Used in conjunction with their web reverse proxy, all data communications are always tightly secured using SSL.

Access control and notifications

Banking Company A can update their apps without relying on their users to update via an app store for JavaScript or Hypertext Markup Language V5 (HTML5) changes. They can also disable the old version of their app. This is for situations in which the distribution of a security fix requires that users obtain a complete, new version of the app from the app store.

Worklight Server can be configured to restrict the access of certain apps, based on their version and the device type, providing a targeted upgrade. Worklight Sever can also be used to disable a rogue or compromised device that is detected.

In both these security-related events, or for routine maintenance such as network or infrastructure outages, a notification can be displayed directly to the user by the *Notification Messaging* feature when the mobile app is started. The notification contains information about the outage so that users are informed, and are not faced with an unknown error that results in multiple calls to the help desk.

The *Remote Disable* feature is described in more detail in 5.5, “Remote Disable” on page 98.

2.1.2 Increased staff productivity

In an increasingly networked world, employees now require mobile connectivity, not just access from the office. In-home mobile lenders of Banking Company A can perform the following tasks directly while visiting their client:

- Evaluate loan requests.
- Complete transfer authorizations.
- Issue approvals.

The in-home lenders require access to critical business systems, but their access must not compromise corporate governance and security controls.

Accelerating daily tasks

When an employee is on site, all actions within their job function are allowed without necessarily needing to be accessed from a traditional desktop PC.

Routine tasks, such as balance inquiries, transactions, and loan approval, do not necessarily need to be done in front of a PC at a desk in the branch, and can instead be done on mobile devices while assisting customers in a comfortable waiting area or satellite desk.

Exceptional tasks, such as approving a loan after hours or from outside the office, require a second authority to verify the transaction approval.

Using the features described in “Integration with existing directories and data stores” on page 20, and “Data integrity and encryption” on page 21, Banking Company A is able to provide secure, location-aware, and step up (validation) access to their infrastructure for employees.

Customer service enhancements

A review of in-branch customer service inquiries revealed that, during peak times, branch employees can assist many customers waiting in the teller queue on the spot, and the customers did not actually need teller assistance.

By empowering employees with corporate-owned mobile devices and a subset of the branch teller capabilities, Banking Company A was able to reduce the workload on teller staff and decrease wait time for customers, driving a higher level of customer satisfaction.

To achieve this, Banking Company A used the capabilities described in “Take advantage of existing authentication mechanisms” on page 21.

Bring your own device

Banking Company A reviewed the strategic need for BYOD, and although there are benefits with mobile connectivity for employees to access corporate email and internal websites, a security review governed that private devices cannot be granted permission to access corporate systems for branch-related tasks.

To enforce this protocol, the Device Registration and Remote Disable features are used, as shown in “Access control and notifications” on page 22.

Branch Managers and Senior Executive staff are able to access corporate email and internal websites on their mobile devices remotely using IBM Mobile Connect. Information about this virtual private network (VPN) and access capability is provided in 1.5.6, “VPN software with IBM Mobile Connect” on page 17.

2.1.3 Secure and easy access for customers

A key feature that customers look for when choosing a financial institution is convenient access to their money where and when they need it. They take for granted that the system is inherently secure; however, users are becoming more savvy and security conscious, and expect their bank to provide additional security features.

Mobile apps are designed to be accessible from anywhere at any time, which introduces new security scenarios, because you can no longer rely on physical device ownership and location. The mobile apps themselves, along with their supporting infrastructure, must provide additional security controls to overcome the mobile challenges.

Secure transactions

All connectivity and transactional data and storage on the device must be secured, as shown in “Data integrity and encryption” on page 21.

Depending on the type of transaction being performed with Banking Company A, customers require an additional authentication validation for high-value transactions. For example, when customers simply want to check their balance, they open the mobile app and provide their credentials:

- User names and passwords
- Personal identification numbers (PINs)
- Indiscernible authentication tokens that are stored on the devices

Successfully starting the mobile app does not prove that the owner of the device is the person using the device, because the device can be misplaced or the account details can be compromised. As a safeguard, any high-value transactions, such as a transfer of funds greater than $200, require an additional authentication to validate the identity.

Banking Company A implemented the additional authentication challenge using risk-based access (RBA), as described in 6.3, “Risk-based access” on page 121.

Without requiring changes to the app, Banking Company A can seamlessly add additional fraud detection systems to validate the location of the device in relation to the customer’s listed home and postal address, and the use the time that the transaction is occurring. This is achieved by implementing additional policies in the RBA system, and can be applied and updated without disruption to the online services.

These security measures provide additional assurances for customers, and increased fraud prevention for the bank, to protect unusual transactions from different locations and nonstandard hours.

Ease of access

While most customers enjoy the convenience of the mobile banking app from Banking Company A, many customers also use the stock trading app.

When customers start the banking app on their mobile devices and check their balance for available funds to invest, they then typically open the trading app to begin watching, buying, and selling.

If no other apps from Banking Company A are running or authenticated, users must log in to their trading app to start using it. However, suppose they have logged in to the mobile banking app, checked their balances, and then completed a high-value transaction that required a step up authentication to validate their identity.

In that case, they can start the mobile trading app without reauthenticating. This is achieved using the techniques shown in 5.1.2, “Device single sign-on” on page 79.

Reassurance and convenience

When reviewing their mobile strategy, Banking Company A found that many of their mobile clients have two or more devices, including a smartphone and a tablet. In some instances, they also access their account details from their partner’s devices.

Customers are provisioned for mobile banking access, and their device is registered. Later, if that device is temporarily misplaced, lost, or stolen, access for that one device can be revoked without needing to change their password or affect access on their other devices.

In this instance, the customers are not normally at home, and do not have access to their PC, so this can be completed by these simple actions:

- Phoning the bank
- Validating their identity
- Advising which of their devices needs to be disabled:
  - Temporarily
  - Permanently

For additional convenience, a device can be temporarily disabled using their traditional desktop internet banking. For example, a customer can temporarily disable a device when going on an overseas holiday, or permanently disable device access before giving the device to a child or disposing of it.

The device registration feature is described in 5.3, “Client-side device provisioning and application authenticity” on page 90, and the temporary or permanent disabling of the device is demonstrated in the authentication process flow of 6.2.2, “Tivoli Federated Identity Manager OAuth single sign-on” on page 111.

2.1.4 Rapid development and deployment

To deliver mobile apps to their employees and customers, Banking Company A first needs to build their apps. Later, they will need to maintain, update, and enhance their existing apps, and provide new apps in the future.

Using Worklight Studio, they are able to take advantage of the features and capabilities of the Worklight customer, and also build and reuse Worklight shell components for common content in their apps.

The Worklight Console and the Worklight Application Center can then be used to rapidly distribute, update, control, and gather feedback on their apps.

Common toolkits

Worklight Studio enables the development of rich multiplatform apps using HTML5 and JavaScript for hybrid and native application types. It provides access to device application programming interfaces (APIs) using native code, or standard web languages, over a uniform Apache Cordova (PhoneGap) bridge for portability.

Worklight Studio also easily integrates third-party tools, libraries, and frameworks, such as JQuery Mobile, Sencha Touch, and Dojo Mobile. Therefore, Banking Company A can deploy mobile apps that most of their customers can use.

The Worklight Studio can also be used to design, develop, and test Worklight adapters for back-end data source connectivity and data retrieval.

Secure applications and endpoints

As part of end-to-end security and vulnerability testing, Banking Company A employs the capabilities of the IBM Security AppScan suite to ensure endpoint security for their web-based systems, and to perform static analysis on their mobile app source code.

AppScan Source provides support for native code written for Android and iOS apps, to detect and correct errors such as data leakage. It includes language support for Objective-C, JavaScript, and Java, providing the ability to perform call and data-flow analysis that will generate trace information to help isolate mobile-specific security risks.

AppScan Standard provides broad coverage of emerging threats, including for Web 2.0 application vulnerabilities, Cross-Site Scripting (XSS) Analyzer for cutting-edge XSS detection and exploitation, and JavaScript Security Analyzer for static analysis of client-side security issues.

The use of these tools as part of both the development phase and ongoing monitoring ensures that Banking Company A provides secure apps connecting to secure systems.

Feedback and usability analytics

The Worklight Console is used by Banking Company A to collect and analyze user statistics. In addition to the Remote Disable and Notification Messaging capabilities illustrated in “Access control and notifications” on page 22, it can also be used to perform other tasks.

They can use the Worklight Console to complete these actions:

- Monitor all deployed apps, adapters, and push notification rules.
- Collect user statistics from all running apps.
- Generate built-in, pre-configured user adoption and usage reports.
- Configure data collection rules for application-specific events.

Integrating with Worklight Application Center, IBM Endpoint Manager for Mobile Devices, and IBM Tealeaf CX Mobile, Banking Company A can collect and analyze security-related data:

- Determine actual usage patterns.
- Identify compromised or jailbroken (rooted) devices.
- Rapidly deploy new apps for testing.
- Solicit and map feedback on their apps.

These technologies and products are shown in 1.2.4, “Optimization of mobile experiences with IBM MobileFirst Analytics” on page 6.

2.2 Conclusion

Having completed an enterprise mobile app lifecycle, taking advantage of Worklight for their mobile app development, testing, and deployment, Banking Company A delivered secure mobile apps for their employees and customers.

The security features of Worklight ensured a successful integration with their already highly secure infrastructure:

- Enabled the rapid development, distribution, and adoption of their mobile apps
- Reduced costs
- Enhanced employee productivity
- Created increased customer satisfaction

Throughout this book, each chapter explores one or more of the business drivers for Banking Company A, and describes in more detail the relevant Worklight security features.

3

IBM Worklight security overview

This chapter provides an architectural overview of the security capabilities of IBM Worklight. It also introduces the general security principles, concepts, and terminology with which Worklight is compliant.

This is your starting point for understanding these security capabilities. Subsequent chapters in this book delve deeper into each Worklight security feature, and provide details about how it can be used to secure your mobile application (app), its data, and access to your enterprise resources.

3.1 Security principles and concepts

This section uses the fictitious scenario of Banking Company A to illustrate the need for mobile security. This scenario is described in Chapter 2, “Business scenario used in this book” on page 19.

Enterprises, including the fictitious Banking Company A, need to overcome various security threats to take advantage of mobile technologies. To stay competitive in the industry and satisfy customer needs, Banking Company A needs to achieve information security through compliance with the following principles and concepts:

Authentication	Ensures that the identities of both the sender and the receiver of a network transaction are true. Authentication processes can be interactive or noninteractive.
Authorization	Grants a user, system, or process either complete or restricted access to an object, resource, or function.
Confidentiality	Protects sensitive data from unauthorized disclosure. Common techniques of achieving confidentiality include access control and cryptographic encryption of data.
Integrity	Assures that information that arrives at a destination is the same as the information that was sent. You can use message digests or checksums that are computed at two different times to detect whether the data has been inappropriately modified.
Nonrepudiation	Proves that a transaction occurred, or that a message was sent or received. Nonrepudiation is supported by the use of digital certificates and public key cryptography to digitally sign transactions, messages, and documents.

3.1.1 Authentication and authorization

What do authentication and authorization mean for Banking Company A and its mobile users? For Banking Company A, authentication means verifying the identity of mobile customers, and allowing customers to identify the company. In addition, customers of Banking Company A need to obtain user authorization for mobile banking and payment transactions.

Users of Banking Company A’s mobile banking apps, including the company’s customers and employees, want to protect their important and sensitive information from malware attacks and device theft. They also want to authenticate one time to access multiple protected resources.

Consider the following possibilities if Banking Company A does not meet authentication and authorization requirements:

- With stolen devices, thieves can retrieve user credentials that are stored on memory cards.
- Cyber criminals can bypass authentication controls.
- Malware can intercept text messages that are sent from the bank to the customers to authenticate transactions.
- The authentication process for each transaction is not continuous, so invalid transactions are possible.

To address these challenges, Banking Company A is looking for a multifactor authentication that integrates with their existing authentication infrastructure. By using multi-factor authentication, Banking Company A can perform multiple checks to control access to any protected resources. For instance, the company can verify the devices, the users, and the apps.

Developers at Banking Company A are searching for ways to encrypt the data that is stored on user devices. They also think it is necessary to provide a mechanism that can authenticate offline users.

3.1.2 Confidentiality, integrity, and nonrepudiation

What do confidentiality, integrity, and nonrepudiation mean for Banking Company A?

Banking Company A needs to keep its customer, company, and employee data safe, prevent the data from being maliciously altered, and be able to track the actions that employees and customers perform.

Users of Banking Company A's mobile banking app want their accounts and transactions to be secure. They also want the updates of their mobile apps to require only minimum user involvement.

Here are some possible consequences if Banking Company A fails to maintain data confidentiality, integrity, and nonrepudiation:

- Confidential information, such as details about transactions, can be sniffed.
- During a fund transfer, cyber criminals can modify the account details and the amount of money that is being transferred.
- The servers of Banking Company A are vulnerable to man-in-the-middle attacks.
- It is possible for the mobile banking users of Banking Company A to deny that they sent messages or performed transactions.

To address these challenges, Banking Company A wants to prevent data leakage. It is also thinking of striking a balance between the security measures it takes and the convenience and user experience it brings to its employees and customers.

Developers at Banking Company A are looking into adopting Secure Sockets Layer (SSL), the underpinning of Hypertext Transfer Protocol Secure (HTTPS). In the meantime, they still need to ensure that many of the traditional security pitfalls do not affect the security environment of the mobile apps that they develop. They also want to integrate that security environment with other enterprise security solutions.

3.1.3 Other security concepts

Here are some other important security concepts that you need to understand when reading this book:

Auditing	The act of confirming compliance with a standard or set of guidelines, comparing actual measurements to targets, or verifying the accuracy of recorded information.
Encryption	The process of transforming data into an unintelligible form in such a way that the original data either cannot be obtained or can be obtained only by using a decryption process.

Single sign-on (SSO) An authentication process in which a user can access more than one system or app by entering a single credential (for example, a user ID and password).

3.2 IBM Worklight security capabilities

IBM Worklight provides a set of security capabilities that address the following mobile app security objectives (see Figure 3-1).

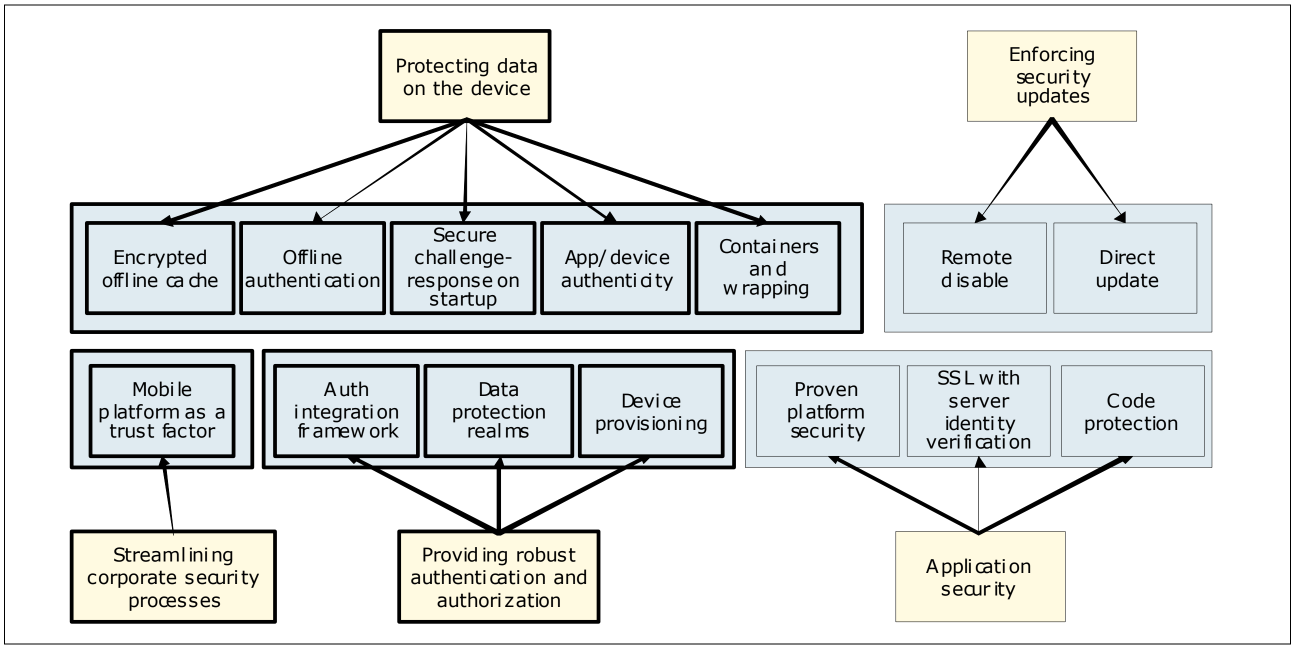

Protecting data on the device
Enforcing security updates
Encrypted offline cache
Offline authentication
Secure challenge-response on startup
App/device authenticity
Containers and wrapping
Remote disable
Direct update
Mobile platform as a trust factor
Auth integration framework
Data protection realms
Device provisioning
Proven platform security
SSL with server identity verification
Code protection
Streamlining corporate security processes
Providing robust authentication and authorization
Application security

Figure 3-1 Worklight security capability mapping to mobile app security objectives

3.2.1 Protect the data on the device

It is common for the mobile app user to have access to sensitive data that can be stored on the mobile device. However, this data stored on-device can potentially be stolen or tampered with by malware existing on the device. In the event that the device is lost or stolen, this sensitive data can be extracted by unauthorized third parties.

In addition, the mobile app can be required to function in an offline context (without any back-end connectivity), and at the same time require that only authenticated users be allowed access to the data stored on the mobile device.

Worklight provides capabilities that help protect the data on the device:

Encrypted on-device storage

Provides security for the data stored on the device by encrypting the data using the advanced encryption standard (AES) algorithm with 256-bit keys generated using the public-key cryptography standards #5 (PKCS5) algorithm based on a user-supplied password and Worklight internal mechanisms.

The data (encrypted or not) can be stored on the device, either as HTML5 local storage, known in Worklight as a *cache*, or in a Worklight-supplied mobile storage feature known as *JSONStore*.

Offline authentication

Is needed when the mobile app does not have network connectivity, but user verification is still needed to access the on-device data. In these scenarios, the encrypted storage itself can be used to authenticate the user, because the encrypted data can only be unlocked using the correct password supplied by the user.

Therefore, the encrypted storage can be used to achieve more secure offline authentication.

3.2.2 Protect the application

In addition to protecting the on-device data, it is also important to protect the mobile app itself on the device. This prevents hackers from unpackaging a legitimate mobile app, and then repackaging it with malicious code. These tampered apps present a significant risk to enterprise data, and to the company brand. You can use Worklight to reduce these risks:

- Worklight provides capabilities to obfuscate and encrypt the app code and web resources, to prevent tampering with the app.
- In certain scenarios, there is also a requirement to ensure that the mobile app is only accessible from authorized mobile devices, to protect access to sensitive data. This requirement helps support the bring your own device (BYOD) trend, allowing employees to use their personal devices for work purposes. At the same time, it is important to maintain and enforce security protocols on those devices.

  Worklight also provides application and device authenticity, testing to ensure that only valid (Worklight Server runtime-aware) apps running on authorized mobile devices can be used.

  Worklight generates a unique identification for the devices, and protects them from tampering by signing the request. The application authenticity is enforced as part of the Worklight platform, using an application signature and the application properties of the iOS and Android operating system.

  Whenever the app tries to access back-end systems through the Worklight server, the server verifies the app's authenticity (and device authenticity, if activated) and only enables access from legitimate apps.
- In addition to ensuring that the app and the device on which the app is running are both authentic and valid, Worklight also supports the traditional methods of authenticating the user before allowing access to the app. This includes a supplied user name and password (for example, credential validation), or even more sophisticated methods, such as multi-factor authentication.

  By combining multiple authenticity tests (multi-factor authentication), Worklight can enforce more stringent levels of security for the app, device, and user. For example, by requiring app, device, and user authenticity tests, it is possible to allow access only to *this* legitimate app running on *that* authorized device for *this* authenticated user.

  An important component of Worklight's authentication capability is the challenge-response mechanism provided by Worklight to ensure that only an authenticated user can access the mobile app or its functions. This challenge-response mechanism is a key part of Worklight's security framework. For more information, see 3.3, "IBM Worklight security framework" on page 34.
- Worklight also extends the concept of SSO to the apps on the mobile device, so that authenticating to one app means that the user does not have to authenticate to other apps on that device if these apps share the same authentication context. This feature is known in Worklight as *Device SSO*.

- Worklight enables a security-rich client and server communication over HTTPS to prevent data leakage, and automatic server certificate verification to thwart known attacks, such as man-in-the-middle attacks.
- Worklight is also flexible enough to work with other third-party mobile security and device management platforms, which provide different levels of security for mobile apps, such as containers, app wrapping, and so on.

3.2.3 Ensure security updates

In today's mobile world, users are free to choose whether to download and install the latest release of a mobile app from an app store. In addition, the mobile app store distribution mechanisms, and contractual restrictions placed by some mobile operating system vendors, make it challenging to maintain an app versioning strategy.

It is difficult to ensure that users are downloading and running the correct version of the app in a timely manner. In the event that a fix is needed to correct a security flaw in the app, a timely propagation of security updates is essential to mitigate the possibility of critical problems.

Worklight provides the *Direct Update* and *Remote Disable* features to help administrators ensure that critical updates are delivered to the apps on the mobile devices:

- The Direct Update feature enables developers to drive updates of the web content of their deployed HTML5 and hybrid apps directly from the Worklight Server upon app launch. This mechanism helps developers ensure that critical updates to JavaScript code reach the user in a timely manner, silently or by user confirmation, while it maintains compliance with mobile operating system (OS) terms of service.
- The Remote Disable feature provides administrators with the ability to disable the old version of the app for situations in which the distribution of a security fix requires that users get the new app version from the app store. The Worklight Server can be configured to restrict the access of certain apps, based on their version and the device type on which they are installed.

  The Remote Disable feature can also be used to prevent apps from accessing back-end systems temporarily, perhaps in the event that the back-end systems are offline for maintenance. This remote disabling of the app provides a graceful way of restricting access, as opposed to a system error message indicating a back-end system connectivity problem.

3.2.4 Streamline corporate security processes

A Worklight hybrid mobile app consists of web resources (HTLM5, JavaScript, and CSS3) surrounded by a Worklight-provided native mobile platform shell. This shell provides access to the native mobile device capabilities, such as the camera, a common user interface, the Worklight authentication framework, and security configuration.

Worklight apps can use customized shells that provide common components used by all apps in your enterprise, allowing access to only specific features of mobile devices, and enforcing the security requirements of the mobile apps in your organization. This helps companies ensure that apps using Worklight are trusted entities that adhere to corporate security policies, and therefore speed up the approval process.

Worklight also supports the concept of using the mobile device as a trust factor, where certificates provisioned to the device by a trusted third party (for example, a mobile device management (MDM) solution) are accepted as part of the authentication process.

3.2.5 Provide robust authentication and authorization

Many companies have developed identity and authentication infrastructure today that is sophisticated and complex due to their business's stringent security requirements, such as two-factor authentication and non-trivial management of tokens and credentials.

Worklight's authentication integration framework has been designed to simplify the task of connecting mobile apps with the enterprise back-end authentication infrastructure. The Worklight framework provides both server-side and client-side mechanisms for assisting with this task.

On the Worklight server side, components define the collection and handling of credentials (Authenticator) and mechanisms to validate or verify the credentials (Login module). These server-side components interact with the client side (mobile app and adapters) security framework in a challenge-response process to ensure that only authenticated identities are used to access protected resources.

This framework can be extended or customized on both the client and server sides.

Worklight also supports a number of commonly used mechanisms for authentication, such as forms-based, cookie-based, or header-based, and other methods. For a complete list of the authentication mechanisms supported by Worklight, see Chapter 4, “Integrating Worklight with enterprise security” on page 41.

In addition to providing its own security framework, Worklight also integrates with the underlying Java Platform, Enterprise Edition app server's own security mechanisms. For example, when running in a WebSphere Application Server or a WebSphere Liberty Profile environment, Worklight security is further strengthened by integration with the following components:

- The application server-supported user registries (for example, operating system, Lightweight Directory Access Protocol (LDAP), federated repositories, and so on)
- Single sign-on tokens, such as Lightweight Third Party Authentication (LTPA)
- Security interceptors, such as the trust association interceptor (TAI)
- The propagation of security credentials through to back-end systems

Resources are protected by authentication realms that define the process to be used to authenticate users, and consists of a mechanism to collect the user credentials (authenticator) and verifying the user credentials (login module) against a user registry (for example, a database or LDAP directory).

When a user attempts to access a protected resource, Worklight checks whether the user is already authenticated according to the process defined for the resource's realm. If the user has not yet been authenticated, Worklight triggers the challenge-response process of obtaining the client credentials and verifying them as defined in the realm.

Worklight differentiates between realms that are used for personalization, such as user preferences, and those that are used for authentication. There are different kinds of realms:

Environment realms	Are used for personalization. In such realms, a user can be identified by a persistent cookie or by user identity in the hosting environment (for example, a Google account or a Facebook account).
Resource realms	Are used for authentication. Such realms will typically use mechanisms, such as form-based authentication or SSO, to collect user credentials, and an authentication service to validate them.

Worklight’s flexible security framework enables mobile apps to integrate with reverse proxies and security gateways, such as IBM Security Access Manager and IBM WebSphere DataPower, by taking advantage of the header-based authentication mechanism (Authenticator and Login module) provided by Worklight.

In this scenario, the reverse proxy validates the user’s credentials and forwards a custom HTTP header with the user identity to Worklight. If Worklight is configured to trust the header (and, by extension, the reverse proxy), Worklight authenticates the session based on the user identity in the header.

In addition to using a custom HTTP header to establish trust with Worklight, IBM Security Access Manager and IBM WebSphere DataPower can also use the IBM proprietary LTPA token as a more secure means of establishing the user identity with Worklight. This LTPA token is sent to Worklight as an HTTP cookie.

3.3 IBM Worklight security framework

The IBM Worklight security framework serves two main goals. It controls access to protected resources and it propagates the user (or server) identity to the back-end systems through the adapter framework.

Adapters: An adapter uses JavaScript code running in the Worklight Server to access back-end systems on behalf of the mobile app. This JavaScript code can also be extended by using Java code, meaning an adapter can be developed as a combination of server-side JavaScript and Java code.

The key to the Worklight security framework is that it does not include its own user registry, credentials storage or access control management. Instead, it delegates all those functions to the existing enterprise security infrastructure. This delegation enables the Worklight Server to integrate smoothly as a presentation tier into the existing enterprise landscape. Integration with the existing security infrastructure is an important feature of the Worklight security framework and supports custom extensions that allow integration with virtually any security mechanism.

The Worklight security framework requires security checks before allowing access to any protected resources and provides support for multi-factor authentication. Therefore, any protected resource can require multiple security checks before allowing access. A typical example of multi-factor authentication is the combination of device, app, and user authentication.

A protected resource can be any of the following items:

Application	Any request to the app requires successful authentication in all realms of the security test that is defined in the app descriptor (`application-descriptor.xml`).
Adapter procedure	Procedure invocation requires successful authentication in all realms of the security test that is defined in the adapter descriptor. The user identity and credentials that are obtained during such authentication can be propagated downstream to the enterprise information system represented by this adapter.
Event source	Subscription to push notifications requires successful authentication in all realms of the security test that are defined in the event source definition (in the corresponding adapter JavaScript code).

Static resource Static resources are defined as URL patterns in the authentication configuration file (`authenticationConfig.xml`). They allow protection of static web apps such as the Worklight Console.

Each protected resource is defined as being protected by a named entity called a *Security Test* which is a collection of security checks that needs to be done before access is allowed to the protected resource. Each security check is defined as a *Realm* which defines how the user credentials are collected (*Authenticator*) and how those credentials are validated (*Login module*).

All of these security configuration entities are defined in a single configuration file (`authenticationConfig.xml`) per Worklight application project so that these definitions can be reused across different protected resources related to that app. For an example of the security configuration file, see Chapter 4, “Integrating Worklight with enterprise security” on page 41.

An implementation of security checks usually includes a client part and a server part. The two parts interact with each other according to their private protocol which is usually a sequence of challenges that are sent by the server and responses that are returned by the client.

The Worklight security framework provides a wire protocol which enables for the combination of challenges and responses for multiple security checks to be included in a single request-and-response round trip. The protocol serves two important purposes: it enables the number of extra round trips between the client and server to be minimized and it separates the app business logic from the security checks implementation.

During a session, an app can access different protected resources. The results of the authentication in different realms are stored in the single session authentication context. These results are then shared among all of the protected resources in the scope of the current session (see Figure 3-2).

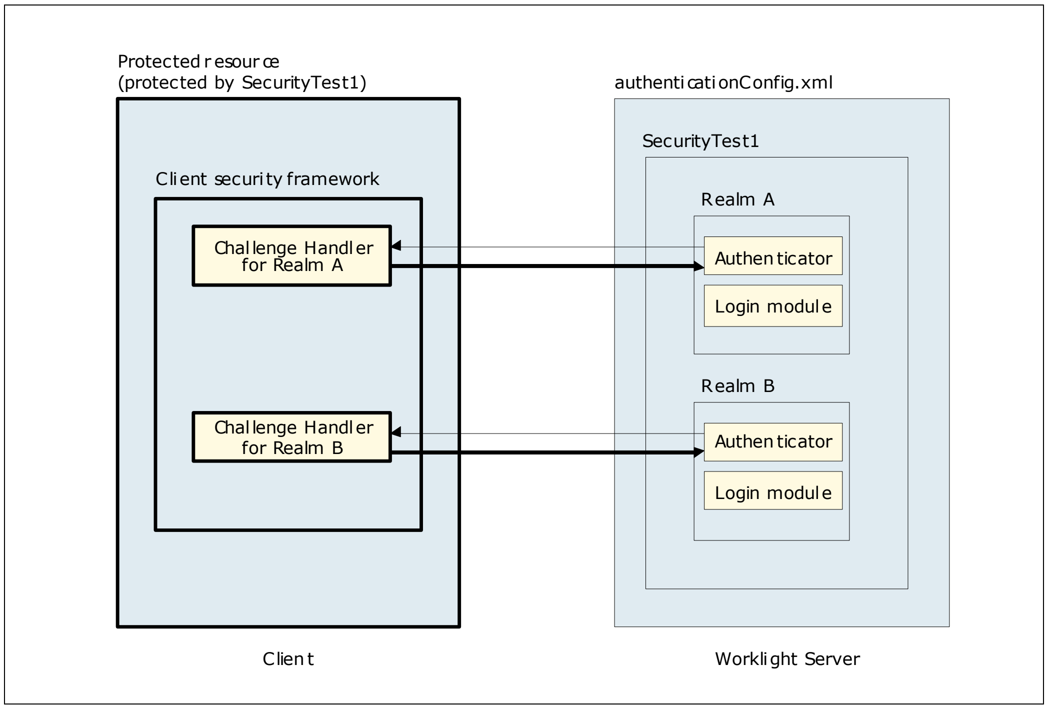


Figure 3-2 Worklight authentication architecture

3.3.1 Realms and security tests

A realm represents a fully configured security check that must be completed before it can allow access to a protected resource. The semantics of the checks are not limited to authentication, but can implement any logic that can serve as protection for the server-side app resources:

- User authentication
- Device authentication and provisioning
- Application authenticity check
- Remote application disable
- Direct update
- Anti-cross-site request forgery (XSRF) check

The realms are defined in the authentication configuration file on the Worklight project level. A realm consists of the authenticator and the login module. The authenticator obtains the credentials from the client, and the login module validates those credentials and builds the user identity.

Worklight provides a set of built-in authenticators and login modules, but it also provides the ability to develop and deploy custom authenticators and login modules in custom realms to handle any special security checks that are required, for example, handling special security tokens.

The realms are grouped into security tests which are defined in the same authentication configuration file. The security test defines the group of realms, in addition to the order in which they must be checked. For example, it often makes sense not to ask for the user credentials until it is certain that the app itself is authentic.

Some of realms are relevant only to mobile or to web environments and Worklight provides predefined security test configurations for the mobile (mobileSecurityTest) and web (webSecurityTest) environments. These security test configurations can be modified to suit different security requirements. It is also possible to create a custom security test from scratch with custom realms and custom authenticators and login modules.

3.3.2 Worklight protocol and client challenge handlers

Each security check defines its own protocol, which is a sequence of challenges that are sent by the server and responses that are sent by the client. On the server side, the component that implements this private protocol is the authenticator. On the client side, the corresponding component is called the challenge handler.

When the client request tries to access to a protected resource, the Worklight Server checks all the appropriate realms specified in the defined security test. One or more realms can decide to send a challenge to the client. Challenges from multiple realms are composed into a single response and sent back to the client.

The Worklight client security framework extracts the individual challenges from the response and routes them to the appropriate challenge handlers defined in the mobile app code. When a challenge handler finishes the processing, it submits its response to the Worklight client security framework. As an example, this occurs when it obtains the user name and password from a login user interface. When all the responses are received, the Worklight client security framework resends the original request with all the challenge responses.

The Worklight Server extracts those responses from the request and passes them to the appropriate authenticators. If an authenticator is satisfied, it reports a success and the

Worklight Server calls the login module. If the login module succeeds in validating all of the credentials, the realm is considered successfully authenticated. If all the realms of the security test are successfully authenticated, the Worklight Server enables the request processing to proceed.

If a realm check fails, its authenticator sends another (or the same) challenge to the client and the whole security challenge-response process repeats itself.

Combining multiple challenges and responses into a single response and request maximizes security efficiency by reducing the number of extra round trips. For example, the checks for device authentication, app authenticity, and direct update can be done in a single round trip.

The Worklight client security framework automatically resends the original request with the challenge responses allowing for separation between the app business logic and the security handling aspects. Though any app request can result in a security challenge, the app business logic must not include any special processing for that security challenge. The challenge handlers are not related to the app context and can therefore focus on the security-related logic.

3.3.3 Integration with web container security

The Worklight Server is technically a web app hosted by an app server (such as WebSphere Application Server). Therefore, it is often desirable to reuse the authentication capabilities of the app server for Worklight Server and vice versa. To do this, one must understand the differences between the Worklight and the web Container authentication models.

The Java Platform, Enterprise Edition model enables only one authentication scheme for a web app, with multiple resource collections that are defined by URL patterns with authentication constraints defined by a list of role names.

The Worklight model enables protection of each resource by multiple authentication checks and the resources are not necessarily identified by the URL pattern. In some cases authentication can be triggered dynamically during the request processing.

As a result, the authentication integration between Worklight Server and the Java Platform Enterprise Edition container is implemented as a custom Worklight realm. This realm can interact with the container and obtain and set the container's authenticated principal as the Worklight user identity.

Worklight Server includes a set of login modules and authenticators for WebSphere Application Server Full Profile and WebSphere Application Server Liberty Profile that implement this integration with LTPA tokens. The integration works using the following rules:

- If the caller principal (an entity that can be authenticated) of the incoming HTTP request to the Worklight server is already set, then the container authentication was previously successful, and therefore the same principal is set as the Worklight user identity.
- If the incoming request contains an LTPA token, the login module validates it and creates the Worklight user identity.
- If the request does not contain an LTPA token, the authenticator requests the user name and password from the client. The login module validates them and creates the Worklight user identity. In addition, it creates the LTPA token, which it returns back to the client as a cookie. In this case, the authentication capabilities of WebSphere Application Server are reused by the Worklight realms in the validation and building of an LTPA token.

3.3.4 Integration with web gateways

Web gateways, such as IBM WebSphere DataPower and IBM Security Access Manager, provide user authentication, so that only authenticated requests can reach downstream apps. These downstream apps can obtain the result of the authentication performed by the gateway from a special header (for example, an LTPA token cookie.) This is an example of SSO.

When the Worklight server is protected by a web gateway, the client request is first routed through to the web gateway. The gateway requests the user credentials from the user or mobile app (perhaps through a login form), and then validates the credentials. If the validation is successful, the gateway forwards the request to the Worklight Server, along with the appropriate gateway-issued SSO token and header. This sequence implies the following requirements on the Worklight security elements:

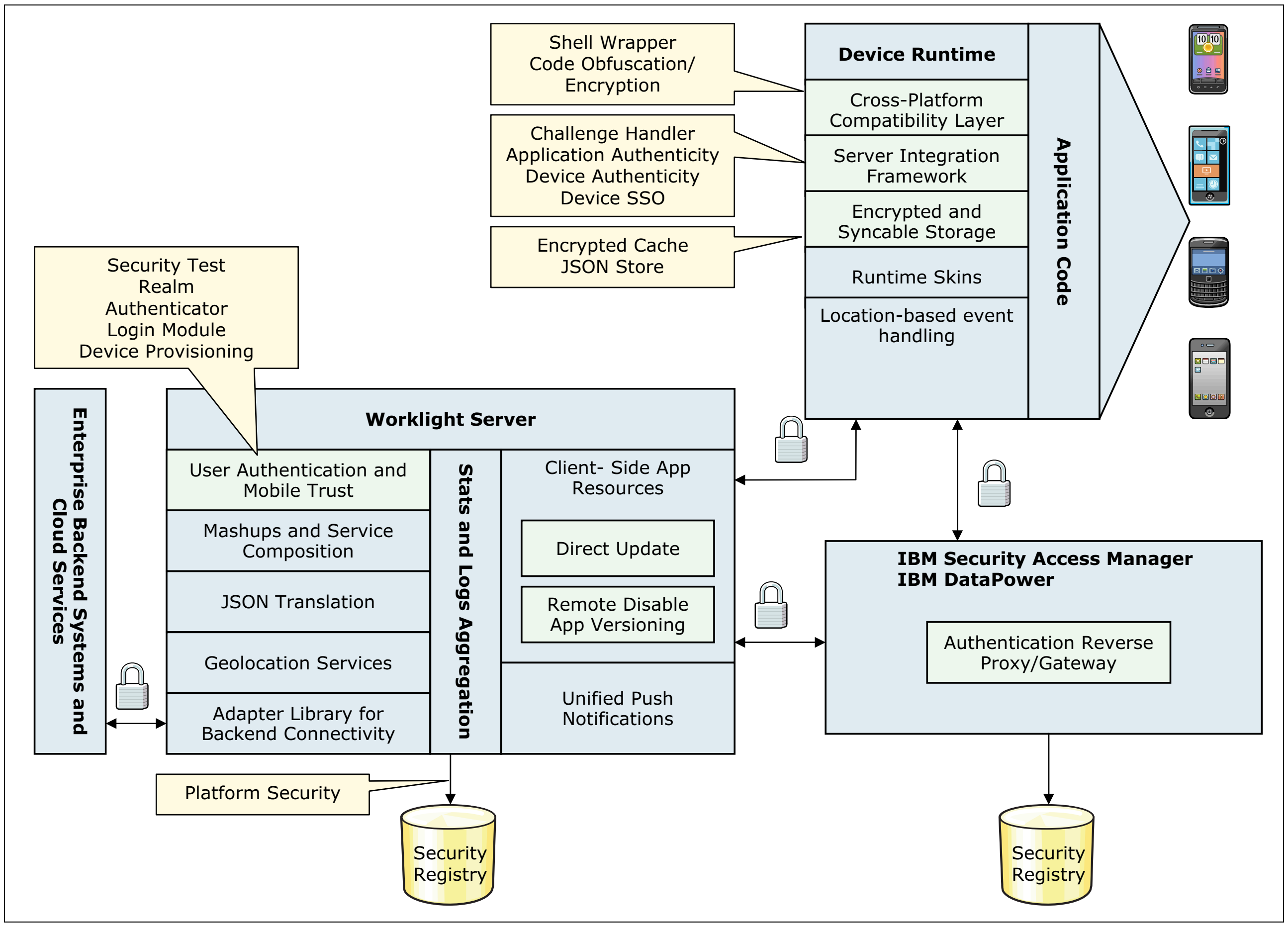
- The client-side challenge handler must be able to present the gateway's login form, collect and submit the credentials to the gateway, and also handle the login failure or success.
- The authentication configuration must include the realm that can obtain and validate the specific token or header that is provided by the gateway.
- The security test configuration must take into account that the user authentication is always done first in this case. For example, there is no point in using the Device SSO feature, because the user credentials are always requested by the gateway for each mobile app on the device.

Device SSO: Device SSO is a Worklight feature that provides the ability to authenticate with one app on the mobile device, and then access other apps on that same mobile device, without having to re-authenticate with each app if the apps share the same realm.

3.4 Conclusion

In summary, Worklight helps organizations to integrate security into the overall mobile app lifecycle related to development, delivery, and execution. Using Worklight, developers in your organization can help create and deliver security-rich mobile apps to an ever-growing number of stakeholders who are using mobile devices, such as smartphones and tablets.

With security features available to the app on the mobile device and on the Worklight server, Worklight offers a robust framework to secure your mobile apps and their data. See Figure 3-3 on page 39 for an overview of the Worklight security components.



Shell Wrapper
Code Obfuscation/
Encryption
Challenge Handler
Application Authenticity
Device Authenticity
Device SSO
Encrypted Cache
JSON Store
Device Runtime
Cross-Platform Compatibility Layer
Server Integration Framework
Encrypted and Syncable Storage
Runtime Skins
Location-based event handling
Application Code
Security Test
Realm
Authenticator
Login Module
Device Provisioning
Enterprise Backend Systems and Cloud Services
Worklight Server
User Authentication and Mobile Trust
Mashups and Service Composition
JSON Translation
Geolocation Services
Adapter Library for Backend Connectivity
Stats and Logs Aggregation
Client- Side App Resources
Direct Update
Remote Disable
App Versioning
Unified Push Notifications
IBM Security Access Manager
IBM DataPower
Authentication Reverse Proxy/Gateway
Platform Security
Security Registry
Security Registry

Figure 3-3 Worklight security components

The following chapters present a more detailed look at these Worklight security features.

4

Integrating Worklight with enterprise security

This chapter provides information about the IBM Worklight security framework, and how it addresses and supports the most common non-functional security requirements of enterprise use cases.

The Worklight security framework components are presented, described, and explored in detail, providing a deep understanding of each component, and how to configure and use those components.

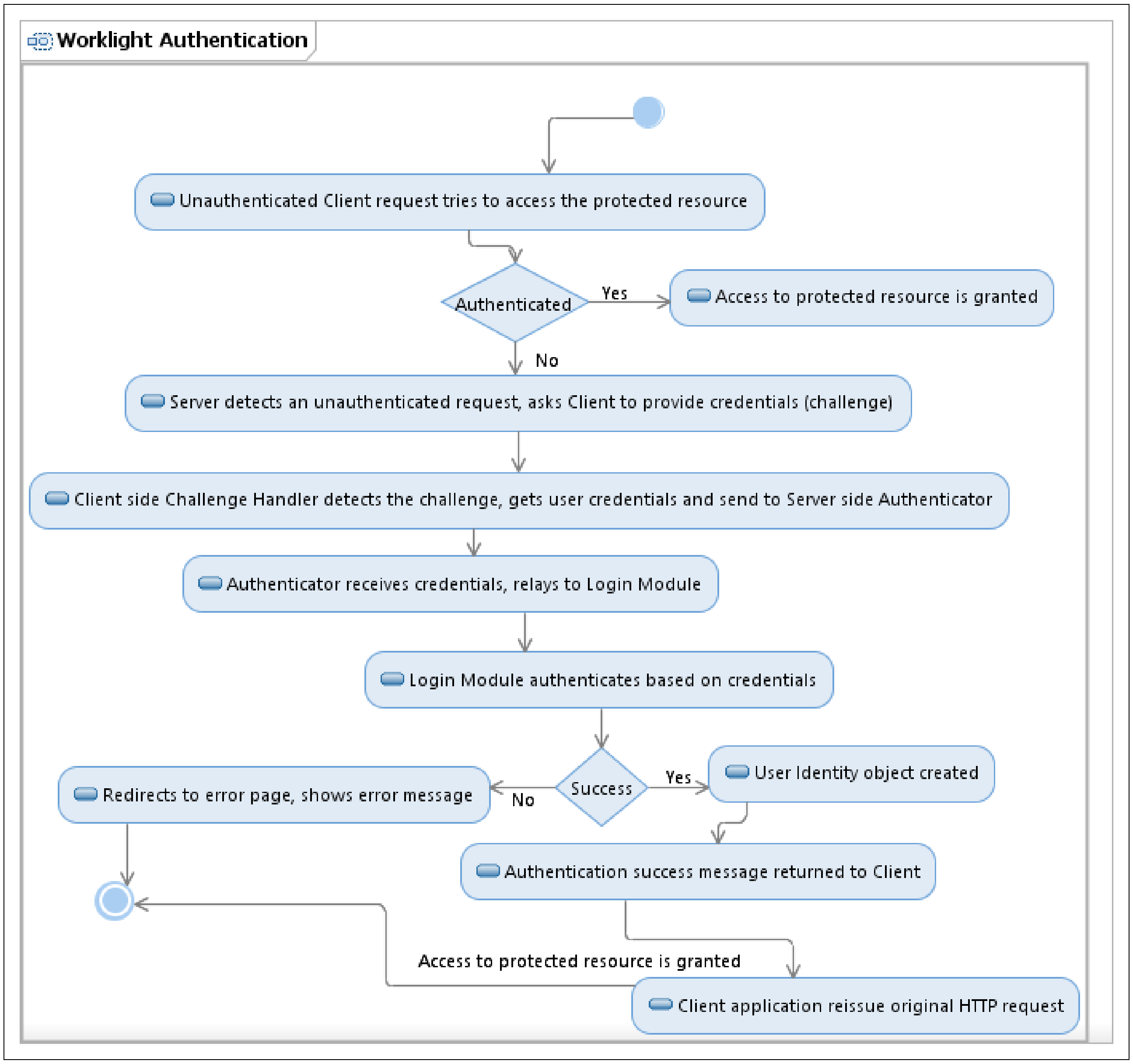
4.1 IBM Worklight security framework

The IBM Worklight security framework has a robust yet flexible set of components that address the most common security requirements, such as *authentication*, *authorization*, and *auditing* (the so-called “triple-a non-functional security requirements”).

The Worklight security framework supports controlled access to protected resources, and appropriate identification and propagation of a given user or device identity through all of the components encompassed by the framework, from the client-side to the server-side, and integrated back-end components.

The authentication is based on the provided credentials (typically a user ID or name and a password), and the process involves the retrieval and encapsulation of the identity information of a given user or device, which is then used as the primary identity regarding further requests.

Worklight authentication flow is shown in the activity diagram in Figure 4-1 on page 43.



Worklight Authentication
Unauthenticated Client request tries to access the protected resource
Authenticated
Yes
Access to protected resource is granted
No
Server detects an unauthenticated request, asks Client to provide credentials (challenge)
Client side Challenge Handler detects the challenge, gets user credentials and send to Server side Authenticator
Authenticator receives credentials, relays to Login Module
Login Module authenticates based on credentials
Success
Yes
User Identity object created
No
Redirects to error page, shows error message
Authentication success message returned to Client
Access to protected resource is granted
Client application reissue original HTTP request

Figure 4-1 Activity diagram for Worklight authentication

The authentication and authorization steps are addressed and supported by Worklight framework components:

- The authentication realms
- A client-side challenge handler
- One or multiple login modules
- Security tests that effectively protect the resources
- The target user registry used as a repository for authentication to retrieve the information used to build the user identity object

All of the aforementioned components are described in detail in this chapter.

4.1.1 Challenge handlers

A challenge handler is a client-side component that collects and sends the user credentials for authentication. It recognizes and handles the authentication challenges sent by the server-side authenticator and acts accordingly.

A full explanation of how a client-side challenge handler works is detailed in 5.1.1, "Challenge handler" on page 78.

4.1.2 Authentication configuration file

The authentication configuration file, `authenticationConfig.xml`, is in the `/server/conf` directory of a Worklight project. It is the main file used to configure the Worklight framework security components.

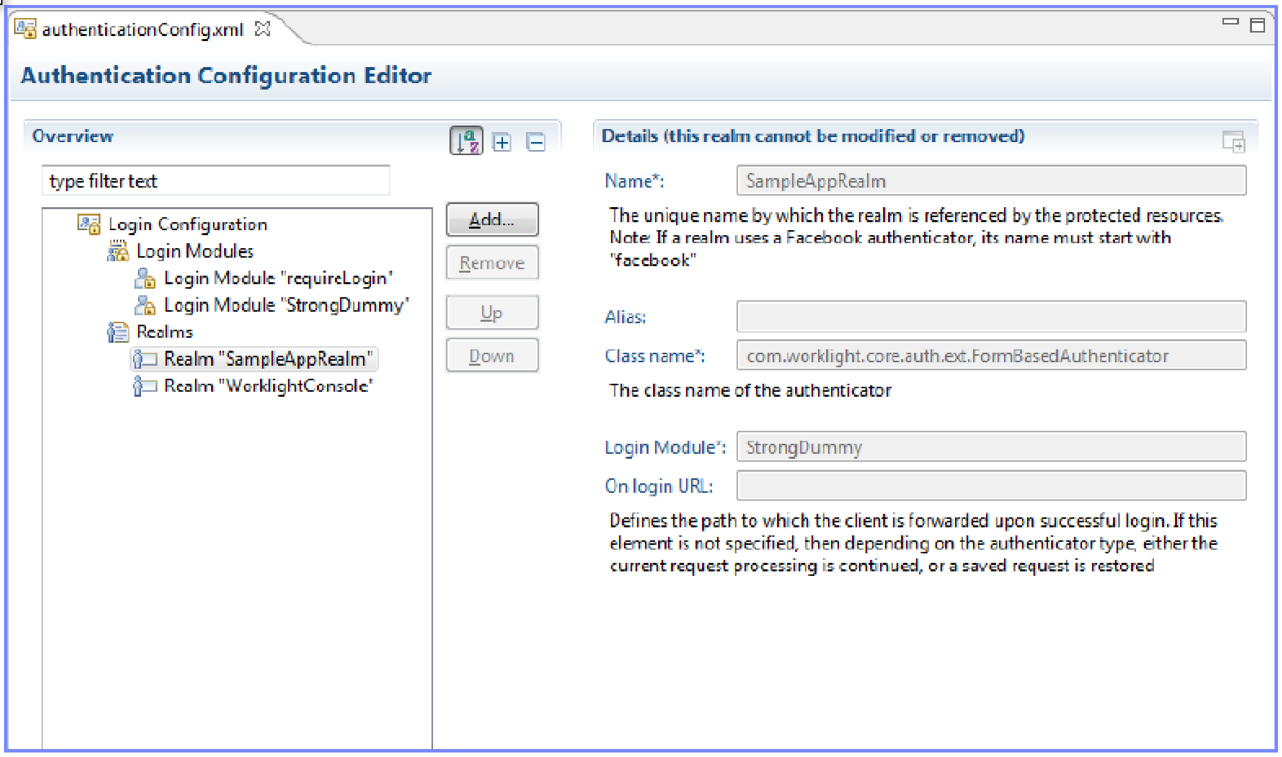
Static web resources, security tests, authentication realms, login modules, and authenticators are properly configured using the `authenticationConfig.xml` file, which consists of several Worklight security components, as shown in Example 4-1.

Example 4-1 The authenticationConfig.xml file and its XML elements

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfiguration xmlns:tns="http://www.worklight.com/auth/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <staticResources>
    <resource> … </resource>
    <resource> … </resource>
  </staticResources>
  <securityTests>
    <customSecurityTest> … </customSecurityTest>
    <customSecurityTest> … </customSecurityTest>
  </securityTests>
  <realms>
    <realm> … </realm>
    <realm> … </realm>
  </realms>
  <loginModules>
    <loginModule> … </loginModule>
    <loginModule> … </loginModule>
  </loginModules>
</tns:loginConfiguration>
```

Configure the Worklight framework security components in the `server/conf/authenticationConfig.xml` file:

- You can modify them by editing the Extensible Markup Language (XML) file directly.
- You can modify them by using the graphical user interface (GUI)-based Authentication Configuration Editor in Worklight Studio, as shown in Figure 4-2 on page 45.


authenticationConfig.xml

Authentication Configuration Editor

Overview

type filter text

- Login Configuration
  - Login Modules
    - Login Module "requireLogin"
    - Login Module "StrongDummy"
  - Realms
    - Realm "SampleAppRealm"
    - Realm "WorklightConsole"

Add... Remove Up Down

Details (this realm cannot be modified or removed)

Name*: SampleAppRealm

The unique name by which the realm is referenced by the protected resources.
Note: If a realm uses a Facebook authenticator, its name must start with "facebook"

Alias:

Class name*: com.worklight.core.auth.ext.FormBasedAuthenticator

The class name of the authenticator

Login Module*: StrongDummy

On login URL:

Defines the path to which the client is forwarded upon successful login. If this element is not specified, then depending on the authenticator type, either the current request processing is continued, or a saved request is restored

Figure 4-2 Authentication Configuration Editor

A file named `authenticationConfig.xsd` has all of the definitions for each XML element, and the possible parameters allowed when configuring them.

References to those XML element definitions are included in the forthcoming explanations, when applicable, to clarify how to configure them.

The <staticResources> and <resource> XML elements

Use the parent <staticResources> and child <resource> XML elements to restrict access to a web resource.

You must provide an ID for each resource, in addition to the `securityTest` name that is used to protect the resource.

The child <urlPatterns> element is used to inform the Uniform Resource Locator (URL) pattern that it will be matched against, as shown in Example 4-2.

Example 4-2 The <staticResources> and <resource> XML elements

```
<staticResources>
  <resource id="subscribeServlet" securityTest="SubscribeServlet">
    <urlPatterns>/subscribeSMS*</urlPatterns>
  </resource>
...
</staticResources>
```

For a complete reference about how to configure these elements, see the XML excerpt from `authenticationConfig.xsd`, as shown in Example 4-3.

Example 4-3 XML element definitions from authenticationConfig.xsd

```
<complexType name="ResourceList">
  <sequence>
    <element name="resource" type="tns:ResourceType" maxOccurs="unbounded" />
  </sequence>
</complexType>
...
<complexType name="ResourceType">
  <sequence>
    <element name="urlPatterns" type="string" />
    <element name="accessList" type="string" minOccurs="0" />
  </sequence>
  <attribute name="id" type="string" use="required" />
  <attribute name="securityTest" use="required" />
</complexType>
```

The <securityTests> and <customSecurityTest> XML elements

Use the parent <securityTests> and child <customSecurityTest> XML elements to configure the security tests that will be used to restrict access to protected resources.

You can use the predefined tests that are supplied ready for use for standard web and mobile security requirements, or you can write your own custom security tests and define the sequence in which they are run.

Example 4-4 provides sample configurations for web, mobile, and custom security tests. Security tests are explained in detail in 4.1.5, “Security tests” on page 68.

Example 4-4 Web, mobile, and custom security tests

```
<webSecurityTest name="SampleWebSecurityTest">
  <testUser realm="SampleReal"/>
</webSecurityTest>
...
<mobileSecurityTest name="sampleMobileSecurityTest">
  <testUser realm="SampleRealm"/>
</mobileSecurityTest>
...
<customSecurityTest name="SampleCustomSecurityTest">
  <test realm="SampleRealm1" step="1"/>
  <test realm="SampleRealm2" step="2"/>
  <test realm="SampleRealm2" isInternalUserID="true" step="3"/>
</customSecurityTest>
```

For a complete reference about how to configure these elements, see the XML excerpt from `authenticationConfig.xsd`, as shown in Example 4-5.

Example 4-5 XML element definitions from authenticationConfig.xsd

```
<complexType name="SecurityTestsList">
   <choice minOccurs="0" maxOccurs="unbounded">
      <element name="mobileSecurityTest" type="tns:MobileSecurityTestType" />
      <element name="webSecurityTest" type="tns:WebSecurityTestType" />
      <element name="customSecurityTest" type="tns:CustomSecurityTestType" />
   </choice>
</complexType>
...
<complexType name="SecurityTestType">
   <attribute name="name" type="tns:NonEmptyString" use="required" />
</complexType>
...
<complexType name="CustomSecurityTestType">
   <complexContent>
      <extension base="tns:SecurityTestType">
         <sequence>
            <element name="test" type="tns:TestType" minOccurs="0"
maxOccurs="unbounded" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
...
<complexType name="MobileSecurityTestType">
   <complexContent>
      <extension base="tns:SecurityTestType">
         <all>
            <element name="testUser" type="tns:TestUserType" minOccurs="0
"maxOccurs="1" />
            <element name="testDeviceId" type="tns:TestDeviceIdType" minOccurs="0"
maxOccurs="1" />
            <element name="testAppAuthenticity" type="tns:TestAppAuthenticityType
"minOccurs="0" maxOccurs="1" />
         </all>
      </extension>
   </complexContent>
</complexType>
...
<complexType name="WebSecurityTestType">
   <complexContent>
      <extension base="tns:SecurityTestType">
         <sequence>
            <element name="testUser" type="tns:TestUserType" minOccurs="1"
maxOccurs="1" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

The <realms> and <realm> XML elements

Use the parent <realms> and child <realm> XML elements to configure the authentication realms that will be used to authenticate users.

The <realm> element requires a unique name and a `loginModule` name as arguments to its parameters.

The child <className> element is used to inform the fully qualified Java class name (Java package name plus class name) for the authenticator component that is used by the target authentication realm, as shown in Example 4-6.

Example 4-6 The <realms> and <realm> XML elements

```
<realms>
  <realm name="SubscribeServlet" loginModule="headerLogin">
    <className>com.worklight.core.auth.ext.HeaderAuthenticator</className>
  </realm>
  ...
</realms>
```

For a complete reference about how to configure these elements, see the XML excerpt from `authenticationConfig.xsd`, as shown in Example 4-7.

Example 4-7 XML element definitions from authenticationConfig.xsd

```
<complexType name="RealmList">
  <sequence>
    <element name="realm" type="tns:RealmType" maxOccurs="unbounded" />
  </sequence>
</complexType>
...
<complexType name="RealmType">
   <sequence>
      <element name="className" type="string" />
      <element name="parameter" type="tns:ParameterType" minOccurs="0"
         maxOccurs="unbounded" />
      <element name="onLoginUrl" type="string" minOccurs="0" />
   </sequence>
   <attribute name="name" type="string" use="required" />
   <attribute name="alias" use="optional" type="string">
      <annotation>
         <documentation>
            Defines an alias for this realm, used by the client
            JS library.
         </documentation>
      </annotation>
   </attribute>
   <attribute name="loginModule" type="string" use="required" />
</complexType>
```

The <loginModules> and <loginModule> XML elements

Use the parent <loginModules> and the child <loginModule> XML elements to configure the Java Authorization and Authentication Service (JAAS) login modules that are responsible for verifying the user credentials, and for creating a user identity object that will hold all of the data about the authenticated user or device.

The <loginModule> element requires a unique name. The child <className> element is used to inform the fully qualified Java class name (Java package name plus the class name) for the `LoginModule` implementation class, as shown in Example 4-8 on page 49.

Example 4-8 The <loginModules> and <loginModule> XML elements

```
<loginModules>
  <loginModule name="headerLogin">
    <className>com.worklight.core.auth.ext.HeaderLoginModule</className>
    <parameter name="user-name-header" value="username"/>
  </loginModule>
  ...
</loginModules>
```

For a complete reference about how to configure them, see the XML excerpt from `authenticationConfig.xsd`, as shown in Example 4-9.

Example 4-9 XML element definitions from authenticationConfig.xsd

```
<complexType name="LoginModuleList">
   <sequence>
      <element name="loginModule" type="tns:LoginModuleType"
         maxOccurs="unbounded" />
   </sequence>
</complexType>
...
<complexType name="LoginModuleType">
   <sequence>
      <element name="className" type="string" />
      <element name="parameter" type="tns:ParameterType" minOccurs="0"
         maxOccurs="unbounded" />
   </sequence>
   <attribute name="name" type="string" use="required" />
   <attribute name="audit" type="boolean" default="false" use="optional" />
   <attribute name="ssoDeviceLoginModule" type="string" use="optional"/>
</complexType>
```

4.1.3 Authentication realms

An authentication realm defines the process used to authenticate users. The realm is always composed of one authenticator and one JAAS login module, as shown in Figure 4-3.

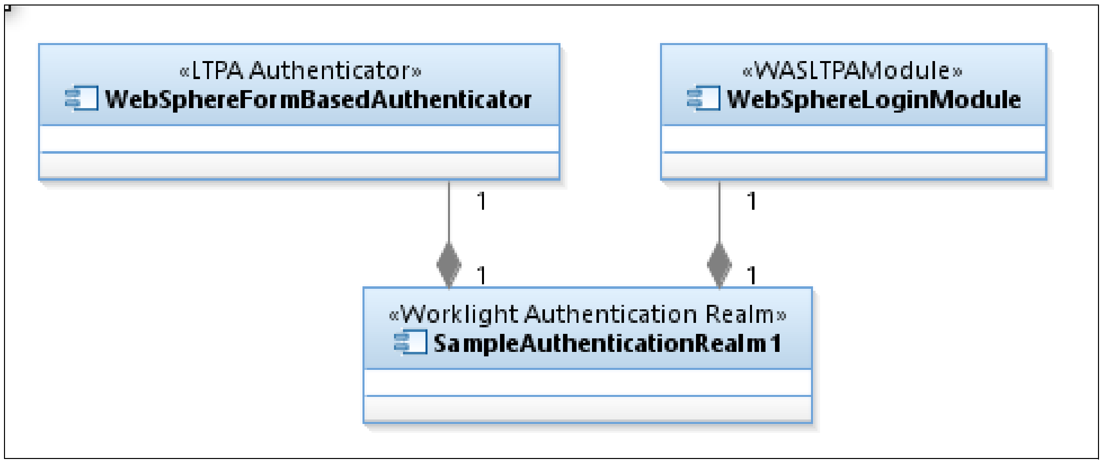

«LTPA Authenticator»
WebSphereFormBasedAuthenticator
«WASLTPAModule»
WebSphereLoginModule
1
1
1
1
«Worklight Authentication Realm»
SampleAuthenticationRealm1

Figure 4-3 Realm, authentication, and login module relationships

The authentication and the user identity object creation processes are supported by authentication realms.

Resources are protected by authentication realms, in addition to security tests, and the same realm can be used by several security tests. There is a one-to-many relationship between a given realm and the associated security tests, as shown in Figure 4-4.

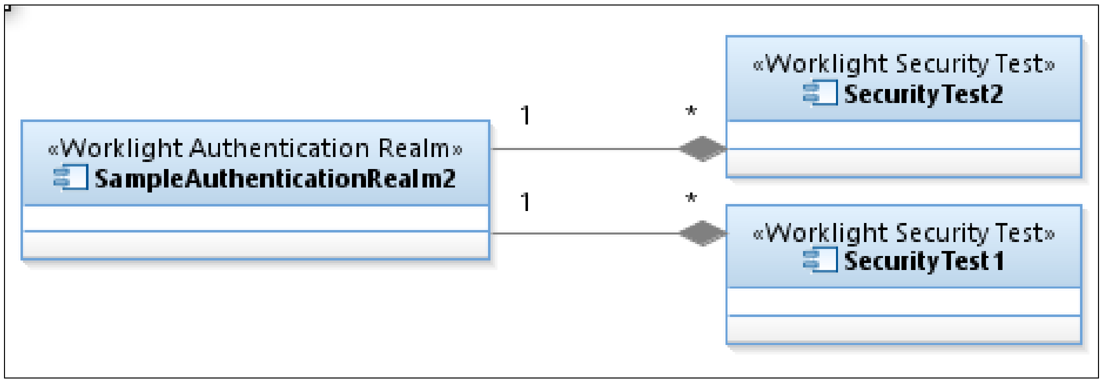

«Worklight Authentication Realm»
SampleAuthenticationRealm2
1
1
*
*
«Worklight Security Test»
SecurityTest2
«Worklight Security Test»
SecurityTest1

Figure 4-4 Realm and security tests relationships

Note that in the aforementioned relationship, the security test encompasses the realm, and also has a relationship. In these cases, the application (app) clients must undergo the authentication process that is defined for the realm only one time.

Worklight provides several predefined authentication realms ready for use, as shown in Table 4-1.

Table 4-1 Predefined realms: properties of the <test realm> XML element

Authenticator class name	Login module reference	Description
`wl_antiXSRFRealm`	`WLAntiXSRFLoginModule`	Implements anti-cross-site request forgery (XSRF) header check
`wl_deviceNoProvisioningRealm`	`WLDeviceNoProvisioningLoginModule`	Implements device authentication without provisioning
`wl_deviceAutoProvisioningRealm`	`WLDeviceAutoProvisioningLoginModule`	Implements device authentication with auto-provisioning
`wl_authenticityRealm`	`wl_authenticityLoginModule`	Implements application authenticity check

Each authentication realm that is defined in the `authenticationConfig.xml` configuration file must have a corresponding challenge handler in the client side application.

The authentication realms determine how the user credentials will be collected and validated, and how the user identity object that holds the data specific to a given user will be instantiated.

Each authentication realm defines its authentication flow, consisting of the following logic:

- What happens after the authentication process is triggered?
- What is the form of challenge that is sent to the client application?

- Which credentials are collected?
- How and when are credentials collected?
- How are credentials sent to server?
- How are credentials validated by the server?
- What is the result of the credential validation?
- What are the properties of the user identity object?

It is important to emphasize that the authentication process can be interactive or non-interactive.

A good example of *interactive authentication* is a login form that is presented when a user attempts to access a protected resource. The authentication process involves verifying the user credentials that must be provided, and requires user interaction.

Regarding *non-interactive authentication*, a good example is when you have a user cookie that the authentication process looks for in the HTTP request when that user attempts to access a protected resource. If the cookie is available, it is used to authenticate the user, requiring no further interaction (user input) by the user.

The authenticator, login module, and user identity instances are stored in a session scope that exists while the session is valid.

User identity realms and device identity realms

When using a `customSecurityTest` to protect a resource, it is possible to specify if the realm being informed is related to a user or device identity.

For a `customSecurityTest`, no predefined realms are added, so you must use the `customSecurityTest` to define your custom security requirements, and the sequence and grouping in which you want them to be processed. Security tests are explained in more detail in 4.1.5, “Security tests” on page 68. For now it is just important to understand that there is a difference between a *device identity realm* and a *user identity realm*.

To specify a realm as a user identity realm, add the `isInternalUserId` property with a value equal to `true` to the security test. The `isInternalUserID` attribute is used to inform that this realm is used for user identification for reporting and push subscription procedures.

Likewise, to specify a realm as a device identity realm, add the `isInternalDeviceID` property with a value equal to `true` to the security test. The `isInternalDeviceID` attribute is used to inform that this realm is used for device identification for reporting, push subscription, and device single sign-on (SSO) features.

There must be exactly one such realm for every security configuration that is applied to a mobile resource.

Single-realm and multiple-realm authentication

It is possible to have multiple user realms that a user or device must authenticate against. For example, a requirements scenario can exist that requires one user to authenticate against two different user registries.

If you define two user realms in a custom security test without specifying the `step` attribute, both requests for authentication are sent simultaneously in one request.

If you specify the `step` attribute following a sequence such as `one` and `two`, authentication follows this sequence. If the `SampleAuthenticationRealm1` realm is set as `step=1`, and `SampleAuthenticationRealm2` is set as `step=2`, `SampleAuthenticationRealm1` authenticates first, and a separate request to authenticate `SampleAuthenticationRealm2` is sent.

More details about authentication sequence configuration are provided in 4.1.6, “User registries” on page 70.

HTTP basic authenticator

The basic authenticator implements basic HTTP authentication as defined in HTTP Request for Comments (RFC), presenting an authentication box to the user. It is available only for web applications. Mobile apps cannot use it.

The basic authenticator fully qualified Java class name is `com.worklight.core.auth.ext.BasicAuthenticator`, and it has a mandatory parameter called `basic-realm-name` that indicates which authentication realm is to be used. An illustration of its configuration is shown in Example 4-10.

Example 4-10 HTTP basic authenticator

```
<realm name="realmForMyApp" loginModule="DatabaseLoginModule">
  <className> com.worklight.core.auth.ext.BasicAuthenticator </className>
  <parameter name="basic-realm-name" value="My App" />
</realm>
```

Form-based authenticator

The form-based authenticator presents a login form to the user that is returned in the HTTP response from the server when the client application tries to access a protected resource. Although it is most suitable for web applications, there are no restrictions when it is used in mobile apps, so you can use the form-based authentication in mobile apps as well.

The login form must contain fields named `j_username` and `j_password`, and the submit action must be `j_security_check`. These are the same field and action names used by standard Java Platform, Enterprise Edition (Java EE) security.

The configuration uses a login form page and an error page. If the login fails, the user is redirected to the error page. If an authentication error occurs, the authenticator will pass an error message to the login page. To display the error message, use the placeholder `${errorMessage}` on your login page.

The form-based authenticator fully qualified Java class name is `com.worklight.core.auth.ext.FormBasedAuthenticator`, and it requires a mandatory parameter called `login-page` that is used to inform the path to the login page being used. It is a relative path to the web application context under the `/conf` directory.

There is an optional parameter called **`auth-redirect`** that you can use in case you are planning to host your login page somewhere other than in the local web application. This parameter can be used to provide a URL that points to a remote login page. The Worklight Server will redirect to that external login page in another domain.

Note that the **`auth-redirect`** and **`login-page`** parameters are exclusive, which means you cannot use them together in the same configuration. A sample `login.html` file is always generated when creating a new Worklight project in Worklight Studio.

An illustration of a form-based authenticator configuration is shown in Example 4-11.

Example 4-11 Form-based authenticator

```
<realm name="AppAuthRealm" loginModule="DatabaseLoginModule">
  <className> com.worklight.core.auth.ext.FormBasedAuthenticator </className>
  <parameter name="login-page" value="login.html" />
</realm
```

You can download a project with a sample of form-based authentication from the following website:

http://public.dhe.ibm.com/software/mobile-solutions/worklight/docs/v600/FormBasedAuthenticationProject.zip

Persistent cookie-based authenticator

The persistent cookie-based authenticator checks to see if there is a specific cookie included in the HTTP request. If the cookie is not available in the request, the authenticator creates the cookie and sends it back in the response.

Note that this is a non-interactive authentication so, as explained earlier, it does not require any user interaction to provide the credentials for authentication.

The persistent cookie-based authenticator fully qualified Java class name is `com.worklight.core.auth.ext.PersistentCookieAuthenticator`.

The persistent cookie-based authenticator has only one option parameter, **`cookie-name`**, and as the parameter name suggests it is used to define a custom name for the persistent cookie. In case the parameter is not used, the default name will be assigned, which is `WL_PERSISTENT_COOKIE`.

An illustration of its configuration is shown in Example 4-12.

Example 4-12 Persistent cookie-based authenticator

```
<realm name="PersistentCookie" loginModule="dummy">
  <className> com.worklight.core.auth.ext.PersistentCookieAuthenticator
</className>
</realm>
```

Worklight adapter-based authenticator

The adapter authenticator enables you to develop custom authentication logic in JavaScript. Worklight adapter-based authentication is the most flexible type of authentication to implement, and it includes all of the benefits of the Worklight Server authentication framework.

When you use adapter-based authentication, the entire authentication logic (including the credential validation) can be implemented in an adapter by using plain JavaScript. Any login module can be used in the adapter-based authentication as an extra authentication layer.

Depending on the security requirements, it can be useful to support the development of additional authentication steps in a multi-step login process that cannot be implemented by just configuring another type of authenticator to achieve the additional required steps.

It is important to mention that you can use an adapter authentication to only protect adapter procedures. The adapter authenticator code performs all of the validation, and the creation of the user identity. A mandatory requirement for adapter-based authentication is the use of a `NonValidatingLoginModule`, because all authentication-related implementation is done in the adapter's code.

The adapter authenticator fully qualified Java class name is `com.worklight.integration.auth.AdapterAuthenticator`, and it has a mandatory parameter, **`login-function`**, and an optional parameter, **`logout-function`**:

- The **`login-function`** parameter specifies the name of the JavaScript function in the format <adapter-name.function-name>, which is started when the login process is triggered. The triggering can happen when either the client application explicitly invokes the WL.Client.login function, or when it tries to access a protected resource.

  The function also receives the HTTP request headers as an input parameter so that it can access the user agent and any other header available in the request. Note that this function cannot be shown as a procedure by the adapter.

- The **`logout-function`** parameter specifies the name of the JavaScript function that is started when the session is invalidated. The session termination can be triggered by having the client application start the `WL.Client.logout` function explicitly, or when the Worklight Server invalidates the session. The **`logout-function`** receives no other parameters.

  Note that this function cannot be exposed as a procedure by the adapter. An illustration of its configuration is shown in Example 4-13.

Example 4-13 Worklight adapter-based authenticator

```
<realm name="ACMERealm" loginModule="ACMELoginModule">
  <className> com.worklight.integration.auth.AdapterAuthenticator </className>
  <parameter name="login-function" value="ACMEAuthAdapter.triggerLogin" />
  <parameter name="logout-function" value="ACMEAuthAdapter.logout" />
</realm>
```

In Example 4-14, you have an instance of the `NonValidatingLoginModule` configuration. Remember that using a `NonValidatingLoginModule` class name means that no additional validation is performed by the Worklight platform. The developer takes responsibility for the credential validation within the adapter implementation logic.

Example 4-14 NonValidatingLoginModule

```
<loginModules>
  <loginModule name="ACMELoginModule">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
  </loginModule>
...
</loginModules>
```

You can download a project with a sample of adapter-based authentication from the following website:

http://public.dhe.ibm.com/software/mobile-solutions/worklight/docs/v600/AdapterBasedAuthenticationProject.zip

Custom Java-based authenticator

The `com.worklight.server.auth.api.WorklightAuthenticator` Java interface contains the methods necessary to write a custom authenticator in Java. A custom authenticator class must implement this interface and provide concrete implementation for its defined methods.

The interface contract has many methods that you must implement. The first method defined is called `init`, and its signature is shown in Example 4-15 on page 55.

Example 4-15 The init method

```
void init(java.util.Map options)
         throws com.worklight.server.auth.api.MissingConfigurationOptionExceptio
```

This method is called when the authenticator object instance is created, and then it receives the options that are specified in the realm definition in the `authenticationConfig.xml` configuration file as its argument. This assumes that you have provided the realm definition shown in Example 4-16 in the `authenticationConfig.xml` file.

Example 4-16 Options argument result from parameter definitions

```
<realm name="realmForMyApp" loginModule="DatabaseLoginModule">
  <className> com.worklight.core.auth.ext.BasicAuthenticator </className>
  <parameter name="basic-realm-name" value="My App" >
</realm>
```

The `init` method will be called with the options map that is populated with the entry:

```
"basic-realm-name"="My App"
```

Next, you have the `processRequest` that is called for each request from an unauthenticated session. Its signature is shown in Example 4-17.

Example 4-17 The processRequest method

```
com.worklight.server.auth.api.AuthenticationResult
processRequest(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response, boolean
isAccessToProtectedResource) throws java.io.IOException,
javax.servlet.ServletException
```

The **`request`** and **`response`** parameters are normal HTTP servlet objects. The `isAccessToProtectedResource` defines whether this is a request to access a protected resource.

A value of `true` means that the request is for a resource protected by this authenticator, and it cannot be ignored. A value of `false` means that the request is for a non-recognized resource, and it can be ignored by returning `AuthenticationResult.createFrom(AuthenticationStatus)` with `AuthenticationStatus.REQUEST_NOT_RECOGNIZED`.

The possible return values are shown in Table 4-2.

Table 4-2 Possible return values for com.worklight.server.auth.api.AuthenticationResult

Return value	**Outcome**
`AuthenticationStatus.SUCCESS`	All of the login data was received and processed properly.
`AuthenticationStatus.FAILURE`	The client cannot provide the login data properly.
`AuthenticationStatus.REQUEST_NOT_RECOGNIZED`	The request does not belong to the login process.
`AuthenticationStatus.CLIENT_INTERACTION_ REQUIRED`	The response was modified (for example, by HTTP forward), and can be returned to the client immediately.

The next method is `processAuthenticationFailure`, which is called if the associated login module returns a credential validation failure. Its signature is shown in Example 4-18.

Example 4-18 The processAuthenticationFailure method

```
com.worklight.server.auth.api.AuthenticationResult
processAuthenticationFailure(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response, java.lang.String errorMessage)
throws java.io.IOException, javax.servlet.ServletException
```

In Example 4-18, the **`request`** and **`response`** arguments are normal HTTP servlet objects, and the method also receives an error message as a string.

The possible return values are shown in Table 4-3.

Table 4-3 Possible return values for com.worklight.server.auth.api.AuthenticationResult

Return value	Outcome
`AuthenticationStatus.FAILURE`	A failure occurred, and there is no second chance.
`AuthenticationStatus.CLIENT_INTERACTION_ REQUIRED`	A failure occurred, and the authenticator has stared the login process again (forwarded to login-error page).

The `processRequestAlreadyAuthenticated` method is called for each request from an already authenticated session. Its signature is shown in Example 4-19.

Example 4-19 processRequestAlreadyAuthenticated method

```
com.worklight.server.auth.api.AuthenticationResult
processRequestAlreadyAuthenticated(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response) throws java.io.IOException,
javax.servlet.ServletException
```

The **`request`** and **`response`** arguments are normal HTTP servlet objects. The possible return values are shown in Table 4-4.

Table 4-4 Possible return values for com.worklight.server.auth.api.AuthenticationResult

Return value	Outcome
`AuthenticationStatus.FAILURE`	A failure occurred, and there is no second chance at all.
`AuthenticationStatus.CLIENT_INTERACTION_ REQUIRED`	A failure occurred, but the authenticator has started the login process again, forwarding to a login-error page first.

The `getAuthenticationData` method is used by the login module to get the credentials collected by an authenticator. It has no parameters, and returns the authentication data collected from the client as a Map. Its signature is shown in Example 4-20.

Example 4-20 The getAuthenticationData method

```
java.util.Map getAuthenticationData()
```

The `changeResponseOnSuccess` method is called after authentication succeeds. It is used to add data to the response after the authentication is successful. Its signature is shown in Example 4-21.

Example 4-21 The changeResponseOnSuccess method

```
boolean changeResponseOnSuccess(javax.servlet.http.HttpServletRequest request,
javax.servlet.http.HttpServletResponse response) throws java.io.IOException
```

The **`request`** and **`response`** parameters are normal HTTP servlet objects. It returns `true` only if the method actually changed the response. Lastly, you have the `getRequestToProceed` method that is already defined in this interface. This method is started after the login module successfully validates the credentials that are collected by an authenticator. Note that

Note: This method is deprecated in Worklight V5.0.0.3 and later.

The method signature is shown in Example 4-22.

Example 4-22 The getRequestToProceed deprecated method

```
javax.servlet.http.HttpServletRequest
getRequestToProceed(javax.servlet.http.HttpServletRequest currentRequest,
javax.servlet.http.HttpServletResponse currentResponse,
com.worklight.server.auth.api.UserIdentity user) throws java.io.IOException
```

You can download a project with a sample of Custom Authenticator and Login Module from the following website:

`http://public.dhe.ibm.com/software/mobile-solutions/worklight/docs/v600/CustomLoginModuleProject.zip`

HTTP header-based authenticator

The HTTP header authenticator is a non-interactive authenticator, and it must be used with the header login module. It checks HTTP request headers to see if specific headers are available to perform the authentication. Its fully qualified Java class name is `com.worklight.core.auth.ext.HeaderAuthenticator`, and it has no parameters.

An HTTP header-based authenticator configuration is shown in Example 4-23.

Example 4-23 HTTP header-based authenticator

```
<realm name="RealmHeader" loginModule="HeaderLoginModule">
  <className> com.worklight.core.auth.ext.HeaderAuthenticator </className>
</realm>
```

WebSphere LTPA authenticator

The WebSphere Lightweight Third-Party Authentication (LTPA) authenticator is used to integrate with the WebSphere Application Server LTPA authentication mechanisms. It is supported only when deploying the server side to a WebSphere Application Server instance.

To avoid unnecessary errors when deploying to other Java EE application servers, the authenticator is commented out in the `authenticationConfig.xml` file that is created by default with an empty Worklight project.

When using WebSphere LTPA-based authentication, a secure token is used in an LTPA cookie to verify the authenticated users, taking advantage of the mechanism to trust users across a secure WebSphere Application Server domain.

When you run Worklight on a WebSphere Application Server along with LTPA authentication, you must use the `WebSphereFormBasedAuthenticator` and the `WebSphereLoginModule` to authenticate to the Worklight application by the LTPA token.

The authentication process flow is shown in Figure 4-5.

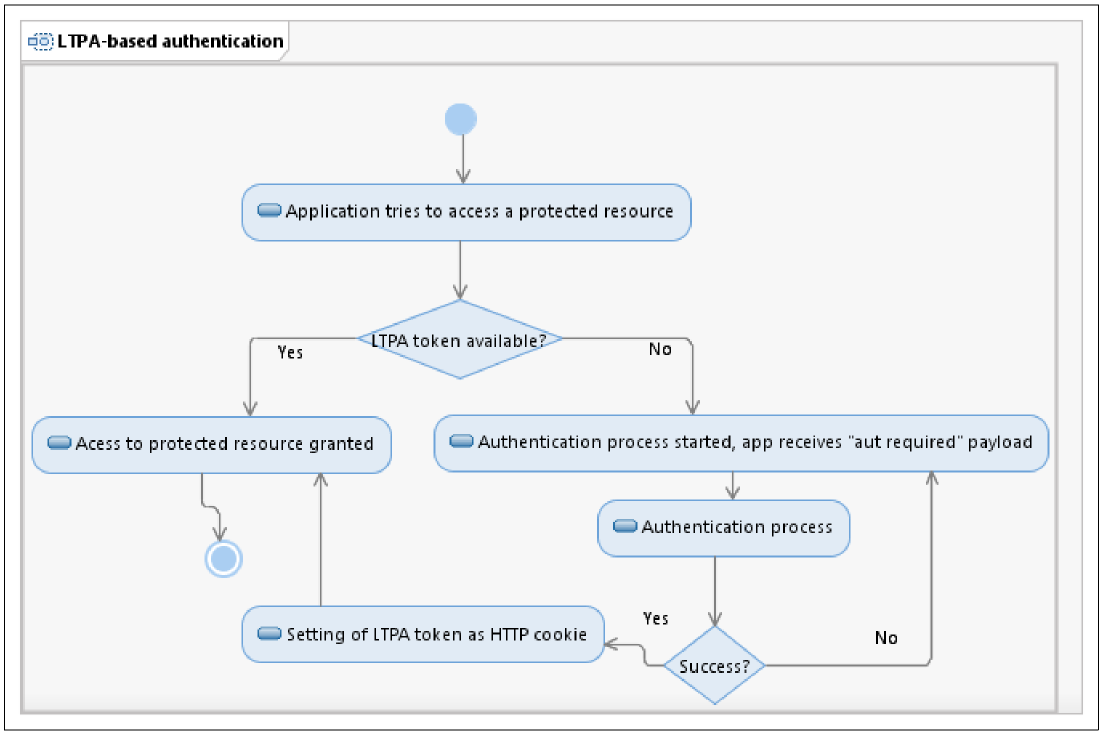

LTPA-based authentication
Application tries to access a protected resource
LTPA token available?
Yes
No
Acess to protected resource granted
Authentication process started, app receives "aut required" payload
Authentication process
Yes
No
Setting of LTPA token as HTTP cookie
Success?

Figure 4-5 LTPA authentication flow

This scheme offers the following options as an approach to support WebSphere LTPA-based authentication for Worklight apps:

- **Option 1: Authentication is enforced by WebSphere Application Server**

  This option can be used when the enterprise policy requires web archive (WAR) files to be protected on a secured WebSphere Application Server. Using this approach, you can secure the web resources in the Worklight project WAR file by specifying the resource and the user role.

  The authenticator and login module that are defined as part of this configuration scheme will authenticate the user based on the provided credentials by using the underlying WebSphere Application Server security application programming interface (API), as shown in Figure 4-6 on page 59.

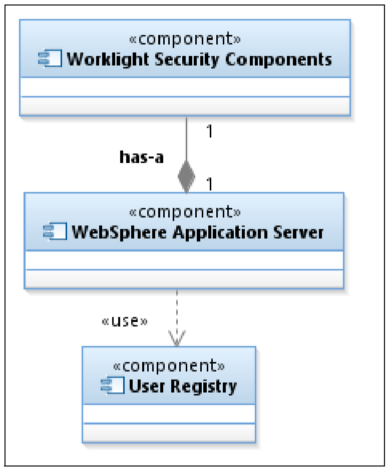


Figure 4-6 Authentication is enforced by WebSphere Application Server

- **Option 2: Worklight Server enforces authentication using the WebSphere Application Server configuration**

  This option can be used when the Worklight security configuration will handle user authentication at the Worklight platform level by using the security configuration of the underlying WebSphere Application Server, as shown in Figure 4-7.

  The Worklight project that is deployed as a WAR file on WebSphere Application Server is not secured. Therefore, the Java EE deployment descriptor `web.xml` file that is part of the WAR file does not reference any security constraints that protect the web resources.

  The authenticator and login module that are defined as part of this configuration authenticate the user based on the provided credentials, by using the underlying WebSphere Application Server security API.

  If the user provides a user name and password on initial login, this data is used to authenticate the user against the underlying registry that WebSphere Application Server is configured against. Otherwise, if a valid LTPA token is provided on subsequent access, then this LTPA credential is used.

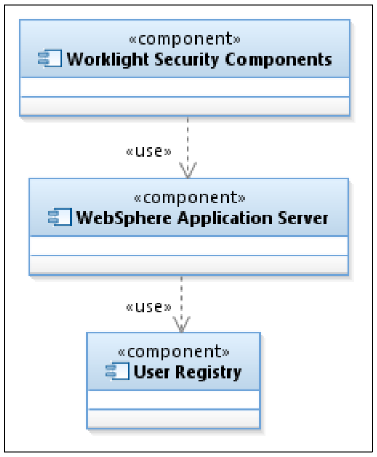


Figure 4-7 Worklight Server enforces authentication using the WebSphere Application Server configuration

Both options present benefits and have different usages, as shown in Table 4-5.

Table 4-5 Comparison between option 1 and option 2

	Option 1: WebSphere Application Server	**Option 2: Worklight Server**
Benefits	This option uses the traditional WebSphere Application Server authentication and trust model. The container enforces all security, and can reuse existing investments in securing the Java EE container by using third-party SSO products.	This option uses the traditional WebSphere Application Server authentication and trust model without the impact of modifying the Worklight project WAR file. The container enforces all security, therefore can reuse existing investments in securing the Java EE container by using third-party SSO product. The layered authentication of device, application, application instance, and user works as intended. Flexibility in configuring specific security settings that are Worklight runtime-specific without being hindered by the underlying container security.
Usage	This option is suitable for scenarios where the devices can be trusted, and access for rogue applications is restricted.	This option is suitable for scenarios where the devices or the apps on the devices *cannot* be trusted. The multi-step authenticity checking that is built into Worklight platform ensures denial of services to jailbroken devices, rogue applications, and unauthorized users.

Based on the aforementioned characteristics, if your business requirements do not mandate the use of option 1, use option 2.

The fully qualified Java class name for the LTPA authenticator is `com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator`. The LTPA authenticator has two mandatory parameters, **`login-page`** and **`error-page`**, and three optional parameters, **`cookie-domain`**, **`httponly-cookie`**, and **`cookie-name`**:

- The **`login-page`** parameter specifies the login page URL relative to the web application context.
- The **`error-page`** parameter specifies the error page URL relative to the web application context as well.
- The **`cookie-domain`** parameter is a string such as `ibm.com` that specifies the domain in which the LTPA SSO cookie applies. If this parameter is not set, no domain attribute is set on the cookie. The SSO is then restricted to the application server host name (`localhost`), and does not work with other hosts in the same domain.
- The **`httponly-cookie`** is also a string, with a value of either `true` or `false`, that specifies whether the cookie has the `HttpOnly` attribute set. This attribute is used to prevent cross-site scripting attacks.

- The **cookie-name** parameter is a string that specifies a custom name for the LTPA SSO cookie. If this parameter is not set, the default cookie name, `LtpaToken`, is used.

  An LTPA authenticator configuration is shown in Example 4-24.

Example 4-24 WebSphere LTPA authenticator

```
<realm name="WASLTPARealm" loginModule="WASLTPAModule">
  <className>com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator
   </className>
  <parameter name="login-page" value="/login.html"/>
  <parameter name="error-page" value="/loginError.html"/>
  <parameter name="cookie-domain" value="example.com"/>
  <parameter name="httponly-cookie" value="true"/>
  <parameter name="cookie-name" value="LtpaToken"/>
</realm>
```

4.1.4 Login modules

This section provides information about the various login modules.

Non-validating login module

The non-validating login module accepts any choice of user name and password passed by the authenticator. Its fully qualified Java class name is `com.worklight.core.auth.ext.NonValidatingLoginModule`. This login module has no parameters. An illustration of its configuration is shown in Example 4-25.

Example 4-25 Non-validating login module

```
<loginModule name="dummy" canBeResourceLogin="false"
isIdentityAssociationKey="true">
  <className> com.worklight.core.auth.ext.NonValidatingLoginModule </className>
</loginModule>
```

Database login module

The database login module verifies the user name and password by running a Structured Query Language (SQL) query against a relational database. Its fully qualified Java class name is `com.worklight.core.auth.ext.RDBMSLoginModule`.

The database login module has two mandatory parameters: **dsJndiName** and **principalsQuery**. The **dsJndiName** parameter specifies the Java Naming and Directory Interface (JNDI) name of the data source that defines the database.

The data retrieved from the database is used to authenticate the user, and to retrieve the user information to compose the user identity. The **principalsQuery** parameter specifies the SQL query used to select the information about the specific user.

The query has the standard structure shown in Example 4-26.

Example 4-26 Principals SQL query

```
SELECT UserID, Password, DisplayName FROM UsersTable WHERE UserID=?
```

It is important to remember that the database table used for authentication must have the columns as specified in the **SELECT** query.

UserTable is the database table name, and UserID, Password, and DisplayName are the names of the columns that contain user login names, passwords, and display names.

An illustration of a database login module configuration is shown in Example 4-27.

Example 4-27 Database login module

```
<loginModule name="MyDatabase" canBeResourceLogin="true"
isIdentityAssociationKey="true">
  <className> com.worklight.core.auth.ext.RDBMSLoginModule </className>
    <parameter name="dsJndiName" value="java:/MyDS"/>
    <parameter name="principalsQuery">
    SELECT userid, password, concat(firstName, ' ', lastName) as display_name FROM
users WHERE userid=?
    </parameter>
</loginModule>
```

LDAP login module

The Lightweight Directory Access Protocol (LDAP) login module is used to authenticate users against LDAP servers, for example Active Directory, OpenLDAP, or IBM Tivoli Directory Server.

This login module implements the `com.worklight.core.auth.ext.UserNamePasswordLoginModule` Java interface, so you must use it with an authenticator that implements the `UsernamePasswordAuthenticator` Java interface. Its fully qualified Java class name is `com.worklight.core.auth.ext.LdapLoginModule`.

It has five mandatory parameters: **`ldapProviderUrl`**, **`ldapTimeoutMs`**, **`ldapSecurityAuthentication`**, **`validationType`**, and **`ldapSecurityPrincipalPattern`**. It also has two optional parameters: **`ldapSearchFilterPattern`** and **`ldapSearchBase`**. The following parameters are mandatory:

- The **`ldapProviderUrl`** parameter specifies the Internet Protocol (IP) address or Domain Name System (DNS) name of the target LDAP server. It uses the LDAP protocol as the URL scheme. The URL starts with `ldap://`. Examples of the possible values for this parameter are `ldap://10.0.1.2` and `ldaps://10.0.1.3`.
- The **`ldapTimeoutMs`** parameter specifies the connection timeout regarding connections to the LDAP server in milliseconds. A sample value for this parameter is `2000` (Integer).
- The **`ldapSecurityAuthentication`** parameter specifies the LDAP security authentication type. The value is usually simple. However, it is important to use the correct value for this parameter, which is aligned with what the remote LDAP server is expecting to receive.

  As a leading practice, consult your LDAP administrator to obtain the correct authentication type. Possible values (String) are `none`, `simple`, and `strong`.
- The **`validationType`** parameter is a mandatory one. It specifies the type of validation to be used against the LDAP server. Possible values are `exists`, `searchPattern`, or `custom`, as detailed in Table 4-6.

Table 4-6 LDAP validation types

Value	Description
`exists`	The login module tries to establish the LDAP binding with the supplied credentials. The credential validation is successful if the binding is successfully established.

Value	Description
`searchPattern`	The login module tries to perform the `exists` validation. When the validation succeeds, the login module issues a search query to the LDAP server context according to the **`ldapSearchFilterPattern`** and **`ldapSearchBase`** parameters. The credentials validation is successful if the search query returns one or more entries.
`custom`	With this value, you can implement custom validation logic. The login module tries to do the existing validation. When the validation succeeds, the login module calls a public Boolean `doCustomValidation(LdapContext ldapCtx, String` *`username`*`)` method. To override this method, you must create a custom Java class in your Worklight project, and extend from `com.worklight.core.auth.ext.UserNamePasswordLoginModule`.

- The last mandatory parameter is the **`ldapSecurityPrincipalPattern`**. This parameter is specified depending on the target LDAP server type, because it can require security credentials that you must supply, and it can vary according to several input formats. This property is used to define the pattern to create your user name-based credentials.

  You can use the `{`*`username`*`}` as the placeholder for the user name or ID. Examples of values for this parameter are `{`*`username`*`}`, `{`*`username`*`}@myserver.com`, and the longer standard LDAP Data Interchange Format (LDIF) form, such as `CN={username}, DC=myserver, DC=com`.

The remaining parameters are optional:

- The **`ldapSearchFilterPattern`** parameter will be required only if the value of the **`validationType`** parameter is `searchPattern`. You use this parameter to define a search filter pattern that will be run when the LDAP binding is successfully established. The user credentials validation against the LDAP is considered successful when the search returns one (or more) entries as the result.

  You can use the `{`*`username`*`}` as the placeholder for this parameter. It is important to note that the syntax can change, depending on the LDAP server type that you are authenticating against.

  Examples of possible values for this parameter are `(sAMAccountName={`*`username`*`})` and the longer standard LDIF form such as `(&(objectClass=user)(sAMAccountName={`*`username`*`})(memberof=CN=Sales,OU=Groups, OU=MyCompany, DC=myserver, DC=com`.

- The last parameter is **`ldapSearchBase`**. It is required only if the **`validationType`** parameter has a value equal to `searchPattern`. You must use this parameter to specify the LDAP search base in the LDAP tree. An example of a possible value is `dc=myserver, dc=com`.

  A sample configuration is shown in Example 4-28.

Example 4-28 LDAP Login Module

```
<loginModule name="LDAPLoginModule">
  <className>com.worklight.core.auth.ext.LdapLoginModule</className>
  <parameter name="ldapProviderUrl" value="ldap://10.0.1.2"/>
  <parameter name="ldapTimeoutMs" value="2000"/>
  <parameter name="ldapSecurityAuthentication" value="simple"/>
  <parameter name="validationType" value="searchPattern"/>
  <parameter name="ldapSecurityPrincipalPattern"
value="{username}@myserver.com"/>
  <parameter name="ldapSearchFilterPattern"
value="(&(objectClass=user)(sAMAccountName={username})(memberof=CN=Sales,
   OU=Groups,OU=MyCompany,DC=myserver,DC=com))"/>
  <parameter name="ldapSearchBase" value="dc=myserver,dc=com"/>
</loginModule>
```

You can download a project with a sample showing how to use the LDAP Login Module to authenticate users with LDAP server here:

http://public.dhe.ibm.com/software/mobile-solutions/worklight/docs/v600/LDAPLoginModuleProject.zip

Header login module

The header login module validates the authentication by checking the request for specific headers. It must always be used with the HTTP header authenticator. Its fully qualified Java class name is `com.worklight.core.auth.ext.HeaderLoginModule`. It has two parameters: one mandatory parameter called **`user-name-header`**, and one optional parameter called **`display-name-header`**.

The **`user-name-header`** parameter specifies the name of the header that contains the user name. In case the request does not contain the specified header, authentication fails. The **`display-name-header`** parameter specifies a different display name other than the value obtained from the **`user-name-header`** parameter. In case this parameter is not specified, the user name is then used as the display name as well.

An example of their configuration is shown in Example 4-29.

Example 4-29 Header login module

```
<loginModule name="HeaderLoginModule" canBeResourceLogin="true"
isIdentityAssociationKey="true" audit="true">
  <className>com.worklight.core.auth.ext.HeaderLoginModule</className>
  <parameter name="user-name-header" value="userid"/>
  <parameter name="display-name-header" value="username"/>
</loginModule>
```

Single identity login module

The single identity login module is used to grant access to the Worklight Console to a single user. The credentials for this single user are defined in the `worklight.properties` configuration file.

Normally this login module is used only for testing purposes, because of its simplicity and ease of use. Its fully qualified Java class name is `com.worklight.core.auth.ext.SingleIdentityLoginModule`, and it has no additional configuration parameters.

In addition to the `authenticationConfig.xml` configurations used by all login modules, the `worklight.properties` file must also include extra configuration properties.

The `console.username` property specifies the name of the user who will be given access to the Worklight Console. The `console.password` property specifies the password of the user who will be given access to the Worklight Console. Note that the password can also be informed in an encrypted format for this property.

A sample configuration is shown in Example 4-30.

Example 4-30 Single identity login module

```
<loginModule name="Console" canBeResourceLogin="false"
isIdentityAssociationKey="false">
```

```
  <className> com.worklight.core.auth.ext.SingleIdentityLoginModule </className>
</loginModule>
```

WebSphere LTPA login module

The WebSphere LTPA login module is used to integrate with WebSphere Application Server, and to take advantage of its LTPA authentication mechanisms. This login module is only supported on WebSphere Application Server.

To avoid unnecessary errors when Worklight is run on other application servers, the login module configuration sample for this login module is commented out in the default `authenticationConfig.xml` file that is created with an empty Worklight project.

Its fully qualified Java class name is `com.worklight.core.auth.ext.WebSphereLoginModule`.

A sample configuration is shown in Example 4-31.

Example 4-31 WebSphere LTPA login module

```
<loginModule name="WASLTPAModule" canBeResourceLogin="true"
isIdentityAssociationKey="false">
  <className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
</loginModule>
```

Provisioning login modules

Provisioning is the process of obtaining a security certificate. There are three modes of the provisioning process:

- No provisioning
- Auto-provisioning
- Custom provisioning

It is important to emphasize that auto-provisioning and custom provisioning are supported only on iOS and Android devices.

Worklight provides login modules that are related to provisioning:

- There is a login module called `WLDeviceAutoProvisioningLoginModule` that must be used with an authenticator whose Java class name is `wl_deviceAutoProvisioningRealm`. They are used to implement device authentication with auto-provisioning scenarios.
- There is also a login module called `WLDeviceNoProvisioningLoginModule` that must be used with an authenticator whose Java class name is `wl_deviceNoProvisioningRealm`. They are used to implement device authentication without provisioning.

A full explanation about provisioning is beyond the scope of this publication, but more information can be found in the IBM Worklight V6.0.0 Information Center:

`http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/devref/c_mobile_device_provisioning.html`

Regarding custom provisioning, more information can be found on the following website:

`http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/devref/t_configuring_and_implementing_d.html`

There is also more information in 5.3, “Client-side device provisioning and application authenticity” on page 90.

You can download a project with an example of custom device provisioning from the following website:

http://public.dhe.ibm.com/software/mobile-solutions/worklight/docs/v600/CustomDeviceProvisioningProject.zip

Custom login module implementation

There are some security scenarios where your *authentication* and *user identity* object creation requirements are specific.

One example is to build a custom login module that enables you to use an existing mainframe deployment as the provider for the user registry that you authenticate against, and from where you need to retrieve the user data used to build the user identity object. If this target mainframe platform does not have a ready-for-use login module component, you will need to develop a custom login module to meet your non-functional security requirements.

Worklight provides a well-defined set of interfaces that you can use to build a custom login module. As of Worklight V5.0.6, the `WorkLightLoginModule` Java interface that was used to implement a custom login module has been deprecated. It has been replaced by the `WorkLightAuthLoginModule` Java interface instead, so this is the interface that your custom Java class must implement to develop a custom login module in Worklight V6.0.0 and later.

This interface inherits from the `com.worklight.server.auth.api.WorkLightLoginModuleBase` Java interface, where you have all the common JAAS method definitions as per the Java EE specification for the JAAS API, including the `abort`, `clone`, `equals`, `hashCode`, `init`, `login`, and `logout` methods.

The interface defines only one method that must be overridden, called `createIdentity`. It receives a string argument specifying the login module name, and then returns a user identity object as shown in Example 4-32.

Example 4-32 Full method signature for createIdentity method

```
com.worklight.server.auth.api.UserIdentity createIdentity(java.lang.String
loginModule)
```

This method is used to create an authenticated `UserIdentity` object after the credential validation succeeds. The fully qualified Java class name for the object is `com.worklight.server.auth.api.UserIdentity`.

It is important to emphasize that you must use this `createIdentity` method, as defined in the `WorkLightAuthLoginModule` interface as a replacement of the deprecated `createIdenity` method that is defined in the deprecated `WorkLightLoginModule` interface. Both components are deprecated as of IBM Worklight V5.0.6.

The Java class and interface relationships for a custom login implementation in Worklight V6.0.0 are shown in Figure 4-8 on page 67.

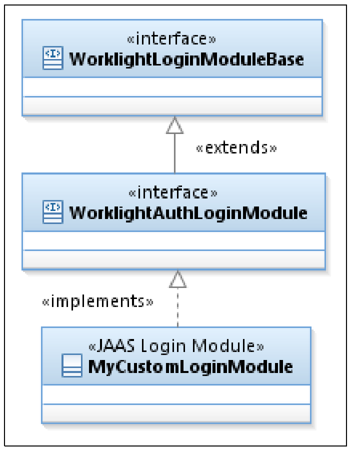


Figure 4-8 Class and interface relationships for a custom login implementation in Worklight V6.0.0

To use your custom login module, you just need to declare it using the `authenticationConfig.xml` file in the same way as for the other predefined login modules, as shown in Example 4-33.

Example 4-33 Defining a custom login module

```
<loginModules>
  <loginModule name="CustomLoginModule">
  <className>com.mypackage.MyCustomLoginModule</className>
</loginModule>
...
</loginModules>
```

Make sure the compiled Java class (`.class`, `bytecode`) is included in the server-side package to ensure that it is available in the class path of your application (WAR file), and for the target application server classloader.

For a full explanation of JAAS and how it works, see the JAAS-related specifications as defined by the Java Community Process (JCP), related Java Specification Requests (JSRs), and the JAAS references included in Figure 4-9 on page 68.

For more information about the JCP, see the following website:

http://www.jcp.org/en/home/index

You can find the *Introduction to JAAS* and *Java GSS-API* tutorials on the following website:

http://docs.oracle.com/javase/7/docs/technotes/guides/security/jgss/tutorials/index.html

You can find the *Java Authentication and Authorization Service (JAAS) Reference Guide* on the following website:

http://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html

4.1.5 Security tests

A security test defines a security configuration for a protected resource, so it is the security test that really enforces the protection of a given resource. A security test specifies one or more authentication realms, and an authentication realm can be used by any number of security tests.

A protected resource is protected by a security test, and can be protected by any number of realms. Therefore, when you define your security tests, it is also important to define the security realms that are used by the security tests, because this is a strict requirement.

Worklight has some predefined tests that comprise tests for a range of standard web and mobile security requirements. However it is a flexible mechanism, and you can also write your own custom security tests.

When a client application attempts to access a protected resource, Worklight checks whether the client is already authenticated according to all of the realms of the corresponding security test. In case the client is not authenticated yet, Worklight triggers the process of authentication for all unauthenticated realms.

The security tests are defined in the `application-descriptor.xml` configuration file by using the `securityTest="test_name"` property.

If no security test is defined, Worklight will perform only a minimal set of default platform tests.

Authentication sequence

Worklight is flexible regarding the possibility of defining the sequence in which the security tests will be performed. You can define the order in which the authentication sequence happens. For example, the request for authentication in Realm2, as shown in Figure 4-9, is issued only after the Realm1 authentication succeeds.

A resource can be protected by either of the following security tests:

- Using SecurityTest1, the user will have to authenticate in both Realm1 and Realm2, each one with its own set of rules.
- Using SecurityTest2, the user needs to authenticate in Realm3 only.

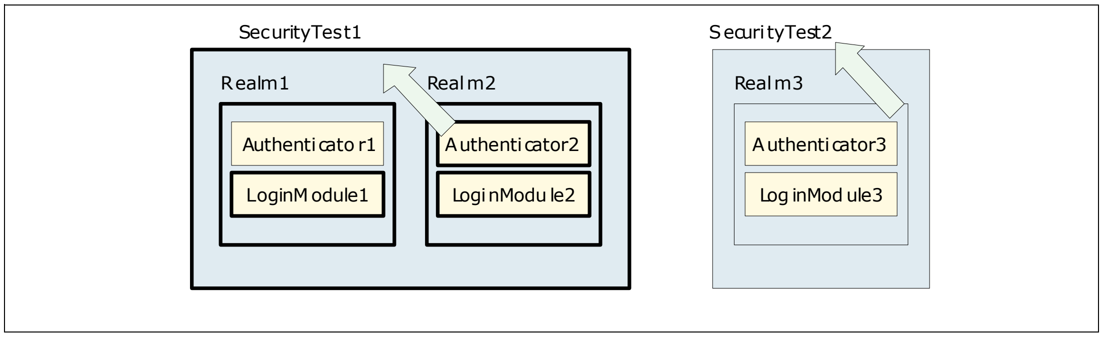


Figure 4-9 Security tests example

Realms, authenticators, and login modules are reusable components. Therefore, you can reuse their definitions as per your security requirements. The example in Figure 4-10 on page 69 shows that Realm2 is being reused.

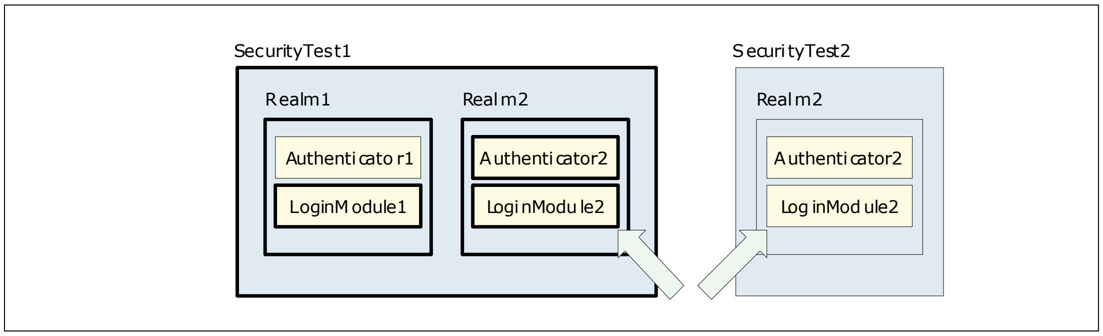


Figure 4-10 Sample security test reusing Realm2

It is important to emphasize the interplay between the security tests and the comprised realms:

- Protecting a resource with SecurityTest1 means a need to authenticate in both Realm1 and Realm2.
- Protecting a resource with SecurityTest2 means a need to authenticate in Realm2 only.

You can create elaborate and strict security test combinations and test sequences, allowing you to run several tests before access is granted to a given protected resource.

To specify the sequence in which a client must authenticate in the different realms, add the `step="n"` property to each test, where n indicates the sequence. If a sequence is not specified, all tests are done in a single step.

The webSecurityTest

The webSecurityTest is a test that is predefined to contain realms that are related to web security, so it is used to protect web applications. It must contain one `testUser` element with a realm definition for user authentication, and the identity that is obtained from this realm and is considered to be the user identity.

By default, a webSecurityTest includes protection against XSRF attacks.

A sample configuration is shown in Example 4-34.

Example 4-34 webSecurityTest configuration

```
<webSecurityTest name="SampleWebSecurityTest">
  <testUser realm="SampleReal"/>
</webSecurityTest>
```

The mobileSecurityTest

The mobileSecurityTest is a test that is predefined to contain realms that are related to mobile security, so you can use a mobileSecurityTest to protect mobile apps.

If working with user realms, a mobileSecurityTest must contain one `testUser` element with a realm definition for user authentication, and the identity that is obtained from this realm is considered to be the user identity.

On the contrary, if you are working with device realms, a mobileSecurityTest must contain one `testDevice` element with a realm definition for device authentication. The identity that is obtained from this realm is considered to be a device identity.

By default, a mobileSecurityTest includes protection against XSRF attacks, and the ability to remotely disable the mobile app from the Worklight Console.

A sample configuration is shown in Example 4-35.

Example 4-35 mobileSecurityTest configuration

```
<mobileSecurityTest name="sampleMobileSecurityTest">
  <testUser realm="SampleRealm"/>
</mobileSecurityTest>
```

The customSecurityTest

The customSecurityTest defines a custom security test with no predefined realm tests. You can use a customSecurityTest to define your own security requirements, and the sequence and grouping in which they occur. It is possible to define any number of tests within a customSecurityTest, and each test must specify one realm.

A sample configuration is shown in Example 4-36.

Example 4-36 customSecurityTest configuration

```
<customSecurityTest name="SampleCustomSecurityTest">
  <test realm="SampleRealm1" step="1"/>
  <test realm="SampleRealm2" step="2"/>
  <test realm="SampleRealm2" isInternalUserID="true" step="3"/>
</customSecurityTest>
```

A specific requirement is that when you use device auto provisioning in customSecurityTests, an authenticity realm must also be present within the tests, otherwise provisioning will not succeed.

To define a realm as a device auto provisioning realm, use the `isInternalDeviceID` attribute to specify that this realm is used for device identification for reporting, push subscriptions, and device SSO features. There must be exactly one such realm for every security configuration that is applied to a mobile resource.

Finally, to define a realm as a user identity realm, add the `isInternalUserId="true"` property to the test. The `isInternalUserID` attribute means that this realm is used for user identification for reporting and push subscriptions. There must be exactly one such realm for every security configuration that is applied to a mobile or web resource.

4.1.6 User registries

Worklight enables you to use several remote user registries as the target repository for both authentication and authorization. You can use relational databases, LDAP servers, Java-friendly property files, and many other choices. You can even develop your custom JAAS login module or adapter to support your security needs. There is also an HTTP header-based login module that enables you to get user credentials from HTTP requests.

With this design approach, Worklight ensures that any enterprise security requirements can be fully met, offering a robust yet flexible security framework that can be reused or adapted as required.

4.2 Restricting access to resources with authentication realms

As already discussed, Worklight components, such as applications, adapter procedures, and static web resources, can be protected from an unauthorized access.

The protection rules are defined by security tests that contain one or more authentication realms, and one authentication realm can be used to protect several resources. The authentication realm defines the process to be used to authenticate users or devices.

Each authentication realm consists of an authenticator and a login module on the server side. The authentication realm also requires a challenge handler component to be present on the client side.

This section explores the options supported by Worklight to protect resources from unauthorized access.

4.2.1 Protecting Worklight applications

When you protect an application, authentication will be required immediately after the application tries to connect to the Worklight server. A separate securityTest can be defined for each application environment on the `application-descriptor.xml` file. If no securityTest is defined for a specific environment, only a minimal set of default platform tests will be performed.

Several security test usages are shown in Example 4-37.

Example 4-37 Protecting applications using securityTest in the application-descriptor.xml file

```
<common securityTest="SampleWebSecurityTest"/>

<android version="1.0" securityTest="SampleMobileSecurityTest">
  <worklightSettings include="true"/>
  <pushSender key="a" senderId="b"/>
  <security>
    <encryptWebResources enabled="true"/>
    <testWebResourcesChecksum enabled="true"/>
  </security>
</android>
```

4.2.2 Protecting Worklight adapter procedures

When you protect an adapter procedure, authentication is required when this adapter procedure is started by a client application.

A separate securityTest can be defined for each adapter procedure in the adapter XML file, as shown in Example 4-38.

Example 4-38 Protecting adapter procedures using securityTest in the adapter XML file

```
<wl:adapter xmlns:wl="http://www.worklight.com/integration"
xmlns:http="http://www.worklight.com/integration/http"
xsi:schemaLocation="http://www.worklight.com/integration integration.xsd
http://www.worklight.com/integration/http http.xsd">

<displayName>DummyAdapter</displayName>
```

```
<description>DummyAdapter</description>
<connectivity>
  <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
  <protocol>http</protocol>
  <domain>rss.cnn.com</domain>
  <port>80</port>
  </connectionPolicy>
  <loadConstraints maxConcurrentConnectionsPerNode="2"/>
</connectivity>

<procedure name="getSecretData" securityTest="DummyAdapter-securityTest"/>

</wl:adapter>
```

4.2.3 Protecting static Worklight web applications

A static resource is a URL loaded from the Worklight server that is, for example, the Worklight console URL or a mobile web application.

When you protect a static resource, the Worklight server will require authentication when an attempt to request the protected URL is sent.

The static resources and their protection can be defined in the `authenticationConfig.xml` file as usual. A sample is shown in Example 4-39.

Example 4-39 Protecting static resources

```
<staticResource>
  <resource id="worklightConsole" securityTest="WorklightConsoleSecurityTest">
    <urlPatterns>/console*</urlPatterns>
  </resource>
</staticResources>
```

4.2.4 Protecting event sources

In Worklight V6.0.0, it is possible to specify and use event sources. A mobile security test is mandatory for creating event sources using the `WL.Server.createEventSource(JSON-parameter-block)` function. The security test is used to reference the user identity against the unique push token sent by the push service, which is either Apple Push Notification Service (APNS) or Google Cloud Messaging (GCM).

When you want to send a notification, you must reference the user's identity, which is looked up in the database to obtain the associated push token. If you want to subscribe to an event source, it must be authenticated against that security test.

This function creates an event source according to the parameters in the JSON parameter block. The JSON parameter block contains the two mandatory properties that are called **`name`** and **`security test`**, and five optional ones called **`onUserSubscribe`**, **`onUserUnsubscribe`**, **`onDeviceUnsubscribe`**, **`onUserChange`**, and **`poll`**.

The mandatory parameters perform the following actions:

- The **name** parameter specifies a string that contains the name of the event source.
- The **securityTest** parameter specifies the appropriate securityTest from the `authenticationConfig.xml` file to be used for this event source.

All of the other remaining parameters are optional:

- The **onUserSubscribe** parameter specifies the name of the JavaScript function in the adapter file that is called when the user subscribes to this event source for the first time, on first device subscription. The callback function receives the user subscription object as an input parameter.
- The **onUserUnsubscribe** parameter specifies the name of the JavaScript function in the adapter file that is called when the user unsubscribes from this event source for the first time, on first device subscription. The callback function receives the user subscription object as an input parameter.
- The **onDeviceUnsubscribe** parameter specifies the name of the JavaScript function that is called when the device subscription is removed by a client request, or by the cleanup task. The callback function receives the device subscription as an input parameter.
- The **onUserChange** parameter specifies the name of the JavaScript function that is called when a subscription from this device exists for a different user. The callback function receives the options sent to the createEventSource function.
- The callback function in each case must return a JSON block that must contain at least an `isSuccessful` property, indicating whether the subscription is created. The returned JSON block can contain other custom properties as well, and it is transferred back to the client application.
- Finally, the **poll** parameter specifies if the method of getting the notification data from the back-end uses the polling approach. If so you must provide a couple of additional properties, and an integer that specifies the interval in seconds between the polls and onPoll, that is the name of JavaScript function which is called on each poll.

An illustration of how to use the `WL.Server.createEventSource` function is shown in Example 4-40.

Example 4-40 WL.Server.createEventSource function usage

```
WL.Server.createEventSource({
name: 'newCoupons',
onUserSubscribe: 'subscribeUser',
onUserUnsubscribe: 'unsubscribeUser',
onUserChange: 'onUserChange',
poll : {
interval: 2,
onPoll: 'produceNotifications'
}
});
```

4.3 Configuring Worklight for LTPA authentication

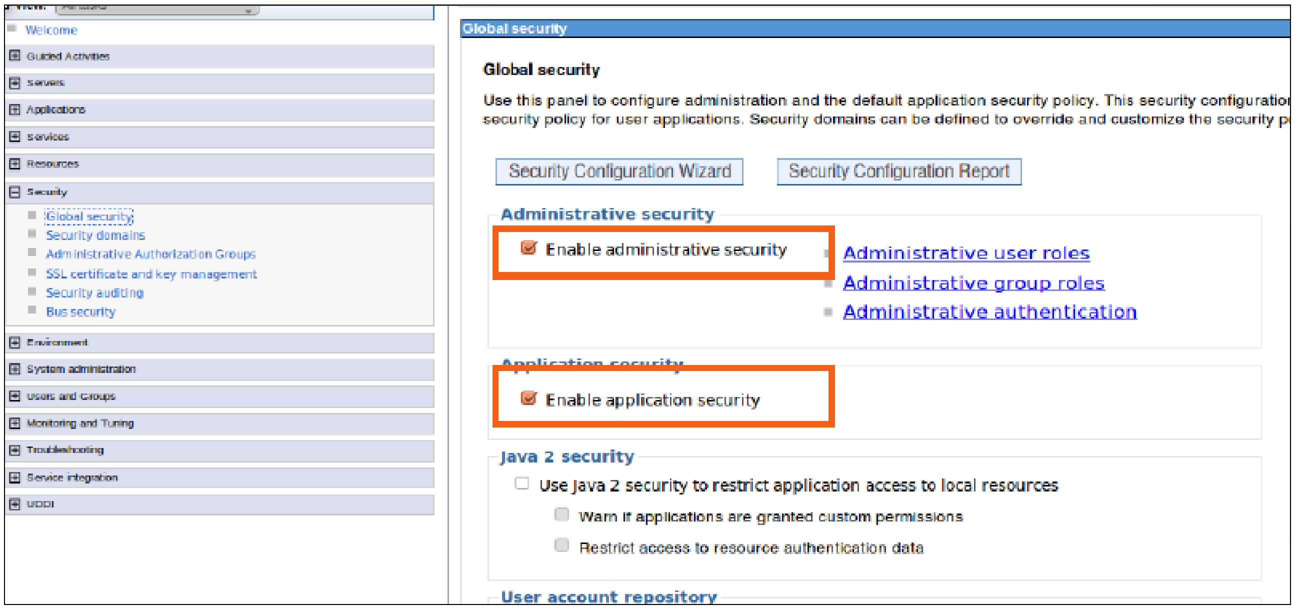
There are different approaches you can take when configuring Worklight for LTPA authentication:

- Authentication is enforced by WebSphere Application Server.

- Worklight Server enforces the authentication by relying on WebSphere Application Server configuration.

The following section describes how to enable both options, and the differences in their configurations:

1. Enable WebSphere Application Server security by selecting the appropriate options, as shown in Figure 4-11:
   a. For option one, select **Enable administrative security** and **Enable application security**.
   b. For option two, select **Enable administrative security**.



Global security

Use this panel to configure administration and the default application security policy. This security configuratio
security policy for user applications. Security domains can be defined to override and customize the security p

Administrative security

✓ Enable administrative security

Administrative user roles
Administrative group roles
Administrative authentication

Application security

✓ Enable application security

Java 2 security

☐ Use Java 2 security to restrict application access to local resources

☐ Warn if applications are granted custom permissions

☐ Restrict access to resource authentication data

User account repository

Figure 4-11 Enabling administrative security and application security in WebSphere Application Server

2. Configure `realm` and `authentication` in the `authenticationConfig.xml` file.
3. Uncomment the `realm` under the `\u201cFor websphere\u201d` comment in the `authenticationConfig.xml` file found in `{`*`WAS_HOME`*`}/profiles/{`*`yourprofile`*`}/installedApps/{`*`your node`*`}/{`*`worklight EAR`*`}/{`*`worklight WAR`*`}/WEBINF/classes/conf`. This should result in the following XML configuration, as shown in Example 4-41.

   Example 4-41 WebSphere Application Server configuration in authenticationConfig.xml

```
<!-- For websphere -->
<realm name="WASLTPARealm" loginModule="WASLTPAModule">

<className>com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator</classNa
me>
 <parameter name="login-page" value="/login.html"/>
 <parameter name="error-page" value="/loginError.html"/>
</realm>
```

   Optionally, you can include the **`cookie-domain`**, **`cookie-name`**, and **`httponly-cookie`** parameters.

4. Uncomment the `loginModule` under the `For websphere` comment in the `authenticationConfig.xml` configuration file, as shown in Example 4-42 on page 75.

Example 4-42 WebSphere Application Server LTPA login module

```
<!-- For websphere -->
<loginModule name="WASLTPAModule">
  <className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
</loginModule>
```

5. Configure the related security tests in `authenticationConfig.xml`, as shown in Example 4-43:
   a. Add webSecurityTest if you plan to develop for web environments.
   b. Add mobileSecurityTest if you plan to develop for mobile environments.

Example 4-43 WebSphere Application Server security tests

```
<securityTests>
 <webSecurityTest name="wasWebSecurity">
   <testUser realm="WASLTPARealm"/>
 </webSecurityTest>
 <mobileSecurityTest name="wasMobileSecurity">
  <testAppAuthenticity/>
  <testDeviceId provisioningType="none"/>
  <testUser realm="WASLTPARealm"/>
 </mobileSecurityTest>
</securityTests>
```

If authentication is enforced by WebSphere Application Server, there are a few additional steps required:

6. First you must create a login page. An example of a `login.html` file is shown in Example 4-44.
7. Create a new file named `login.html` and save it to the root of your WAR file at `{`*`WAS_HOME`*`}/profiles/{`*`your profile`*`}/installedApps/{`*`your node`*`}/{`*`worklightEAR`*`}/{`*`worklight WAR`*`}`.

Note: it is a standard Java EE login page file using the same standard Java EE names for `form action`, `Username`, and `Password`.

Example 4-44 Login page: login.html

```
<html>
<head></head>
<body>
  <form action="j_security_check" method="post">
    Username: <input type="text" name="j_username" size="20"><br>
    Password: <input type="password" name="j_password" size="20"><br>
    <input type="submit" value="Login">
  </form>
</body>
</html>
```

8. The next step is similar, you must create a login error page.

   Create the `loginError.html` error page and place it in the root of your WAR file at `{`*`WAS_HOME`*`}/profiles/{`*`your profile`*`}/installedApps/{`*`your node`*`}/{`*`worklightEAR`*`}/{`*`worklight WAR`*`}`.

Its content can be as simple as shown in Example 4-45.

Example 4-45 Login error page - loginError.html

```
<html>
<head></head>
<body>
  Login invalid.
</body>
</html>
```

9. Configure the standard Java EE web Deployment Descriptor, `web.xml`.
10. Locate the `web.xml` file at `{`*`WAS_HOME`*`}/profiles/{`*`yourprofile`*`}/installedApps/{`*`yournode`*`}/{`*`worklight EAR`*`}/{`*`worklightWAR`*`}/WEB-INF/web.xml`, and configure it as shown in Example 4-46. The server will now know what configuration the WAR will be expecting.

Example 4-46 web.xml configuration

```
<security-constraint id="SecurityConstraint_1">
 <web-resource-collection id="WebResourceCollection_1">
   <web-resource-name>Snoop Servlet</web-resource-name>
   <description>Protection area for Snoop Servlet.</description>
   <url-pattern>/*</url-pattern>
   <http-method>GET</http-method>
   <http-method>POST</http-method>
</web-resource-collection>

<auth-constraint id="AuthConstraint_1">
 <description>Snoop Servlet Security:+:All Authenticated users for Snoop
Servlet.</description>
 <role-name>Role 3</role-name>
</auth-constraint>

<user-data-constraint id="UserDataConstraint_1">
 <transport-guarantee>NONE</transport-guarantee>
 </user-data-constraint>
</security-constraint>

<security-role id="SecurityRole_1">
 <description>All Authenticated Users Role.</description>
 <role-name>Role 3</role-name>
</security-role>

<login-config>
 <auth-method>FORM</auth-method>
 <form-login-config>
 <form-login-page>/login.html</form-login-page>
 <form-error-page>/loginError.html</form-error-page>
 </form-login-config>
</login-config>
```

5

Applying Worklight security features

IBM Worklight provides various security features that can be used by managing and configuring the client side and server side. This chapter provides information about all of these features:

- Use Worklight to secure sensitive personal data through the use of the encrypted offline cache. You can also use JavaScript Object Notation Store (JSONStore) to maintain a local secure copy of the application (app) data.
- Worklight provides strong authentication mechanisms, and ensures that only authorized users have access to protected resources. This is supported by the challenge handler, which is located on the client side and helps to verify whether the current session is already authenticated.
- To ensure that applications can only be installed on sanctioned devices, Worklight provides device provisioning capabilities that help to obtain and determine device identity.
- Worklight can be used to ensure timely propagation of important security updates, which reduces the risk of possible security flaws being discovered. It also helps to reduce the time between the necessary updates, because the whole process is simple to learn and replicate.
- To ensure that the user is forced to update an application, or to disable a specific version, the Remote Disable feature can be used to further enhance security and application management capabilities.

5.1 Client-side authentication concepts and entities

The main client-side component involved in the authentication process is a challenge handler. This section provides an overview and usage details associated with this component.

The client-side involvement regarding device single sign-on (SSO) is also detailed in this section.

5.1.1 Challenge handler

The challenge handler is located on the client side, and is responsible for the management of the authentication process. Its main purpose is to detect and handle the authentication challenges that are sent as Worklight server responses.

A challenge handler is able to process challenges sent by the Worklight server, in addition to external authentication services, such as security gateways and proxies.

Worklight Server checks all of the appropriate realms associated with a protected resource for which a client request has been issued. It is possible that several realms can send challenges, which are then composed into a single response and sent back to the client.

The challenge handler is responsible for collecting the required credentials after it has received an authentication challenge from the server. It will then resend the original request with all of the challenge responses.

If an authenticator on the server side is satisfied with the extracted responses, the authenticator will report it as a success and call a login module. If the credentials are validated by the login module, and if validation succeeds, the realm is authenticated. This indicates the final stages of the authentication flow, and the challenge handler sends a notification to the Worklight framework indicating success or failure.

If the realm verification fails, the associated authenticator will send the same challenge back to the client and repeat the cycle.

The challenge handler is highly customizable, and Worklight enables the creation of the challenge handler with a set of predefined methods associated with handling the challenge and submitting credentials to the server.

To ensure security and efficiency, Worklight combines multiple challenges and responses into a single message, which reduces the necessary interaction between the client and server, avoiding unnecessary network round trips. Worklight is capable of grouping the checks for direct update, device authentication, and application authenticity into a single round trip.

A more in-depth overview of the server-side authentication concepts and entities can be found in 4.1, “IBM Worklight security framework” on page 42.

Authentication concepts

The challenge handler is an important client-side component responsible for managing requests associated with access to protected resources. Figure 5-1 on page 79 clearly indicates the authentication process and the role of the challenge handler.

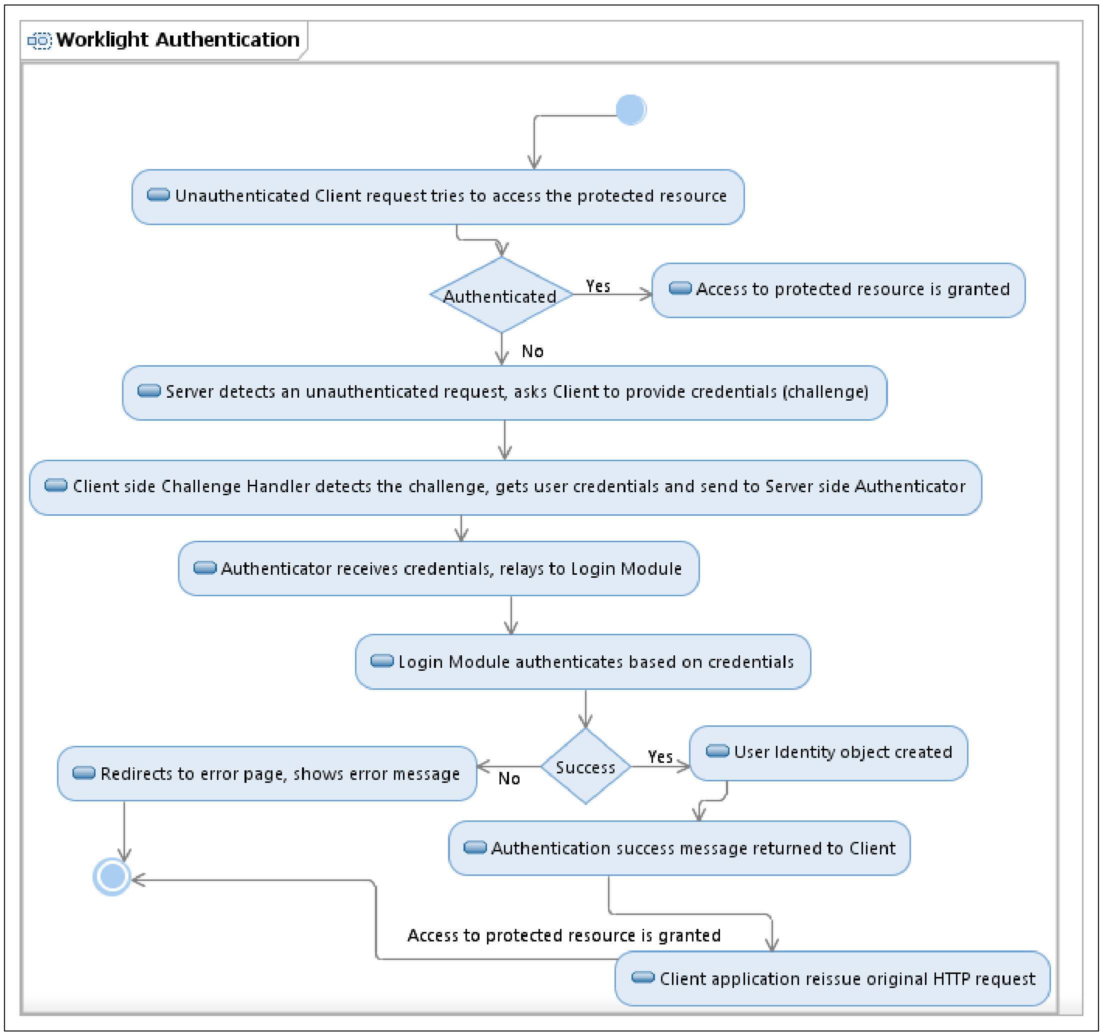

Worklight Authentication
Unauthenticated Client request tries to access the protected resource
Authenticated
Yes
Access to protected resource is granted
No
Server detects an unauthenticated request, asks Client to provide credentials (challenge)
Client side Challenge Handler detects the challenge, gets user credentials and send to Server side Authenticator
Authenticator receives credentials, relays to Login Module
Login Module authenticates based on credentials
Redirects to error page, shows error message
No
Success
Yes
User Identity object created
Authentication success message returned to Client
Access to protected resource is granted
Client application reissue original HTTP request

Figure 5-1 Activity diagram for Worklight authentication

Challenge handler API

Worklight enables the creation of custom challenge handlers for hybrid (JavaScript) and native applications. This provides the user with the ability to customize the authentication process based on the application requirements.

For a detailed list of all of the available challenge handler application programming interfaces (APIs) for native Android and iOS applications, see the IBM Worklight V6.0.0 Information Center sections on Java client-side API for native Android apps and Objective-C client-side API for native iOS apps:

`http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/apiref/c_client_api.html`

For an implementation of a challenge handler, review “Worklight client challenge handler” on page 130.

5.1.2 Device single sign-on

Device SSO provides the user with the ability to access multiple protected resources (applications and adapter procedures) in different security domains by only authenticating one time. For example, a user can access multiple applications installed on a device by authenticating one time through an SSO login module that is used by the applications. It is supported on Android and iOS devices.

The SSO login module provides the user with access to all resources that are using the same login module. The user must first successfully log in through the SSO login module to ensure only authorized users have access. This concept has been embraced by many businesses looking to provide its users with the best experience possible.

If requests relating to resources protected by the login module are still being issued within the timeout period, the authentication state will remain active and access is granted. The timeout period is usually the same as the session timeout period.

Device SSO requirements

To enable SSO capabilities, device authentication is also required. The device authentication realm must be included within the security test that is protecting the resource that will be protected with SSO. The required changes are reflected in the `authenticationConfig.xml` file.

Enable device SSO

To enable SSO, you must modify the mobile security test or custom security test based on your requirements. A common identifier must also be assigned for each application that is going to incorporate device SSO:

1. For Android, assign the same `sharedUserId` for each application by editing the `application-descriptor.xml` file, as shown in Example 5-1. The sample value is `test.sso`. For iOS you must use the same AppID prefix for each application that is accessed by device SSO. The Apple Developer Provisioning Portal is used to create a new AppID, and an AppID prefix is defined during this process.

Example 5-1 Assigning a shared user ID

```
<android version="1.0" sharedUserId="test.sso">
   <worklightSettings include="true"/>
      <security>
         ...
      </security>
</android>
```

2. Modify the `test` element in the `authenticationConfig.xml` file.
3. If you are using the predefined `mobileSecurityTest` element, to enable SSO you have to provide an additional value (`sso="true"`), as shown in Example 5-2.

Example 5-2 Enable SSO for mobile security test

```
<mobileSecurityTest name="mobileTests">
  <testDeviceId provisioningType="none"/>
  <testUser realm="mobileUserRealm" sso="true"/>
</mobileSecurityTest>
```

4. If you are using a custom security test, you must include the `ssoDeviceLoginModule` property for a user login module, as shown in Example 5-3 on page 81. The example provides sample login modules that have provisioning enabled and disabled.

Example 5-3 SSO login modules

```
<!-- For enabling SSO with no-provisioning device authentication -->
<loginModule name="MySSO"
ssoDeviceLoginModule="WLDeviceNoProvisioningLoginModule">
   <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>

<!-- For enabling SSO with auto-provisioning device authentication -->
<loginModule name="MySSO"
ssoDeviceLoginModule="WLDeviceAutoProvisioningLoginModule">
   <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>
```

5. When using a custom security test, you are also required to configure the device authentication realm at least one step before the user authentication realm to ensure correct SSO functionality (see Example 5-4).
6. In Example 5-4, non-provisioning device authentication is used. The process of obtaining and attaching a certificate to the device identity is referred to as device provisioning. Device provisioning and identity are explained in 5.3.1, “Device provisioning” on page 90.

Example 5-4 AuthenticationConfig.xml file for a custom security test.

```
<securityTests>
   <customSecurityTest name="DummyAdapter">
      <test realm="wl_antiXSRFRealm" step="1"/>
      <test realm="wl_authenticityRealm" step="1"/>
      <test realm="wl_remoteDisableRealm" step="1"/>
      <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true"
step="1"/>
      <test realm="SampleAppRealm" isInternalUserID="true" step="2"/>
   </customSecurityTest>
</securityTests>
```

5.2 Encrypted offline cache and JSONStore

Worklight provides two methods for on-device storage that facilitate the encryption of user data, provide security, and provide enhanced data management capabilities:

- Encrypted offline cache (EOC)
- JSONStore

5.2.1 EOC overview

Worklight provides EOC that is based on the Hypertext Markup Language 5 (HTML5) local storage technology. There is a limit on the amount of information that can be stored within HTML5 local storage, which is 5 megabytes (MB). If that limit is exceeded, the behavior can be unpredictable, and possibly lead to data loss.

By increasing cache size, the performance will degrade due to increased processing times. You must consider the limitations of HTML storage before deciding to integrate EOC. The EOC limitations and features are shown in Table 5-2 on page 83.

EOC is available for all technologies that support HTML5 local storage, which includes mobile, desktop, and web environments. Before developing with Worklight, you must consider mobile browser support for HTML5 storage, as shown in Table 5-1.

Table 5-1 HTML5 web storage support for iOS, Android, and BlackBerry

Versions	iOS Safari	Android Browser	BlackBerry Browser
Previous	5.0 - 5.1	4.1	7.0
Current	6.0 - 6.1	4.2	10.0
Next	🟧	🟧	🟧

🟩 Supported Versions	🟧 No information available

Hybrid applications: If you are developing Worklight hybrid applications targeted for the Android and iOS platforms, use JSONStore instead of EOC. Apple cannot guarantee that HTML5 local storage is going to be persistent in future iOS versions.

Data encryption is achieved through the use of a user-provided password and a server-retrieved randomly generated token. The data is stored in the key-value pairs, and is encrypted using a 256-bit encryption key. The key itself is encrypted using a different 256-bit encryption key generated from a user-supplied password using the Public Key Cryptography Standards #5 (PKCS#5) Password-Based Key Derivation Function 2 (PBKDF2) function.

This type of encryption enables secure on-device data storage. The HTML5 local storage approach helps to ensure strong encryption standards, and facilitates data retrieval during application start and use.

5.2.2 EOC APIs

To create, open, manipulate, and delete EOC, you must use the following APIs:

WL.EncryptedCache.open	Creates or opens an existing EOC.
WL.EncryptedCache.write	Stores a key-value pair within the EOC.
WL.EncryptedCache.read	Based on the provided key, decrypts the value.
WL.EncryptedCache.remove	Removes a key-value pair from the EOC.
WL.EncryptedCache.close	Closes the EOC.
WL.EncryptedCache.destroy	Deletes the EOC.

To create a new EOC, an application must be able to connect to the Worklight server. A random number, which is used in the encryption process, is obtained from the Worklight server. When a new EOC is created, the connection is no longer required, and the EOC can be accessed without it.

For more information, see the section about EOC APIs in the IBM Worklight V6.0.0 Information Center:

http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/apiref/r_encrypted_offline_cache.html

5.2.3 JSONStore overview

JSONStore provides a persistent storage of JSON documents. It provides different types of data encryption, such as 256-bit Advanced Encryption Standard (AES) and PBKDF2. Documents within JSONStore can be accessed within an application even when the device is offline and has no access to the Worklight server.

JSONStore supports the management of multiple users by providing password protection in case there is more than one user sharing a single device.

JSONStore provides the following functionality:

- ▶ JavaScript API for data manipulation
- ▶ File-based storage that is persistent
- ▶ Security and confidentiality by using AES256 encryption
- ▶ Integration with Worklight adapters
- ▶ Worklight Studio wizard to aid in the creation and integration with JSONStore

A single JSONStore can consist of multiple collections, which can contain many documents. Worklight also enables multiple stores within a single application. Figure 5-2 highlights the flow between the device and server when using JSONStore within the Worklight application

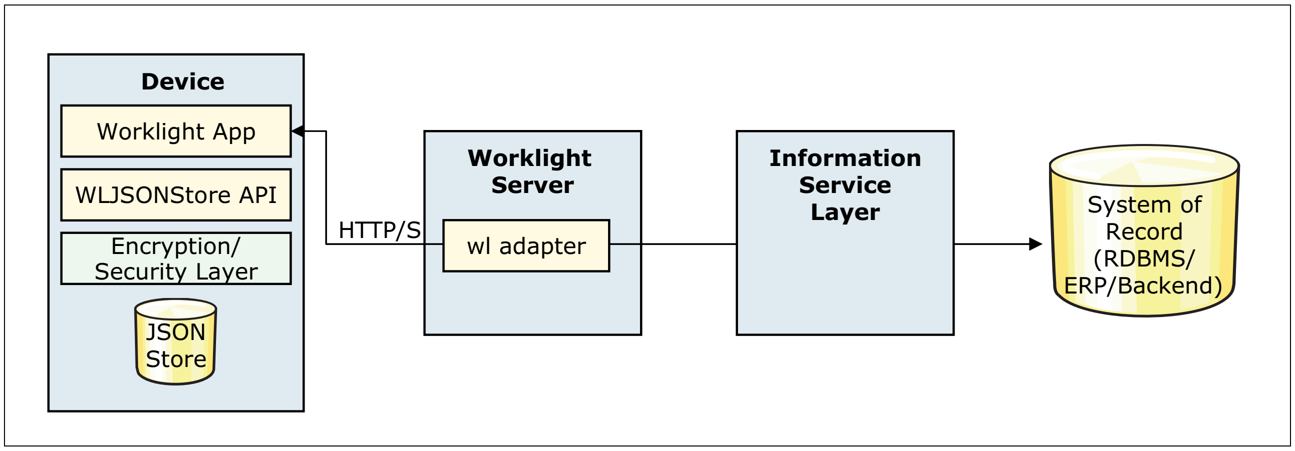

Device
Worklight App
WLJSONStore API
Encryption/ Security Layer
JSON Store
HTTP/S
Worklight Server
wl adapter
Information Service Layer
System of Record (RDBMS/ ERP/Backend)

Figure 5-2 JSONStore flow

Comparison between JSONStore and EOC

JSONStore is a JavaScript API that enables data storage inside hybrid Worklight applications. JSONStore is similar to EOC, and the information in Table 5-2 helps to differentiate between the two on-device storage technologies.

Table 5-2 Comparison of JSONStore and EOC

	JSONStore	EOC
Type of storage	JSON document	HTML5 local storage
Android support	Full support	Full support
iOS Support	Full support	Full support
Web	Only development support	Full support
Data encryption	AES256, PBKDF2, and Federal Information Processing Standard (FIPS) 140-2	AES256 and PBKDF2

	JSONStore	EOC
Storage limit	Limited by device space only	~5 MB
Reliability	Full support	Possible data loss
Adapter integration	Full support	No support
Multi-user capabilities	Full support	Only single-user support
Indexing	Full support	No support

Both JSONStore and EOC make use of the same encryption and security functions, with additional support for FIPS provided with JSONStore.

EOC is supported as a cross-platform data storage technology, with no major technical updates planned in the future. It is highly suggested to use JSONStore as an alternative to EOC.

Enable JSONStore

JSONStore can be enabled by modifying the `application-descriptor.xml` file.

Mobile Browser Simulator feature: It is important to note that JSONStore is not supported in the Worklight-supplied Mobile Browser Simulator. The Android emulator can be used to test JSONStore capabilities.

To use JSONStore capabilities, you must perform the following actions:

1. Open the `application-descriptor.xml` file.
2. Select the Design Tab.
3. Highlight **Features** and click **Add**.
4. The Add Item dialog opens:
   a. Select **JSONStore**.
   b. Click **OK**, as shown in Figure 5-3 on page 85.

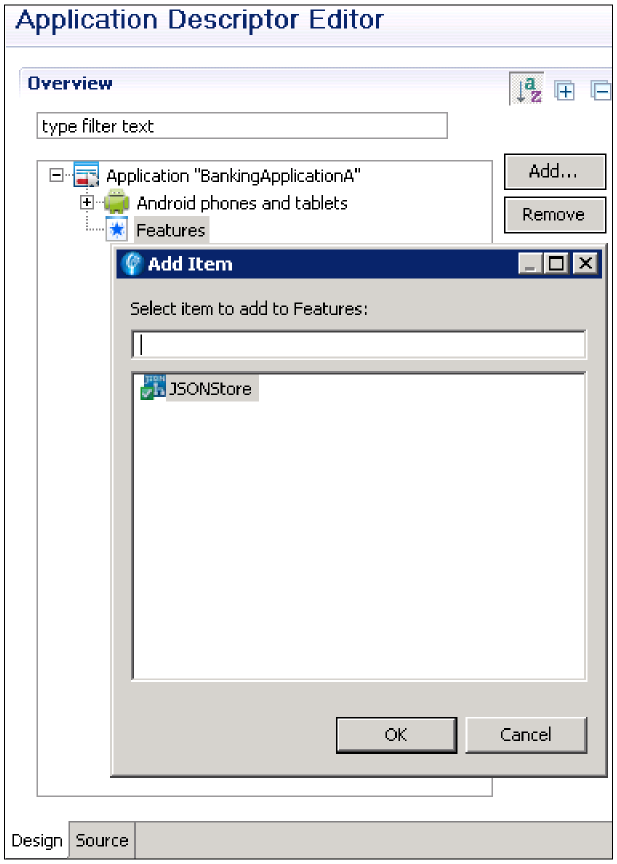

Application Descriptor Editor
Overview
type filter text
Application "BankingApplicationA"
Android phones and tablets
Features
Add...
Remove
Add Item
Select item to add to Features:
JSONStore
OK
Cancel
Design
Source

Figure 5-3 Enable JSONStore on Android

5. Verify that the new features have been added to the `application-descriptor.xml` file, as shown in Example 5-5.

Example 5-5 Enable JSONStore inside application-descriptor.xml.

```
<features>
   <JSONStore/>
</features>
```

Documents, collections, and a store

JSONStore uses the same database objects as Not only Structured Query Language (NoSQL) databases: documents and collections.

For JSONStore, a *document* is the basic building block. It is a JavaScript object with an associated ID and JSON data. A *collection* is a grouping of related documents. A collection also stores a unique integer ID that is automatically generated by a document. A file that contains one or more collections is referred to as a *store*.

Search fields and queries

A *search field* is a JavaScript object with a key and a data type. Search fields are indexed to maximize lookup times. It is not possible to index the object itself, but only the keys inside that object. Keys that are indexed but are not part of the JSON data are known as additional search fields.

A sample search field is shown in Example 5-6.

Example 5-6 Search field.

```
var searchField = {name: 'string value', age: 'integer value'};
```

A JavaScript object that uses search fields to look for documents is referred to as a *query*.

Behavior

JSONStore behaves in the following ways:

- JSONStore data cannot be accessed from the native code, therefore it is only available in hybrid environments.
- JSONStore does not impose a limit on data storage size, which is only controlled by the device memory constraints. Cordova provides APIs to check the size of the store, if needed.
- JSONStore data is persistent, unless the application is removed or the JSONStore APIs are used.
- An error will be returned if the collection name begins with a digit or symbol.

Performance

To improve performance, follow these suggestions:

- Improve memory and data retrieval performance:
  - Only get a subset of documents.
  - Implement pagination when calling `WL.JSONStore.find`.
- Avoid using fuzzy searching that is capable of returning partial results.
- Use short key names to improve the serialization and deserialization process.
- Consider performing retrieval calls using small groups of documents, because adding and loading data from an adapter can cause memory issues.
- Encryption naturally adds additional processing demands to JSONStore operations, and you should consider if it is a defined non-functional requirement before securing the store.

Support

To debug an application that enables JSONStore, ensure that you are using a supported production environment that meets the following minimum requirements:

- Android V2.3 or later ARM/x86 device and emulator
- iOS V5.0 or later device and simulator
- No support for the Mobile Browser Simulator feature

On Android-supported environments, to verify the created files, you can navigate to the JSONStore data folder via the Dalvik Debug Monitor Server (DDMS) perspective:

1. Open the **DDMS** perspective inside Worklight studio.
2. Select the **File Explorer** tab.
3. Navigate to `data/data/com.[application-name]/database/wljsonstore/`.

5.2.4 The JSONStore API

This section covers some of the features that are provided by JSONStore, and the implementation details regarding the major functions.

Encrypt store

To initialize one or more JSONStore collections, use `WL.JSONStore.init`. It provides additional functionality:

- Encrypting data
- Multiple user support
- Integration with Worklight adapters

If you initialize a JSONStore collection with a password for the first time, JSONStore will need a random token to encrypt the data inside. The token can either be obtained from the client, or from the Worklight server.

It is possible to enforce data encryption locally, without the need to connect to the server, as shown in Example 5-7. In order for the client to encrypt the data offline, set `localKeyGen = true`. This method is less secure, but it does not require connectivity.

It is also possible to generate the random token on the server side, which can be sent to the client with an Asynchronous JavaScript and XML (Ajax) call. This method provides more security but requires connectivity.

A 256-bit AES key that is strengthened with PBKDF2 is used to encrypt the store.

Example 5-7 Random token generation options and initialization

```
//Optional options object
var options = {};

//Optional local key generation flag
options.localKeyGen = boolean value;

WL.JSONStore.init(collections, options)

.then(function () {
    //handle success
})
```

Search

To query the location of documents inside a collection, you must use `WL.JSONStore.find`.

There are additional APIs available for finding all of the documents, and providing the ability to search by a document's unique ID:

- WL.JSONStore.findAll
- WL.JSONStore.findById

To improve performance, you can set a limit for the maximum number of returned results. To configure the search behavior, you can specify two different searching mechanisms: fuzzy or exact. With fuzzy search, the queries are capable of returning partial results. With exact search, the matches must be exact but not case-sensitive.

Fuzzy searching is the default behavior. Example 5-8 shows how to use fuzzy search with a maximum limit of five results returned. Change the value of `{exact: true}` to enable exact searching.

Example 5-8 Search options

```
var options = {
    exact: false,
    limit: 5 //returns a maximum of 5 documents
};
```

Multi-user support

With Worklight, you have the ability to create multiple stores. The stores can contain different collections, and they all can be stored within a single Worklight application.

JSONStore provides the capability to add **username** and **password** parameters to ensure that separate stores are encrypted with different passwords, as shown in Example 5-9. This feature enables users to share their physical devices, and to feel assured that their data will not be compromised.

Example 5-9 Multiple user support

```
var options = {};

//Optional username
options.username = 'username';

//Optional password
options.password = 'password';

WL.JSONStore.init(collections, options)

.then(function () {
    //handle success
})
```

See the section about WL.JSONStore APIs in the IBM Worklight V6.0.0 Information Center for additional information:

http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/devref/c_wl_jsonstore_core_api.html

5.2.5 JSONStore integration with Worklight adapters

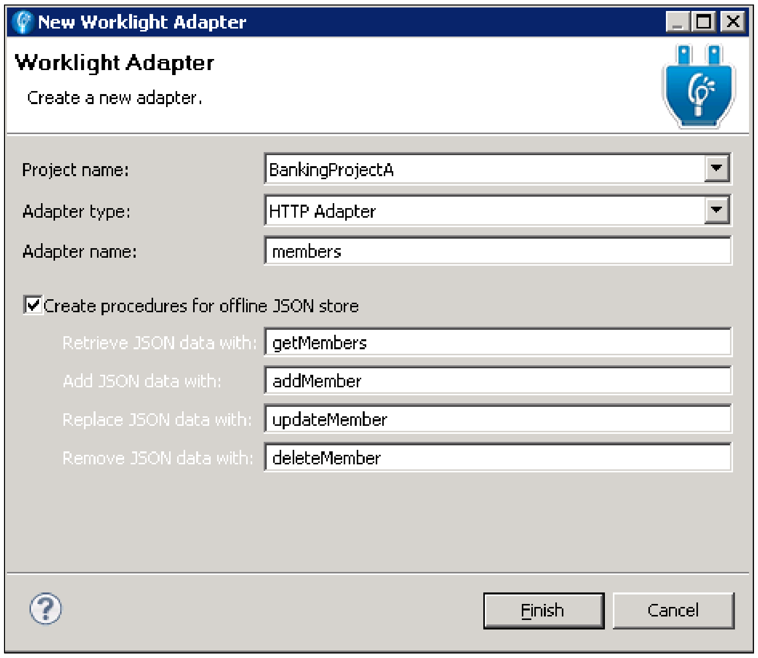
JSONStore data synchronization is enabled by linking a collection to a Worklight adapter. The implementation of this concept involves a local copy of data that is maintained by a Worklight application. The local changes can then be pushed to a back-end service to synchronize the content.

JSONStore is capable of tracking data changes made locally and by linking a collection to an adapter JSONStore can send and retrieve data to and from a collection.

To integrate JSONStore with an adapter, you can use either of the following functions, which are used to transmit and receive data:

- WL.Client.invokeProcedure
- jQuery.ajax

Worklight Studio facilitates the creation of adapters, and can be used to define the basic procedures for JSONStore, as shown in Figure 5-4 on page 89.


New Worklight Adapter

Worklight Adapter

Create a new adapter.

Project name: BankingProjectA

Adapter type: HTTP Adapter

Adapter name: members

☑ Create procedures for offline JSON store

Retrieve JSON data with: getMembers

Add JSON data with: addMember

Replace JSON data with: updateMember

Remove JSON data with: deleteMember

Figure 5-4 JSONStore adapter

Here is an example use case involving a bank employee who has installed and started to use a financial application:

1. The employee downloads a set of financial documents while at work. Optimally this is achieved using a wireless fidelity (WiFi) network.
2. The data is securely stored on the device, because the employee is prompted to provide a password to store the content inside JSONStore.
3. The employee can access the information and edit the content offline.
4. When the employee connects to the network, possibly a corporate WiFi network, the employee synchronizes the locally made JSONStore updates with the adapter. The Worklight adapter propagates the changes to the back-end system.

5.2.6 JSONStore encryption using FIPS 140-2

FIPS are the standards and guidelines that are provided by the United States National Institute of Standards and Technology (NIST) for federal government computer systems.

Android and iOS devices both enable the use of FIPS 140-2 validated libraries for the encryption and decryption of locally stored data. By default, JSONStore uses non-FIPS 140-2 validated libraries. Because JSONStore uses an open source library called Open Secure Sockets Layer (OpenSSL) to encrypt data, the FIPS-compliant version of this library can be used. This will ensure that JSONStore will operate in FIPS-compliant mode.

The following architectures are supported for iOS and Android that will ensure JSONStore is FIPS-compliant, as shown in Table 5-3.

Table 5-3 FIPS 140-2 supported architectures.

Android	iOS
armv7	armv7

Android	iOS
x86	i386

See the section about FIPS 140-2 support found in the IBM Worklight V6.0.0 Information Center for additional guidance:

http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/topic/com.ibm.worklight.help.doc/admin/c_using_FIPS_140-2_support.html

5.3 Client-side device provisioning and application authenticity

Worklight provides secure methods of identifying devices that have the necessary privileges to run and install Worklight applications. The main purpose of *device provisioning* is to achieve device identity validation.

Application authenticity is used to obtain and validate an application's identity. It is based on the same concept of identity validation that is used by Worklight to validate authorized users. It enhances Worklight security and incorporates a check for application integrity. See 4.1.3, "Authentication realms" on page 49 for additional information.

5.3.1 Device provisioning

Device identity is obtained and validated through device provisioning. The device ID that represents the device identity is used to uniquely identify multiple devices with the Worklight server. The client side is responsible for the generation of the device ID. This security mechanism is important, because it enables various other Worklight features, such as push notifications and reports.

The concept of device identity is vital for the development of secure Worklight applications, as it enables for device filtering regarding the connection to the Worklight server.

A pair of *Public Key Infrastructure (PKI)-based keys* are created when a Worklight application is run for the first time on a mobile device. The keys are then used to sign the device and application public characteristics which are then sent to the Worklight Server and used for authentication processing.

The generated keys alone do not provide enough security as it is possible for any application to generate a key pair. A *certificate* must be created, which is handled by an external trusted authority that enhances security by signing the key pair. The external trusted authority can either be the Worklight server or a custom provisioning service. The process of obtaining the certificate is called provisioning. More information regarding provisioning implementation can be found in 5.3.3, "Implementing device provisioning" on page 92.

The application is able to store the key pair within the device keystore when a certificate has been obtained. The access to the device keystore is protected by the operating system.

The Worklight platform provides support for several types of device provisioning:

- No provisioning
- Auto provisioning
- Custom provisioning

The key pair is capable of representing a single application, a group of applications, or an entire device. For the single application provisioning process, each application that is located on the device must generate its own key pair. It is also possible to allocate the same key pair for a group of applications. The key pair can also be used to represent the whole device. This enables the use of the same key pair for all of the applications that are installed on the device.

Client-side device provisioning flow

This section demonstrates the process interaction involved on the client side during the device provisioning cycle.

During the device provisioning process, the client performs the following actions:

1. The client issues a data request to the server.
2. The client generates a device ID, and obtains a certificate from an external trusted authority.
3. The client sends the certificate to the server with the client-side generated device ID.
4. If the certificate is valid, the client issues another data request which also includes the certificate, and the server responds with the requested data.

No provisioning

The provisioning process is not enabled and does not occur. You should only use this type of provisioning during the development stages of the application.

The client application does not initiate the provisioning process.

If you want to disable provisioning for a custom security test that requires device identity, you must include an additional realm as shown in Example 5-10.

Example 5-10 Disable provisioning for a custom security test

```
<customSecurityTest name="customTests">
   ...
   <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true"/>
</customSecurityTest>
```

The default option for mobile apps is no provisioning.

Auto provisioning

In this mode, a certificate for the device and application data is automatically issued by the Worklight server. The server is now responsible for the provisioning services.

The client is responsible for obtaining the device ID, and initiating the auto provisioning cycle, by utilizing the Worklight server as a provisioning service. The client application on the device stores the certificate, and it is then used to sign the payload that is sent to the Worklight server. The client certificate is validated by the server regardless of how it is obtained.

The client responds with a certificate-signed payload when the Worklight server issues a device identity challenge. In case the client does not have a certificate, a certificate signing request is issued by the client to the server. Worklight server responds by sending a certificate to the client. The client automatically responds by sending a signed payload that is deemed valid by the server, and the provisioning process is a success.

The client issues the original data request, which is processed by the server when the necessary authentication and provisioning services have been completed successfully.

It is advisable to use auto provisioning after authenticity checks have been validated, because the certificate is issued to any device that requests it. Therefore application authenticity features should be enabled before using auto provisioning.

Custom provisioning

With custom provisioning, you are provided with the option of using custom provisioning services other than the Worklight server. In this mode, an external trusted system can be used to issue a certificate. This type of provisioning is an extension of auto-device provisioning that facilitates the definition of custom provisioning rules.

Support: Only Android and iOS devices support auto provisioning and custom provisioning.

5.3.2 Device ID on Android and iOS

A device ID is generated using the Worklight framework to provide unique identification for devices that are connected to the Worklight server. The calculation of the unique device ID involves the generation of a Globally Unique Identifier (GUID). The device authentication process is responsible for generating the GUID.

The device and Worklight server can be used to store the device ID. On iOS, the device keychain is used to store the device ID. On Android, the application keystore, which is located in the application sandbox folder, is used to store the device ID. Worklight server can be used to store the device ID when enabling push notification subscription services.

The device operating system determines the availability of the device ID. On Android and iOS, the same device ID can be used for single or multiple applications from the same vendor. The device ID allocated for a specific device is persistent, so if you decide to remove and then reinstall an application, the device ID remains the same.

Worklight stores the device ID in the raw data reports that do not provide any special access controls, because the device ID is not considered sensitive information.

5.3.3 Implementing device provisioning

You are able to configure and change the type of device provisioning that can be implemented, and the keystore that is going to issue a certificate. This section provides an overview of the implementation steps that are associated with device provisioning, and are supported by Worklight.

Configure the keystore

The default behavior of the Worklight server is to use its internal keystore to issue a certificate. To use your own keystore, and to avoid using the default internal Worklight keystore, you must modify the `worklight.properties` file that is located in the server directory, as shown in Figure 5-5 on page 93.

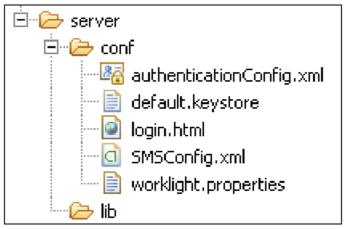


Figure 5-5 Worklight studio server directory

The default values are shown in Example 5-11.

Example 5-11 Custom keystore

```
###################################
#    Worklight Default Certificate
###################################
#wl.ca.keystore.path=
#wl.ca.keystore.type=
#wl.ca.keystore.password=
#wl.ca.key.alias=
#wl.ca.key.alias.password=
```

Provide the following values:

- `wl.ca.keystore.path=conf/default.keystore`

  This value represents the path to the keystore that is relative to the server folder within the Worklight project.

- `wl.ca.keystore.type=jks or pkcs12`

  This value represents the type of the keystore file, Java keystore, or Personal Information Exchange Syntax Standard.

- `wl.ca.keystore.password=worklight`

  This value represents the password to the keystore file.

- `wl.ca.key.alias=customkeypair`

  This value represents the alias of the entry where the key pair and certificate are stored.

- `wl.ca.key.alias.password=worklight`

  This value represents the password to the alias.

Enable auto provisioning and application authenticity on Android

For additional information, refer to "Auto provisioning" on page 91. To enable auto provisioning when using mobile security test, see Example 5-12.

Example 5-12 Auto provisioning for mobile security test

```
<mobileSecurityTest name="mobileTests">
   <testAppAuthenticity/>
   <testDeviceId provisioningType="auto" />
   <testUser realm="myMobileLoginForm" />
</mobileSecurityTest>
```

For the mobile security test, the `<testAppAuthenticity/>` child element ensures that application authenticity is performed. Similarly, for a custom security test, `<test realm=”wl_authenticityRealm”/>` can be used.

You have the option of defining a new realm, or using one that is already provided by Worklight (for example, a form-based authenticator with a non-validating login module). A non-validating login module does not validate user credentials, and any combination of user name and password are accepted. The sample security tests can be found in the `authenticationConfig.xml` file.

Use the application authenticity check with this type of provisioning. To validate the authenticity of an Android application, you must first follow these steps:

1. Set up a public signing key value in the `application-description.xml` file. The value of the certificate that is used to sign Android applications is held by this key.
2. Extract the public key using the provided Worklight studio wizard, or perform this action manually. This section uses the Worklight wizard to obtain the public signing key.
3. Create a Worklight hybrid application with an Android environment. Locate an Android folder, as shown in Figure 5-6.

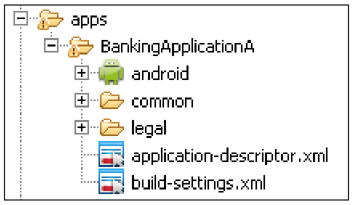
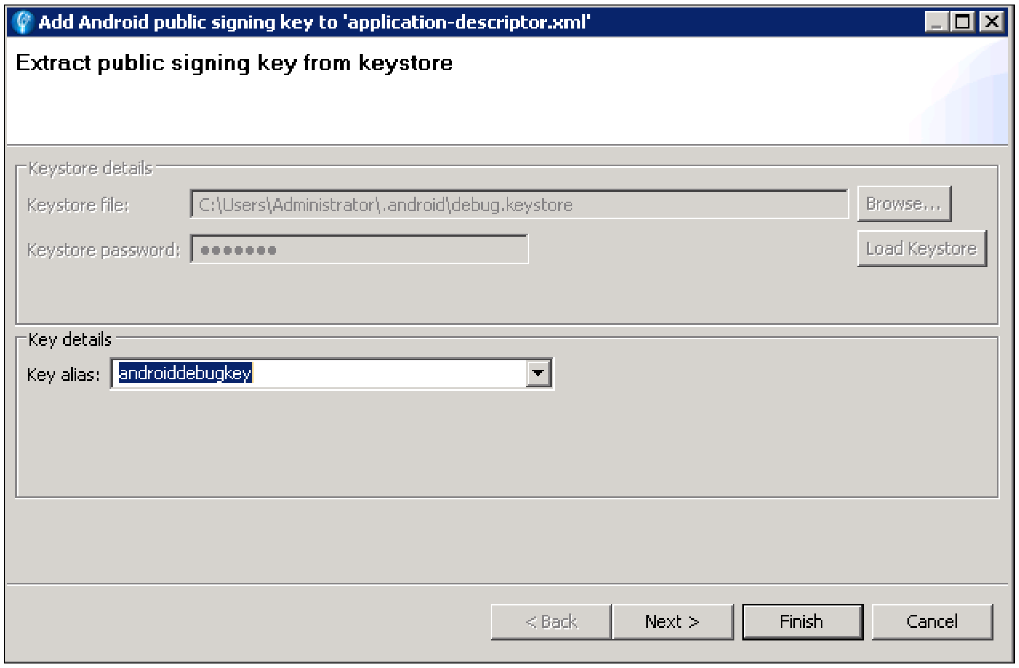


Figure 5-6 Android environment

4. Right-click your Android environment and select **Extract public signing key**. A new wizard will start, as shown in Figure 5-7.



Extract public signing key from keystore

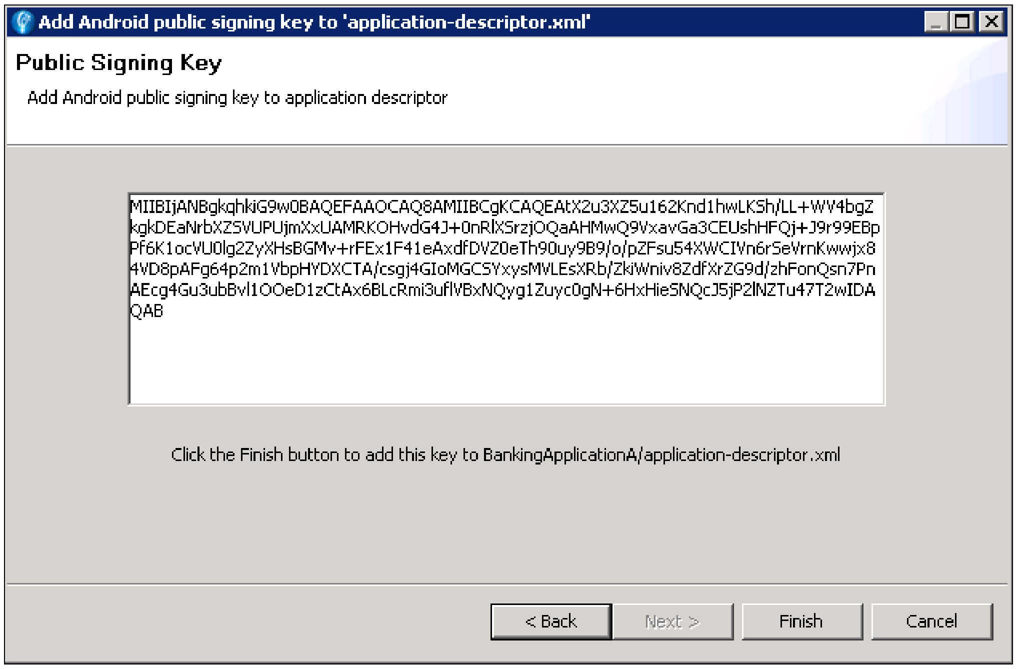
Keystore file: C:\Users\Administrator\.android\debug.keystore

Keystore password: •••••••

Key alias: androiddebugkey

Figure 5-7 Worklight wizard for extracting public signing key

5. Provide the keystore file location. This file is located in `{Users}/.android/debug.keystore`. This is the default location for the development environment.
6. In the keystore password field, enter `android`. This is the default password.
7. Select **Load Keystore**. You are now provided with an additional field for **Key alias**.
8. Select the predefined key alias from the provided list. The alias available in Figure 5-7 on page 94 is **androiddebugkey**.
9. Click **Next**. A new wizard will open, as seen in Figure 5-8.


Add Android public signing key to 'application-descriptor.xml'
Public Signing Key
Add Android public signing key to application descriptor
< Back
Next >
Finish
Cancel

```
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtX2u3XZ5u162Knd1hwLKSh/LL+WV4bgZ
kgkDEaNrbXZSVUPUjmXxUAMRKOHvdG4J+0nRlXSrzjOQaAHMwQ9VxavGa3CEUshHFQj+J9r99EBp
Pf6K1ocVU0lg2ZyXHsBGMv+rFEx1F41eAxdfDVZ0eTh90uy9B9/o/pZFsu54XWCIVn6rSeVrnKwwjx8
4VD8pAFg64p2m1VbpHYDXCTA/csgj4GIoMGCSYxysMVLEsXRb/ZkiWniv8ZdfXrZG9d/zhFonQsn7Pn
AEcg4Gu3ubBvl1OOeD1zCtAx6BLcRmi3uflVBxNQyg1Zuyc0gN+6HxHieSNQcJ5jP2lNZTu47T2wIDA
QAB
```

Click the Finish button to add this key to BankingApplicationA/application-descriptor.xml

Figure 5-8 Android public signing key

10. Click **Finish**. This will augment the `application-descriptor.xml` file by including the newly generated public signing key, as seen in Example 5-13.

Example 5-13 Public signing key in application-descriptor.xml

```
<publicSigningKey>MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtX2u3XZ5u162Knd1h
wLKSh/LL+WV4bgZkgkDEaNrbXZSVUPUjmXxUAMRKOHvdG4J+0nRlXSrzjOQaAHMwQ9VxavGa3CEUshH
FQj+J9r99EBpPf6K1ocVU0lg2ZyXHsBGMv+rFEx1F41eAxdfDVZ0eTh90uy9B9/o/pZFsu54XWCIVn6
rSeVrnKwwjx84VD8pAFg64p2m1VbpHYDXCTA/csgj4GIoMGCSYxysMVLEsXRb/ZkiWniv8ZdfXrZG9d
/zhFonQsn7PnAEcg4Gu3ubBvl1OOeD1zCtAx6BLcRmi3uflVBxNQyg1Zuyc0gN+6HxHieSNQcJ5jP2l
NZTu47T2wIDAQAB</publicSigningKey>
```

It is also possible to implement custom provisioning when you need to change the provisioning process or modify the scope (device, group, or application). To implement custom provisioning, you must create your own custom challenge handler, login module, and authenticator.

5.3.4 Control and confirm application authenticity

Editions: It is important to note that only the Consumer or the Enterprise edition of Worklight Studio provides the capability for application authentication.

The steps mentioned in the previous section, "Enable auto provisioning and application authenticity on Android" on page 93, illustrate the process required to enable application authentication.

There are additional steps that help confirm and test application authenticity. First, you must ensure that the application connects to the Worklight server when it starts. This enables you to test if the application is able to successfully connect to the server. Edit the `initOption.js` file located in the `js` folder, as shown in Example 5-14.

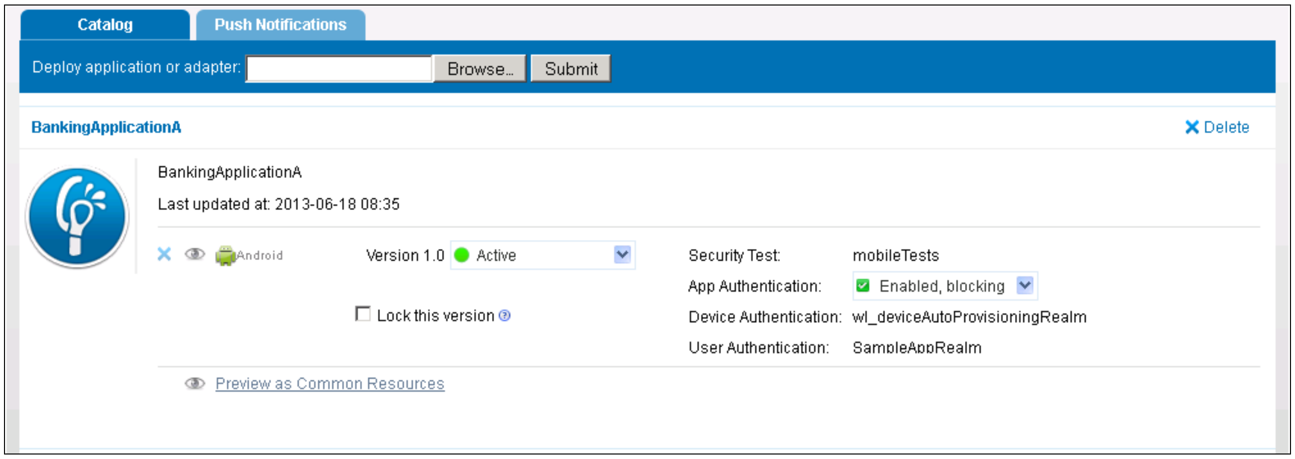
Example 5-14 Worklight connection options

```
var wlInitOptions = {

   // # Should application automatically attempt to connect to Worklight Server on
   //application start up
   // # The default value is true, we are overriding it to false here.
    connectOnStartup : true,
```

The Worklight console also provides additional runtime functionality related to application management and authentication. It is possible to disable and enable the application authenticity realm in run time. The following modes, which can be enabled, are shown in Figure 5-9:

Disabled	The check for application authenticity is disabled.
Enabled, servicing	The application authenticity check is enabled but, if the authentication fails, the application is still processed and served by the Worklight server.
Enabled, blocking	The application authenticity check is enabled but, if the authentication fails, the application is not processed and served by the Worklight server.



BankingApplicationA

Last updated at: 2013-06-18 08:35

Version 1.0 ● Active

Lock this version

Security Test: mobileTests

App Authentication: ✓ Enabled, blocking

Device Authentication: wl_deviceAutoProvisioningRealm

User Authentication: SampleAppRealm

Figure 5-9 Application authenticity control as provided by Worklight console

You are able to define separate security tests for specific application environments. These security tests can be used to protect application environments. If you want to use a security test other than the default Worklight security test, you must modify the `application-descriptor.xml` file. In Example 5-15, a mobile test called `mobileTests` is being used to protect an Android environment that is also shown in Figure 5-9 on page 96.

Example 5-15 Protect Android environment with a security test.

```
<android securityTest="mobileTests" version="1.0">
   <worklightSettings include="true"/>
        <security>
         ...
        </security>
</android>
```

5.4 Direct Update

The Worklight server provides the capability to automatically update hybrid iOS and Android applications with new versions of their web resources (HTML, JavaScript, and CSS). This is achieved without the user having to get these updates from an app store.

The Worklight console can be used to manage your applications. It enables you to view all of the applications that are installed, and groups them by the platform. Direct Update is capable of providing great advantages for a business:

- It is now possible to ensure that all of the users are using the latest version of the application.
- It provides security enhancements relating to the response time of issuing security updates, because the whole process is fast and simple to perform.
- A better management of application functionality is now possible with Direct Update.

Direct Update is only available for Android and iOS, and it is not possible to update an application's native resources, because only the web resources are updated during the process.

Direct Update based on application versions

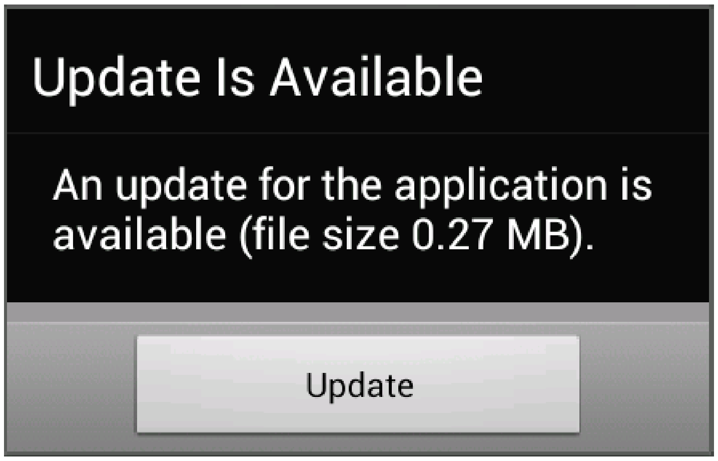
Direct Update is initiated when an application is redeployed to the Worklight server, without changing its version, and an application with older versions of these resources connects to the server. An application checks for updates at startup, and on returning to foreground.

Worklight enables Direct Update by default. You must ensure that the version of the Worklight studio and server that are used to deliver the Direct Updates aligns with the version that was used to build the application.

5.4.1 Using Direct Update

To initiate a Direct Update, modify the web resources of an application and redeploy it to the Worklight server. For example, if a critical security problem has been detected in a banking application, the developers are able to issue a security update that will get propagated to user devices when they first start the application and connect to the Worklight server.

It is also possible to disable an application with Remote Disable, covered in 5.5, “Remote Disable” on page 98. The user will be presented with a dialog indicating a new update is available, as shown in Figure 5-10.


Update Is Available

An update for the application is available (file size 0.27 MB).

Figure 5-10 Direct Update

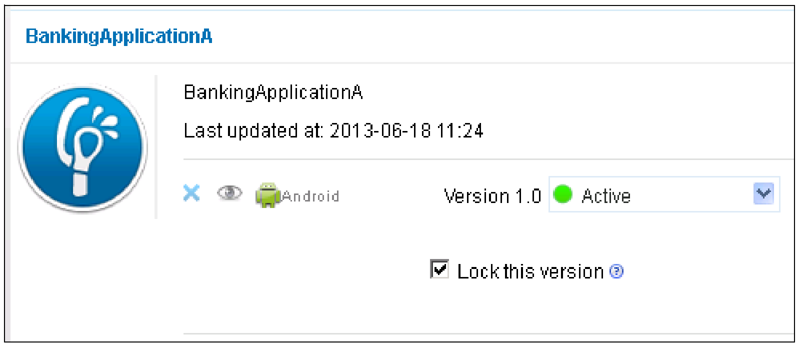
The only option provided is to update the application. If the user initiates the update, a progress indicator launches, and the download of the web resources commences. When the download is complete, the application reloads, and the necessary updates are applied.

5.5 Remote Disable

The Remote Disable feature enables you to disable access from a user to a specific version of an application, to prevent security flaws and prepare for a Direct Update. Direct Update and Remote Disable features can be used to greatly enhance security.

Lock an application

Worklight console enables you to lock an application to prevent further updating, as shown in Figure 5-11.


BankingApplicationA

BankingApplicationA

Last updated at: 2013-06-18 11:24

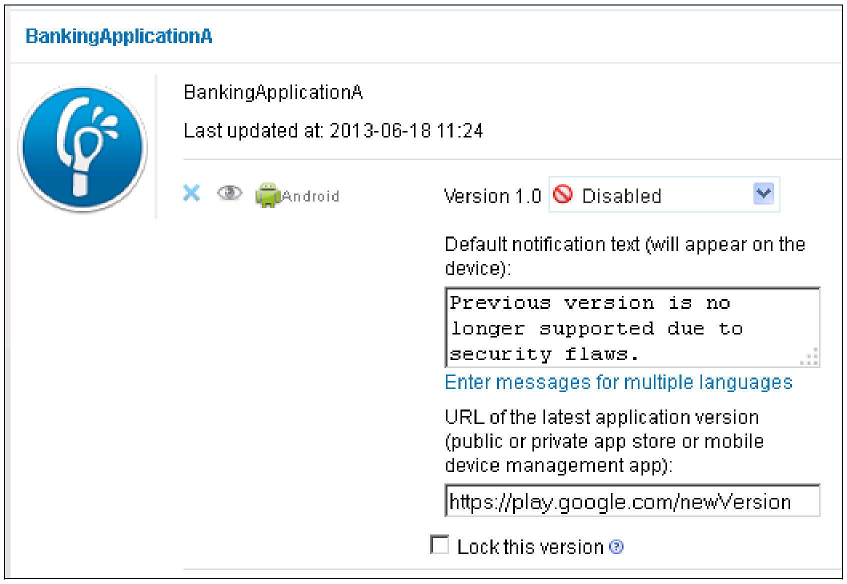
Android Version 1.0 ● Active

[✓] Lock this version

Figure 5-11 Lock an application using Worklight console

Application version control

If an older version of your application contains security issues, or if it is required to disable the current application to enhance security, the Worklight console can be used to perform the necessary actions, as shown in Figure 5-12 on page 99.



BankingApplicationA

BankingApplicationA

Last updated at: 2013-06-18 11:24

Version 1.0 Disabled

Default notification text (will appear on the device):

```
Previous version is no
longer supported due to
security flaws.
```

URL of the latest application version (public or private app store or mobile device management app):

https://play.google.com/newVersion

☐ Lock this version

Figure 5-12   Remote Disable feature

You are also provided with a field for the Uniform Resource Locator (URL) for the new version of the application, which can be acquired from an appropriate application market.

The user will get notified that a new version of the application is available, and they will have the choice of either exiting the application or acquiring a new version. This ensures that all of the users will have the same version of the application, and that the problematic application versions can be eliminated.

6

Integration with IBM Security Access Manager

This chapter explores the integration options achievable with IBM Security Access Manager to provide user authentication, identity attributes, and fine-grained authorization using risk-based access decisions to protect Worklight adapters.

An overview of the IBM Security Access Manager for Cloud and Mobile identifies the key components of the solution offering, and describes the capabilities and roles that each component offers.

This chapter examines options for providing user authentication and single sign-on (SSO), using one of the following methods:

- IBM Security Access Manager for web login form single sign-on
- Tivoli Federated Identity Manager OAuth single sign-on
- Worklight adapter single sign-on

6.3, “Risk-based access” on page 121 is introduced and illustrated to show how context-based access policies are defined, and how they are used to provide assurance and strong authentication for adapter-based transactions in IBM Worklight.

Finally, this chapter reviews how to implement the complete IBM Security Access Manager integration solution by using its individual capabilities.

6.1 IBM Security Access Manager introduction

IBM Security Access Manager for Cloud and Mobile extends user access protection to mobile and cloud environments. It combines the features and strengths of IBM Security Access Manager for Web, IBM Tivoli Federated Identity Manager, and IBM Tivoli Security Policy Manager to provide enhanced threat detection and prevention capabilities for cloud and mobile environments.

It offers user authentication, federated SSO, and risk scoring to help detect and prevent security breaches and web fraud. It provides context-based access management for mobile endpoints, such as smartphones and tablets, so that users do not inadvertently expose sensitive information technology (IT) assets in an insecure environment. It enables you to set up and run consistent security policies across multiple applications and for multiple users.

IBM Security Access Manager for Cloud and Mobile includes the following key features:

- Authentication and authorization of mobile application (app) users and devices, with advanced session management and integration with IBM Worklight
- Federated SSO and identity mediation across different cloud service providers and web services
- Risk-based access as a pluggable and configurable component

6.1.1 IBM Security Access Manager components

Each of the components can be used stand-alone to provide individual solutions and capabilities. However, when integrated, they form a security solution that provides a holistic approach to securing your Worklight applications and infrastructure, and reduces complexities with user directory integration.

IBM Security Access Manager for web

Available as a hardware appliance, virtual appliance, or software, IBM Security Access Manager for web (previously known as IBM Tivoli Access Manager for e-business), provides integration ready for use with web applications, including Worklight.

This integration provides secure, unified, and personalized user experiences by using web SSO and user identity attributes. The scalable web reverse proxy acts as an identity firewall with the authentication, authorization, and administration interfaces, which simplifies integration to help secure access to Worklight.

In addition to the web reverse proxy (IBM Security Access Manager for Web, or WebSEAL) component, the IBM Security Web Gateway Appliances provide access control, authentication, and session management for web applications. In addition, it provides protection from external threats using the IBM X-Force Intrusion Protection Protocol Analysis Module.

Using IBM Security Access Manager for web to provide SSO for Worklight applications is demonstrated in 6.2.1, “IBM Security Access Manager for web login form single sign-on” on page 104.

Tivoli Federated Identity Manager

IBM Tivoli Federated Identity Manager adds support for OAuth 1.0 and 2.0 to your environment, and can also provide additional authorization capabilities that you can take advantage of when integrating with Worklight. Using Tivoli Federated Identity Manager to provide SSO for the Worklight application is demonstrated in 6.2.2, “Tivoli Federated Identity Manager OAuth single sign-on” on page 111.

In addition to federated SSO, Tivoli Federated Identity Manager’s risk-based access (RBA) enforcement features can take advantage of context-based attributes of the mobile device and Worklight adapter properties to identify the risk associated with anytime, anywhere mobile access to Worklight applications. RBA capabilities are demonstrated in 6.3, “Risk-based access” on page 121.

Tivoli Security Policy Manager

IBM Tivoli Security Policy Manager centralizes security policy management and fine-grained data access control for applications, databases, portals, and services. The runtime security services (RTSS) component of Tivoli Security Policy Manager enables the engine to perform RBA decisions for web-based transactions.

By integrating Tivoli Security Policy Manager with your Tivoli Federated Identity Manager and IBM Security Access Manager for web, you can author, test, distribute, and govern rich, attributes-based access policy constructs for your Worklight applications. You can also reuse those policies across other enterprise applications, supporting delegated authorization. Support for the deployment of raw Extensible Access Control Markup Language (XACML) policy is also provided.

6.2 Enabling identity aware applications

When using IBM Security Access Manager for authentication, the authenticated user’s identity is propagated to the Worklight server, without the need for directory synchronization between the two systems. In addition to the user name, additional attributes for the user contained in the IBM Security Access Manager user registry can be supplied as extended attributes to Worklight. These attributes can then be used to provide personalization in the mobile app or Worklight adapters.

All user details populated during SSO are available in the attributes of the realm that was authenticated against by the Worklight login module and authenticator. These attributes can be accessed in an adapter, as shown in Example 6-1, and can be passed back to the client mobile app for additional processing.

Example 6-1 Retrieving user attributes

```
var attributes = WL.Server.getActiveUser().attributes;
var webSealExtendedAttribute = attributes.get('UserSurname');
```

In this section, sample code and configuration extracts are provided to demonstrate IBM Security Access Manager authentication using Login forms and OAuth. The samples provided here are adapted from the IBM Security Access Manager for IBM Worklight integration offering available from the following website:

http://www-01.ibm.com/support/docview.wss?uid=swg24034222.

This book only provides the high-level steps of the IBM Security Access Manager configuration, focusing on Worklight integration examples and settings.

6.2.1 IBM Security Access Manager for web login form single sign-on

WebSEAL can provide SSO for Worklight mobile apps using existing authentication mechanisms, which reduces provisioning of additional systems and directory integration. By asserting the authenticated user's identity to the Worklight server, directory integration is simplified without the need to configure multiple user repositories and login modules in the Worklight server.

Successful user authentication must be completed before access is granted to the Worklight server, which provides a leading practice security strategy called *defense in depth*. This can be extended to include session management and coarse-grained authorization decisions.

The authenticated user's identity can be provided to Worklight using a variety of techniques, including LTPA tokens, HTTP headers, mutually authenticated SSL, and custom login modules. This provides flexibility for integration with existing security architectures.

To achieve SSO, the Worklight client challenge handler must be configured to accept a `pkmslogin.form` challenge from WebSEAL, and to provide valid credentials for successful authentication.

In addition, the Worklight client security test, realm, and the authentication data type inserted by WebSEAL must exactly match the security test, realm, and login module configured in the Worklight server.

Authentication process flow

The login sequence when using forms-based authentication with IBM Security Access Manager is shown in Figure 6-1 on page 105.

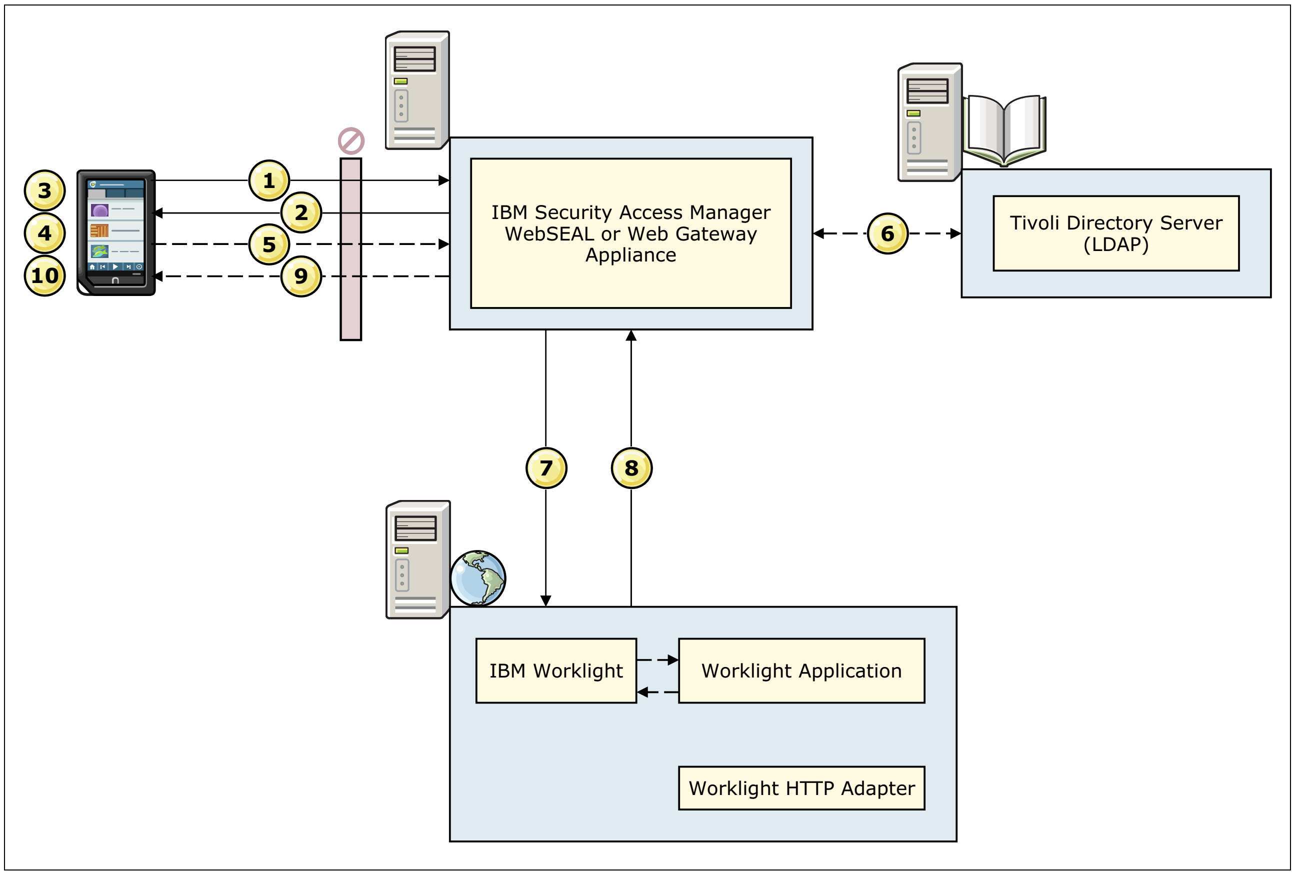

IBM Security Access Manager WebSEAL or Web Gateway Appliance
Tivoli Directory Server (LDAP)
IBM Worklight
Worklight Application
Worklight HTTP Adapter
1
2
3
4
5
6
7
8
9
10

Figure 6-1 IBM Security Access Manager forms-based authentication

Logging in consists of the following steps:

1. The client connects to Worklight server (for example, `/reach`).
2. WebSEAL, acting as the reverse proxy, caches the `/reach` request and responds to the client with a `200 OK` status response and the WebSEAL login form.
3. The client detects the login form as a custom response, and displays the login form on the device to the user.
4. The user enters their user name and password, and clicks **Login.**
5. The client performs a `POST` operation to WebSEAL, providing the login form data.
6. WebSEAL authenticates the user.
7. If successful, WebSEAL forwards the original `/reach` request to the Worklight server and the configured authentication data, either the Hypertext Transfer Protocol (HTTP) header or Lightweight Third Party Authentication (LTPA).
8. The Worklight server, acting as the client, retrieves the authentication data, sets the user data for the realm, and returns the successful `200 OK` response to WebSEAL.
9. WebSEAL, acting as the reverse proxy, applies any path or host name filtering, and returns the response to the client.
10. The client continues to run with subsequent requests through WebSEAL (for example, `/reach`).

Worklight client application challenge handler

The challenge handler of the Worklight application must detect a **`WebSEAL PKMS`** login form challenge, and provide a mechanism for the user to enter credentials for the IBM Security Access Manager server.

One solution is to display a simple login form on the device, allowing the user to enter a user name and password, and submit these using a login button. When the login button is pressed, the user name and password are submitted to the IBM Security Access Manager server for validation.If successful, the original request is forwarded to Worklight.

A simple challenge handler for login form authentication is shown in Example 6-2.

Various events, such as authentication errors, password change notifications, and authentication errors, can also be detected in the `isCustomResponse` function, and an appropriate message or action returned to the user.

Example 6-2 Simple IBM Security Access Manager login form challenge handler

```
var REALM_HTTPHEADER = 'HeaderAuthRealm';
var LOGIN_FORM_TAM = 'pkmslogin.form';

function showLoginScreen() {
   $("#index").hide();
   $("#authPage").show();
}

function showMainScreen() {
   $("#authPage").hide();
   $("#index").show();
}

var websealRealmChallengeHandler =
WL.Client.createChallengeHandler(REALM_HTTPHEADER);
var lastRequestURL;

websealRealmChallengeHandler.isCustomResponse = function(response) {
   //A normal login form has been returned.
   var findLoginForm = response.responseText.search("pkmslogin.form");
   if (findLoginForm >= 0) {
      lastRequestURL = response.request.url;
      return true;
   }

   //Need to also check for errors and handle as appropriate

   //This response is a worklight server response, handle it normally
   return false;
};

websealRealmChallengeHandler.handleChallenge = function(response) {
   showLoginScreen();
};

websealRealmChallengeHandler.handleFailure = function(response) {
   console.log("Error during WebSEAL authentication.");
};
```

```
websealRealmChallengeHandler.submitLoginFormCallback = function(response) {
   var isCustom = websealRealmChallengeHandler.isCustomResponse(response);
   if (isCustom) {
      websealRealmChallengeHandler.handleChallenge(response);
   }
   else {
      //hide the login screen, we are logged in
      showMainScreen();
      websealRealmChallengeHandler.submitSuccess();
   }
};

$("#loginButton").click(function(){
   var reqURL = "/../../../" + LOGIN_FORM_TAM;
   var options = {method: "POST"};
   options.parameters = {
      User name : $("#username").val(),
      password : $("#password").val(),
      "login-form-type" : "pwd"
   };
   options.headers = {};
   websealRealmChallengeHandler.submitLoginForm(reqURL, options,
         websealRealmChallengeHandler.submitLoginFormCallback);
   }
);
```

Worklight server login module

Authentication to the Worklight server is handled by a Worklight login module. You must configure the appropriate login module corresponding to the required authentication mechanism.

Login modules are responsible for validating credentials supplied by an authenticator, and are configured in the `<Worklight_WAR>/WEB-INF/classes/conf/authenticationConfig.xml` file. You can configure more than one login module in the Worklight server, and the realm associated with the security test will determine which login module is called.

Both HTTP Header and LTPA Token login modules are available.

To use HTTP Header, add the `com.worklight.core.auth.ext.HeaderLoginModule` login module, specifying `iv-user` as the `user-name-header` and `display-name-header`, as shown in Example 6-3. These are WebSEAL-specific headers, and are sent by WebSEAL to the Worklight server containing user authentication data.

Example 6-3 HTTP Header authentication login module

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <loginModules>
      ...
      <loginModule name="HeaderAuthModule">
         <className>com.worklight.core.auth.ext.HeaderLoginModule</className>
         <parameter name="user-name-header" value="iv-user"/>
         <parameter name="display-name-header" value="iv-user"/>
      </loginModule>
      ...
   </loginModules>
   ...
</tns:loginConfiguration>
```

To use LTPA Token, add the `com.worklight.core.auth.ext.WebSphereLoginModule` login module, as shown in Example 6-4.

Example 6-4 LTPA Token authentication login module

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <loginModules>
      ...
      <loginModule name="WASLTPAModule">
         <className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
      </loginModule>
      ...
   </loginModules>
   ...
</tns:loginConfiguration>
```

Worklight server realm

Realms are configured to call an authenticator, which is responsible for collecting credentials from the client. You must configure the appropriate authenticator corresponding to the required authentication mechanism from IBM Security Access Manager.

Authentication modules are configured in the `<Worklight_WAR>/WEB-INF/classes/conf/authenticationConfig.xml` file.

You can configure more than one realm, however the name must be unique, and the `loginModule` value must exactly match one of the <loginModule> elements, the configuration of which is detailed in “Worklight server login module” on page 107.

Both HTTP Header and LTPA Token realm authenticators are available.

To use HTTP Header, add the `com.worklight.core.auth.ext.HeaderAuthenticator` authentication module, as shown in Example 6-5.

Example 6-5 HTTP Header authentication Realm

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <realms>
      ...
      <realm name="HeaderAuthRealm" loginModule="HeaderAuthModule">
         <className>com.worklight.core.auth.ext.HeaderAuthenticator</className>
      </realm>
      ...
   <realms>
   ...
</tns:loginConfiguration>
```

To use LTPA Token, add the `com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator` authentication module, specifying the `login-page` and `error-page` as a fallback to forms-based authentication to Worklight, as shown in Example 6-6.

Example 6-6 LTPA Token authentication realm

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <realms>
      ...
      <realm name="WASLTPARealm" loginModule="WASLTPAModule">

<className>com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator</className>
         <parameter name="login-page" value="/login.html"/>
         <parameter name="error-page" value="/loginError.html"/>
      </realm>
      ...
   <realms>
   ...
</tns:loginConfiguration>
```

Worklight server security test

Security tests require a Worklight user to pass the test before access is granted. The Worklight client is configured to handle security tests for a Worklight server realm.

Security tests are configured in the `<Worklight_WAR>/WEB-INF/classes/conf/authenticationConfig.xml` file. You can configure more than one security test, and each test can handle challenges for one or more realms.

To use HTTP Header, add the `HeaderAuthRealm` to the security tests, as shown in Example 6-7.

Example 6-7 HTTP Header authentication security test

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <securityTests>
      ...
      <webSecurityTest name="ISAMAuthentication-web-securityTest">
         <testUser realm="HeaderAuthRealm"/>
      </webSecurityTest>
      <mobileSecurityTest name="ISAMAuthentication-mobile-securityTest">
         <testUser realm="HeaderAuthRealm"/>
         <testDeviceId provisioningType="none"/>
      </mobileSecurityTest>
      ...
   </securityTests>
   ...
</tns:loginConfiguration>
```

To use LTPA Token, add the `WASLTPARealm` to the security tests, as shown in Example 6-8.

Example 6-8 LTPA Token authentication security test

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <securityTests>
      ...
      <webSecurityTest name="ISAMAuthentication-web-securityTest">
         <testUser realm="WASLTPARealm"/>
      </webSecurityTest>
      <mobileSecurityTest name="ISAMAuthentication-mobile-securityTest">
         <testUser realm="WASLTPARealm"/>
         <testDeviceId provisioningType="none"/>
      </mobileSecurityTest>
      ...
   <securityTests>
   ...
</tns:loginConfiguration>
```

Build and deploy the Worklight application

Having configured the Worklight server and the client application for IBM Security Access Manager forms-based authentication challenges, you can now use the Build For Remote Server function in the Worklight Studio.

You must update the Uniform Resource Locator (URL) to correspond to the protocol, the fully qualified domain name, and the port (if nonstandard) of the IBM Security Access Manager server. Also, you need to define a *context path*, which must be created as a corresponding WebSEAL *junction path*.

When deploying the application, Worklight Studio uses the IBM Security Access Manager address to publish, and is unable to authenticate to WebSEAL. Instead, you can log in to the Worklight console directly, and manually upload the application.

IBM Security Access Manager junction

The **`pdadmin`** command-line utility can be used for both the software and web gateway appliance versions of IBM Security Access Manager, or the local management interface (LMI) for the web gateway appliance.

In both instances, create an indiscernible path junction with a name that exactly matches the context path of the Worklight application, as described in “Build and deploy the Worklight application”. To ensure that the Worklight device authentication challenge is not intercepted by IBM Security Access Manager, specify the **`-b ignore`** parameter.

If you are using HTTP Header authentication, ensure that the `iv-user` HTTP Header for the authenticated user is inserted, as shown in Example 6-9.

Example 6-9 HTTP Header authentication junction creation

```
pdadmin sec_master> server task <instance-name>-webseald-<webseal-hostname>
create -t tcp -h <worklight-server-fqdn> -p <worklight-serverport>
-c iv-user -x -b ignore /worklight
```

If you are using LTPA Token authentication, you must export the LTPA key file from the Worklight server, and copy it locally to the IBM Security Access Manager server. Provide the full file name and path to the key file during the junction creation, as shown in Example 6-10.

Example 6-10 LTPA Token authentication junction creation

```
pdadmin sec_master> server task <instance-name>-webseald-<webseal-hostname>
create -t tcp -h <worklight-server-fqdn> -p <worklight-serverport>
-A -F <exported-ltpa-key-path-and-filename> -Z <exported-ltpa-keypassword>
-x -b ignore /worklight
```

After you complete the required Worklight server configuration changes and client updates, IBM Security Access Manager now provides the user authentication and identity assertion to Worklight.

6.2.2 Tivoli Federated Identity Manager OAuth single sign-on

IBM Tivoli Federated Identity Manager adds support for OAuth 1.0 and 2.0 to your environment, which you can use to provide identity-aware applications. Instead of users supplying a user name and password, a defined OAuth registration process is followed, with an enrollment step completed on a separate, trusted, and secure system. The user obtains a time-sensitive registration code, called an *authorization code*, from this system.

The authorization code is valid for a single use only, is time-limited, and must be entered in the device and validated before it expires. The code entry can be manual, using the device keyboard, or it can take advantage of device integration, such as scanning a quick-response (QR) code.

When users start the application, a registration screen opens, on which they submit their registration code (the authorization code). If the validation is successful, they are authenticated, and Tivoli Federated Identity Manager populates a defined list of attributes into their Worklight session identity, including an *access token* and *refresh token*. The refresh token is stored securely on the device using the encryption features of the Worklight client, as described in 5.2, “Encrypted offline cache and JSONStore” on page 81.

Each time the application is started, authentication is seamless, with the device sending the refresh token on the user’s behalf to authenticate and populate their Worklight session identity. After successful authentication with the refresh token, Tivoli Federated Identity Manager provides the Worklight application with the following tokens:

- A new refresh token, which is again stored on the device and overwrites the original
- A time-sensitive access token, which is used during additional requests in the same session for authentication

Using the OAuth pattern, no personal information about the user is ever shown on the device, and the tokens stored on the device can be invalidated, temporarily or permanently preventing access for that device. You can also strengthen the security by adding an optional personal identification number (PIN) during authentication with a refresh token.

The OAuth integration is a complex scenario, and it requires a thorough understanding of the OAuth protocol and Tivoli Federated Identity Manager.

In this scenario, Tivoli Federated Identity Manager is the identity provider that the Worklight adapter calls to validate user authentication. During the authentication, only the required OAuth artifacts and user profile data are returned to the client, and no authentication is performed against WebSEAL.

However, OAuth authentication can be provided to WebSEAL by using a custom external authentication interface (EAI). This enables WebSEAL to accept OAuth tokens as an authentication mechanism, which is validated using a custom trust module running on Tivoli Federated Identity Manager. After successful validation, the provided access token (or refresh token) is exchange for a valid WebSEAL session for that user.

In this scenario, we assume the Tivoli Federated Identity Manager system is already configured for OAuth. See the section on configuring an OAuth federation in *the IBM Tivoli Federated Identity Manager Version 6.2.2 Configuration Guide* for configuration instructions:
http://pic.dhe.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.tivoli.fim.doc_6226/config/concept/OAuthConfiguring.html

Authentication process flow

The following implementation demonstrates a Worklight adapter-based authentication scenario. However, this can also be implemented as a custom login module and authenticator.

By using an adapter-based authentication design for the OAuth implementation, the identity provider can be updated or swapped entirely to a different provider without needing to deploy a new version of the client application. The authentication changes are not apparent to users, and no modifications are required on the Worklight server, which reduces change requests and server administration.

The OAuth flow can also be implemented in the device client itself using JQuery, bet there are limitations:

- The network request and response handling is non-trivial.
- The solution is not scalable.
- The solution requires the distribution of a new app to resolve errors or change details, such as the endpoint URLs.

A high-level overview of the OAuth registration and authentication is shown in Figure 6-2 on page 113.

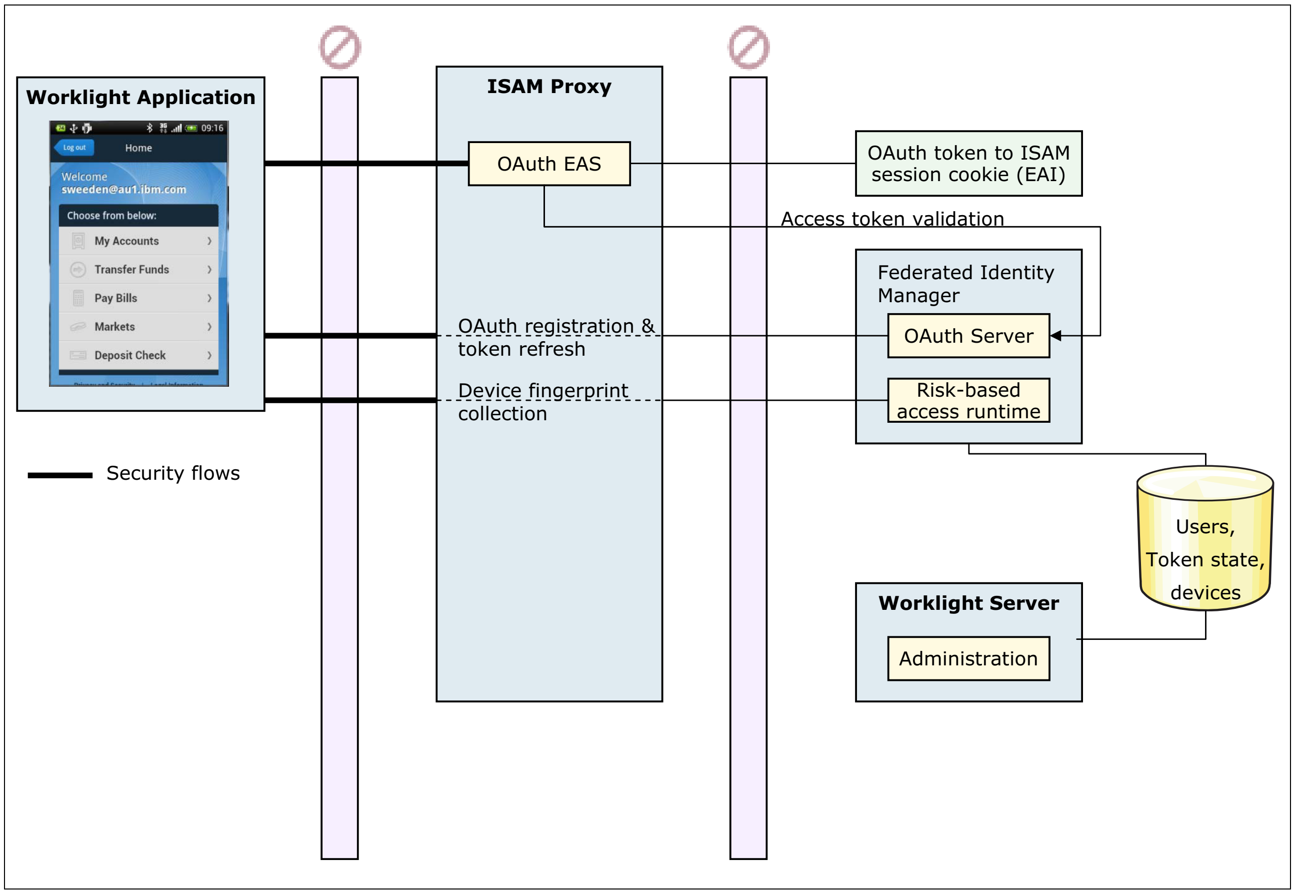


Figure 6-2 IBM Security Access Manager OAuth authentication

The following steps complete OAuth registration and authentication:

1. A client connects to the Worklight server, and accesses an adapter function that is protected by a security challenge (for example, `initOAuthAuthorizationCodeInit`).
2. Worklight detects the authentication requirement, and returns the adapter login function and response to the client.
3. The client detects the custom response for the OAuth challenge handler, and displays the registration code entry screen on the device. The entry screen contains instructions for the user about how to obtain a registration code.
4. The users provision themselves for mobile access in a separate, trusted web browser. They then obtain a time-sensitive, single-use registration code, and enter it into the device.
5. The registration code is sent to the Worklight adapter, which calls the OAuth Java class to validate the registration code.
6. The OAuth Java class calls the token endpoint URL, passing the registration code supplied by the user.
7. The token endpoint validates the registration code and, if successful, sends an access token and refresh token to the OAuth Java class. That class passes the tokens back to the adapter.
8. The adapter extracts the access token and refresh token. It then calls the OAuth Java class to retrieve the profile for the authenticated user, passing both tokens.
9. The OAuth Java class calls the endpoint profile URL, passing both tokens.

10. The profile endpoint validates the tokens and returns the user attributes to the OAuth Java class, which passes the attributes back to the adapter.
11. The adapter validates the authentication and, if profile retrieval was successful, sets the identity for the Worklight session, and then returns a success message to the client.
12. The client detects the success message, submits success to the challenge handler, and stores the refresh token on the device.
13. The challenge handler satisfies the conditions for accessing the initial protected resource, and the adapter function runs.

Worklight client application challenge handler

The challenge handler of the Worklight application detects a JavaScript Object Notation (JSON) parameter called **`authRequired`**, which is provided to the client if the Worklight realm starts the login function specified in the realm configuration.

If the device has not previously been through the initial registration process, and there is no refresh token in the encrypted storage of the device, the user must be prompted to enter a registration code and provided with details about where to get this registration code. After the registration code is entered, the OAuth login can commence.

If a token is available on the device, this can be used for authentication, and the process is not apparent to the user (no interaction is required).

A simple challenge handler for the OAuth adapter authentication is shown in Example 6-11, which provides the main functional logic. However, it does not show the callback functions for handling success and failure of the adapter invocation.

Various events, such as token expiration and authentication errors, can also be detected in the `isCustomResponse` function, and an appropriate message or action returned to the user.

Example 6-11 Simple OAuth challenge handler for adapter based authentication

```
var REALM_TFIMOAUTH = 'TFIMOAuthAdapterRealm';

//During application startup, read the refresh_token from the cache.
//If there isn't a refresh_token, show the registration screen.
function showRegisterScreen() {
   $("#index").hide();
   $("#regPage").show();
}

function showMainScreen() {
   $("#regPage").hide();
   $("#index").show();
}

var tfimOAuthAdapterRealmChallengeHandler =
WL.Client.createChallengeHandler(REALM_TFIMOAUTH);

tfimOAuthAdapterRealmChallengeHandler.isCustomResponse = function(response) {
   if (!response || !response.responseJSON || response.responseText === null) {
      return false;
   }
   //Detect the authentication trigger from onAuthRequired in the adapter
   if (typeof(response.responseJSON.authRequired) !== 'undefined'){
      return true;
   } else {
      return false;
   }
};

tfimOAuthAdapterRealmChallengeHandler.handleChallenge = function(response){
   var authRequired = response.responseJSON.authRequired;

   if (authRequired == true) {
      startOAuthLogin();
   else if (authRequired == false){
      showMainScreen();
      tfimOAuthAdapterRealmChallengeHandler.submitSuccess();
   }
};

function startOAuthLogin() {
   bool isAuthorizationCode = false;

   //Read the refresh_token from the cache.
   var code = readRefreshTokenFromEOC();

   //If no refresh_token is available use the the registration code.
   //The regPage should already be visible
   if (code == “”) {
      code = $("#AuthorizationCode").val();
      isAuthorizationCode = true;
   }

   if (code == null || code == "") {
      tfimOAuthAdapterRealmChallengeHandler.submitFailure();
   }
   else {
      //Determine which token we have. authorization_code is only used once.
      var grantType = "refresh_token";
      if (isAuthorizationCode) {
         grantType = "authorization_code";
      }

      var invocationData = {
         Adapter : "TFIMOAuthAdapter",
         procedure : "submitAuthentication",
         parameters : [ grantType, code ]
      };

      var options =  {
            onSuccess: onSubmitAuthenticationSuccess,
            onFailure: onSubmitAuthenticationFailure
      };
         //Attempt authentication to the adapter. We should loop back to
isCustomResponse on the response.

tfimOAuthAdapterRealmChallengeHandler.submitAdapterAuthentication(invocationData,
options);
```

```
   }
}
```

Worklight server adapter

To start the authentication process, the Worklight client must access a protected resource, such as an adapter, to start a security test for a realm, which in turn starts the authentication process on the Worklight server.

The function in the adapter used to submit authentication data must not require a security test, or must use a security test for a realm that has already been successfully authenticated. Otherwise, it will not be possible to submit authentication, and the user will enter an endless invoke and authentication loop.

In Example 6-12, the Worklight client attempts to start the `initOAuthAuthorizationCodeInit` function of an adapter that is secured by the `TFIMOAuthAdapter-securityTest`. This security test is configured for the `TFIMOAuthRelam`, which starts the adapter-based authentication.

When the Worklight client attempts to authenticate, it provides the credentials using the `submitAuthentication` function. After successful authentication, the `initOAuthAuthorizationCodeInit` function is run, and the response is returned to the Worklight client.

Example 6-12 Adapter-based authentication security test configuration

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wl:adapter xmlns:wl="http://www.worklight.com/integration" ...>
   <procedure name="submitAuthentication"/>
   <procedure name="initOAuthAuthorizationCodeInit"
securityTest="TFIMOAuthAdapter-securityTest"/>

</wl:adapter>
```

The protocol, host, and port configuration of the adapter can be left as the default values, because the adapter does not perform a `WL.Server.invokeHttp` where the endpoint address is controlled by those values.

A sample of code for the adapter is shown in Example 6-13. The `onAuthRequired` function is called from the Worklight server to initiate the authentication process. The client challenge handler, conversely, uses the `submitAuthentication` function to provide authentication data.

The `submitAuthentication` function references a Java class file that is responsible for performing the network operations:

- Contacting the identity provider
- Parsing the response so that it is accessible in the adapter

Example 6-13 Adapter-based authentication function for OAuth single sign-on

```
//Authentication function triggered by the Worklight server realm indicating to
the client that authentication is required and is a custom response.
function onAuthRequired(errorMessage){
   errorMessage = errorMessage ? errorMessage : null;
   return {
      authRequired: true,
      errorMessage: errorMessage
   };
}
```

```
//Main authentication function for validating an access_token or refresh_token as
the grantType when submitted by the client.
function submitAuthentication(grantType, accessCode){
...
   //Define the parameters for the Java class that will perform the network
operaitons
   var CLIENT_ID = 'mobileClient';
   var APPNAME = 'BANK_CO_A';
   //The token exchange URL
   var OAUTH_TOKEN_ENDPOINT =
'https://tfim.ibm.com/FIM/sps/oauth20sp/oauth20/token';
   //The profile URL to obtain user attributes after successful token validation
   var OAUTH_PROFILE_ENDPOINT =
'https://tfim.ibm.com/FIM/demo/oauthprotected/profile.jsp';

   //Initialize a new instance of the Java class for the network operations
   var oauth = new com.ibm.security.worklight.oauth.TFIMOAuth();
   oauth.setClientId(CLIENT_ID);
   oauth.setAppName(APPNAME);
   oauth.setOauthTokenEndpointURL(OAUTH_TOKEN_ENDPOINT);
   oauth.setOauthProfileEndpointURL(OAUTH_PROFILE_ENDPOINT);

   //Call the token endpoint to validate the token
   var authenticateRequest = oauth.sendAuthenticateRequest(grantType, accessCode);

   //If the authenticateRequest was successful parse the response
   var authResponseJSON = JSON.parse(authenticateRequest);

   //If the authenticateRequest was successful call the profile URL with the new
access_token
   var profileRequest = oauth.getProfileRequest(authResponseJSON.access_token);

   //If the profileRequest was successful parse the response
   var profileRequestJSON = JSON.parse(profileRequest);

   //Build the user identity with the attrubites from the authResponseJSON and
profileRequestJSON
   var tfimIdentity = {
      userId : profileRequestJSON.User name,
      displayName : profileRequestJSON.User name,
      attributes : {
         pinprotected : authResponseJSON.pinprotected,
         expires_in : authResponseJSON.expires_in,
         scope : authResponseJSON.scope,
         appInstance : authResponseJSON.appInstance,
         access_token : authResponseJSON.access_token,
         token_type : authResponseJSON.token_type,
         refresh_token : authResponseJSON.refresh_token,
         User name : profileRequestJSON.User name,
         timestamp : profileRequestJSON.timestamp
      }
   };
   //If authentication and profile retrieval was successful set the authenticated
user to the TFIM identity
   WL.Server.setActiveUser("TFIMOAuthAdapterRealm", tfimIdentity);

   //Send a callback to the Worklight client indicating success. The Worlight
client then must manually call the submitSuccess for the challenge handler.
   return {
        authRequired: false
        };
...
}
```

Worklight server login module

When using adapter-based authentication, the Worklight server does not perform any authentication itself. It is reliant on the `submitSuccess` function of the challenge handler and the `WL.Server.setActiveUser()` call in the adapter to add a user identity for the configured realm. The non-validating login module shown in Example 6-14 is used so that the Worklight server does not perform any validation of the user authentication.

Example 6-14 Non-validating login module for adapter-based authentication

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <loginModules>
      ...
      <loginModule name="TFIMOAuthAdapterLoginModule">

<className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
      </loginModule>
      ...
   </loginModules>
   ...
</tns:loginConfiguration>
```

Worklight server realm

The Worklight server realm is responsible for starting the required adapter functions for login and logout events. The two functions do not need to be in the same adapter, but the adapter name and the function specified in that adapter must exist.

In Example 6-15, the `TFIMOAuthAdapter` is called for both login and logout events.

Example 6-15 Adapter Based Authentication Realm

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <realms>
      ...
      <realm name="TFIMOAuthAdapterRealm"
loginModule="TFIMOAuthAdapterLoginModule">

<className>com.worklight.integration.auth.AdapterAuthenticator</className>
         <parameter name="login-function"
value="TFIMOAuthAdapter.onAuthRequired"/>
```

```
         <parameter name="logout-function" value="TFIMOAuthAdapter.onLogout"/>
      </realm>
      ...
   <realms>
   ...
</tns:loginConfiguration>
```

Worklight server security test

The authentication process commences when the Worklight client attempts to access the Worklight server (for example, on a `WL.Client.connect` that has a security test defined for the application at startup), or when it attempts to start an adapter protected by a security test.

In Example 6-16, the client attempts to access a protected adapter function, `initOAuthAuthorizationCodeInit`, which is protected by `TFIMOAuthAdapter-securityTest`.

Example 6-16 Custom security test for adapter-based authentication

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfigurationxmlns:tns="http://www.worklight.com/auth/config" ... >
   ...
   <securityTests>
      ...
      <customSecurityTest name="TFIMOAuthAdapter-securityTest">
         <test isInternalUserID="true" realm="TFIMOAuthAdapterRealm"/>
      </customSecurityTest>
      ...
   <securityTests>
   ...
</tns:loginConfiguration>
```

Build and deploy the Worklight application

Having configured the Worklight server and the client application for IBM Security Access Manager OAuth challenges, you can now use the Build For Remote Server function in the Worklight Studio.

You must update the URL to correspond to the protocol, the fully qualified domain name, and the port (if nonstandard) of the IBM Security Access Manager server. You also need to define a context path, which must be created as a corresponding WebSEAL junction path.

When deploying the application, Worklight Studio will use the IBM Security Access Manager address to publish, and is unable to authenticate to WebSEAL. Instead, you should log in to the Worklight console directly, and manually upload the application.

IBM Security Access Manager junction

In this example, the Worklight server is accessed using a WebSEAL junction. However, the custom EAI and trust module to enable OAuth authentication for WebSEAL has not been reviewed or demonstrated.

The performance and threat capabilities of the IBM Security Web Gateway Appliance can still be used by providing an unauthenticated access control list (ACL) on the junction path for Worklight, and allowing easy migration after the EAI and trust module are deployed.

An example of creating an unauthenticated ACL and attaching it to the Worklight path of the WebSEAL server is shown in Example 6-17.

The **pdadmin** command-line utility can be used for both the software and web gateway appliance versions of IBM Security Access Manager, or the LMI for the web gateway appliance. The object space path to where the ACL is attached must exactly match the context path of the Worklight application, as described in "Build and deploy the Worklight application" on page 119.

Example 6-17 Unauthenticated ACL creation

```
pdadmin sec_master> acl create unauth
pdadmin sec_master> acl modify unauth set unauthenticated Trx
pdadmin sec_master> acl modify unauth set any-other Trx
pdadmin sec_master> acl modify unauth set user sec_master TcmdbsvaBRrxl
pdadmin sec_master> acl attach /WebSEAL/localhost-default/worklight unauth
```

When adding an unauthenticated URL, access is then granted to all child paths. Therefore, an ACL must be created and attached to protect the Worklight console, only allowing authenticated administrators, as shown in Example 6-18.

Example 6-18 Protecting the Worklight Console using an ACL

```
pdadmin sec_master> acl create worklight_console
pdadmin sec_master> acl modify worklight_console set group <group_name> Trx
pdadmin sec_master> acl modify worklight_console set user sec_master TcmdbsvaBRrxl
pdadmin sec_master> acl attach /WebSEAL/localhost-default/worklight/console
worklight_console
```

You must create an indiscernible path junction with a name exactly matching that of the Worklight application context path, as described in "Build and deploy the Worklight application" on page 119. To ensure that the Worklight device authentication challenge is not intercepted by IBM Security Access Manager, specify the **-b ignore** parameter. The sample junction creation is shown in Example 6-19.

Example 6-19 Worklight junction creation

```
pdadmin sec_master> server task <instance-name>-webseald-<webseal-hostname>
create -t tcp -h <worklight-server-fqdn> -p <worklight-serverport>
-c iv-user -x -b ignore /worklight
```

Having completed the required Worklight server configuration changes and client updates, Tivoli Federated Identity Manager now provides the user authentication and identity assertion to Worklight using the OAuth pattern.

6.2.3 Other authentication types

Both IBM Security Access Manager and Tivoli Federated Identity Manager provide ready-for-use, customizable authentication mechanisms for user authentication in the Worklight client.These two security products can provide the following functions:

- Web and federated SSO to users across multiple applications, inside and outside their network infrastructure
- Support for standards such as OAuth, Security Asset Markup Language (SAML), OpenID, Liberty profile, Web Services (WS)-Federation, WS-Security, and WS-Trust

- Security token service supporting WS-Trust, user name, SAML, IBM Resource Access Control Facility (RACF®), X590, and Kerberos tokens

By taking advantage of the Worklight authentication framework, and delegating authentication and coarse-grained authorization to IBM Security Access Manager and Tivoli Federated Identity Manager, the identity of authenticated users can be asserted to Worklight without requiring custom Worklight components to integrate.

6.3 Risk-based access

IBM Security Access Manager for Cloud and Mobile provides context-based access management for mobile endpoints, such as smartphones and tablets. It also identifies precise risks that these devices can introduce with their anytime, anywhere access to enterprise systems in both cloud and mobile scenarios. The solution enables organizations to enhance traditional access control mechanisms that use static credential details with contexts that include device, environment, identity, and behavior patterns.

When using RBA, each transaction is assessed using static and contextual attributes to calculate the risk. This risk assessment provides a score that is based on configurable weights, which are assigned to context attributes and behavior attributes. If the risk score is high, further challenges are presented to the user, or access is denied. If the risk score is low, the user is granted access.

6.3.1 Risk-based access overview

RBA provides an access decision and enforcement that is based on a dynamic risk assessment, or a confidence level in the transaction, and uses behavioral and contextual data analytics to calculate risk. It is a pluggable and configurable component for IBM Tivoli Federated Identity Manager, which can also be integrated with IBM Security Access Manager for Web to protect web-based transactions through the secure web reverse proxy.

The architecture of risk-based access is shown in Figure 6-3.

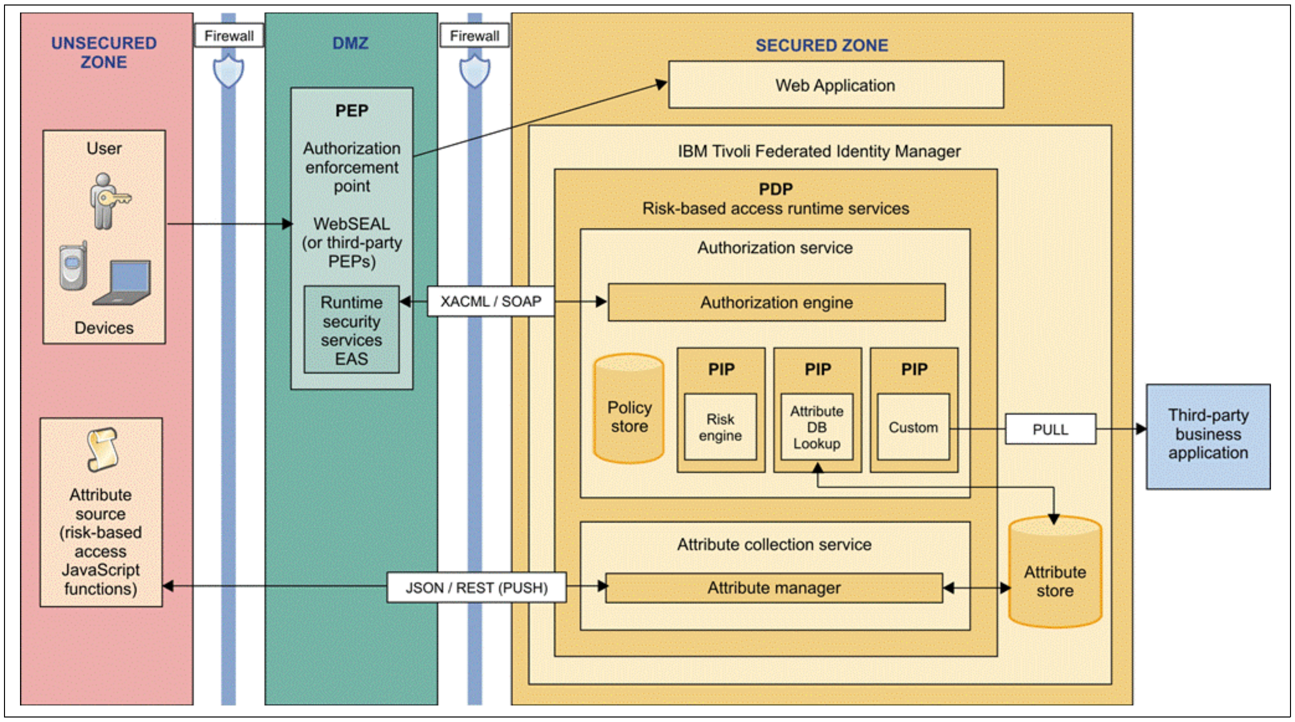

UNSECURED ZONE
Firewall
DMZ
Firewall
SECURED ZONE
User
Devices
PEP
Authorization enforcement point
WebSEAL (or third-party PEPs)
Runtime security services EAS
Web Application
IBM Tivoli Federated Identity Manager
PDP
Risk-based access runtime services
Authorization service
XACML / SOAP
Authorization engine
Policy store
PIP
Risk engine
PIP
Attribute DB Lookup
PIP
Custom
PULL
Third-party business application
Attribute source (risk-based access JavaScript functions)
JSON / REST (PUSH)
Attribute collection service
Attribute manager
Attribute store

Figure 6-3 Risk-based access architecture

The attribute source, known as the *attribute collector service*, is a small JavaScript file that is run locally on the device. The attribute collector service retrieves context attributes about the device that is used for access, such as the device location, operating system, and a custom attribute that can be retrieved using Apache Cordova (PhoneGap). These are made available as attributes for the RBA policy, and can be combined with other environment attributes, such as traditional network, time of day, and other attributes.

The RBA policy for managing risk-based access is configured using commands available through the WebSphere Application Server AdminTask framework, which supports Java Tool Command Language (Java TCL, or Jacl) scripting, Java Python (Jython) scripting, and programmatic Java API. If your environment has deployed Tivoli Security Policy Manager, you manage policies using the central, standardized interface.

Details about installing, configuring, and administering risk-based access are available in the IBM Tivoli Federated Identity Manager, Version 6.2.2 information center:
http://pic.dhe.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.tivoli.fim.doc_6226/rba-home.html.

Configuration instructions for the runtime security services External Authorization Service (EAS) are available in the IBM Security Access Manager for Web, Version 7.0 information center:
http://pic.dhe.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.isam.doc_80/ameb_webseal_guide/concept/con_rba_eas.html.

6.3.2 Securing Worklight adapters with risk-based access

Using the Worklight security framework, adapter access can be controlled by configuring a security test that corresponds to a Worklight server realm, and requires a challenge handler in the Worklight client to provide the necessary authentication. With this approach, the Worklight client has already accessed the endpoint URL of the adapter.

However, by integrating Worklight with RBA, access to the endpoint URLs can be restricted until the Worklight client successfully passes an RBA challenge. In this way, the decision and enforcement of context-based security is moved into the network perimeter (or DMZ) rather than the secured zone.

The RBA policy for Worklight adapters can evaluate access decisions using the adapter name, function, and parameters passed to the adapter, in addition to the standard attributes provided by the attribute collector service and the RBA infrastructure. The Worklight mobile app must be configured to handle the RBA challenge directly, because there is no interaction with the Worklight server until the Worklight client successfully passes the RBA evaluation.

You can use RBA with either the authentication mechanism described in 6.2.1, “IBM Security Access Manager for web login form single sign-on” on page 104, or that in 6.2.2, “Tivoli Federated Identity Manager OAuth single sign-on” on page 111. This is because the challenge handler for RBA is independent of any realms and security tests configured in Worklight.

In Chapter 2, “Business scenario used in this book” on page 19, Banking Company A required an additional authentication challenge for transactions of more than $200. The following sections describe how RBA is used to achieve the additional level of assurance, to prove the identity of the person requesting the transfer, which helps minimize fraud and increase security.

Worklight server adapter

For Banking Company A, an adapter called BankingApprovals is used in their Worklight mobile apps. In the adapter, two functions are provided, `doTransfer` and `getBalance`. Both functions are protected using a Worklight security test that ensures that the user has been authenticated by IBM Security Access Manager, as shown in Example 6-20.

Example 6-20 BankingApprovals adapter

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wl:adapter xmlns:wl="http://www.worklight.com/integration" ...>
   <procedure name="doTransfer"
securityTest="ISAMAuthentication-mobile-securityTest"/>
   <procedure name="getBalance"
securityTest="ISAMAuthentication-mobile-securityTest"/>
</wl:adapter>
```

The sample code extract in Example 6-21 shows the parameters passed to each of the functions used in the adapter.

Example 6-21 Parameter signature of the BankingApprovals adapter

```
function doTransfer(amount, from_account, to_account) {
   ...
   //Do the transfer between accounts
   ...
}
function getBalance(account) {
   ...
   //Get the balance of the named account
   ...
}
```

6.3.3 Authorization process flow

When using risk-based access to protected resources, a number of interactions take place before WebSEAL forwarding the client request to the protected resource, in this case a Worklight adapter, as shown in Figure 6-4.

When a higher authentication level is required to access a protected resource, the client is required to step up their authentication level using a one-time password (OTP). In a raw policy context, this is referred to as *permit with obligations*.

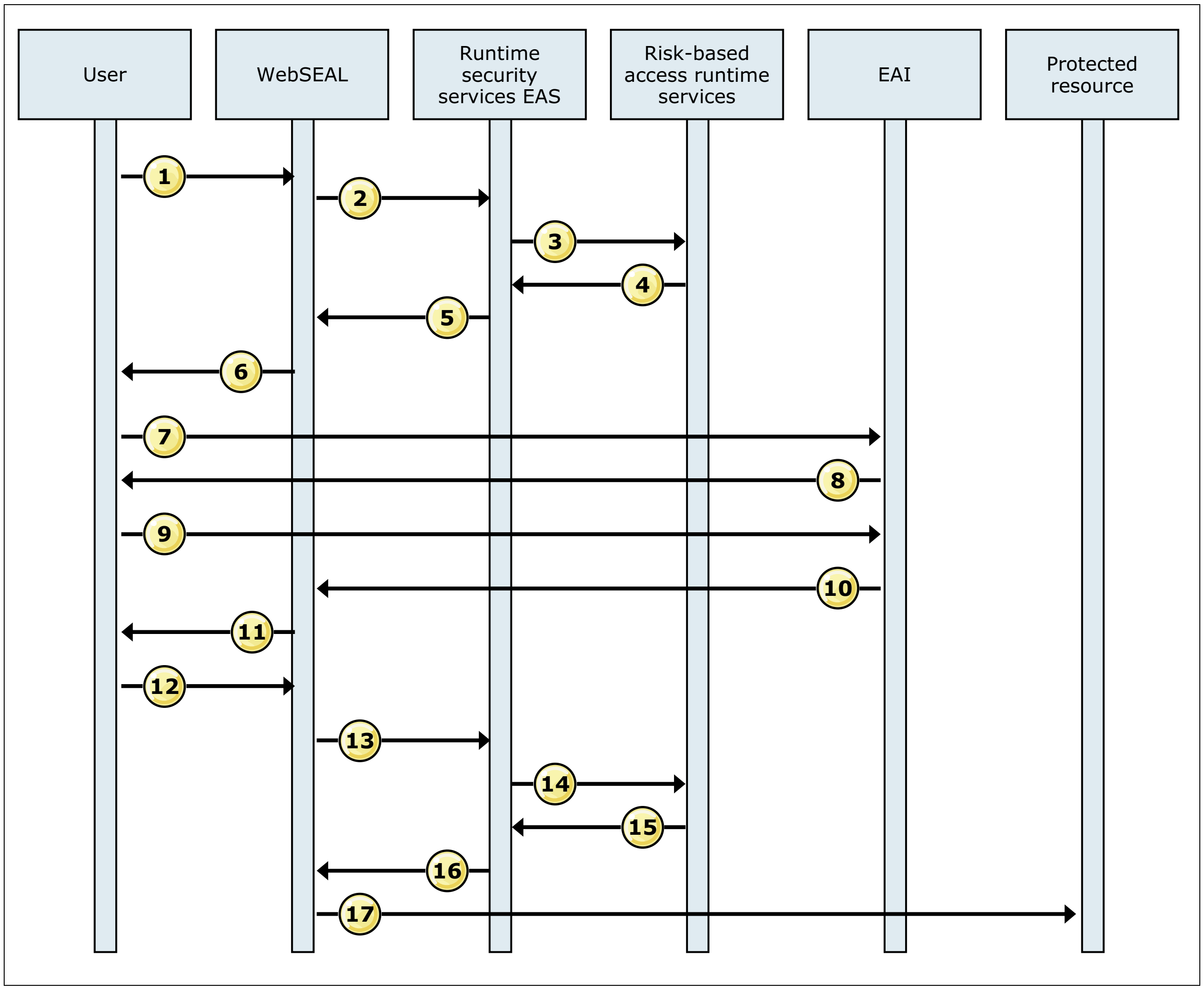

User
WebSEAL
Runtime security services EAS
Risk-based access runtime services
EAI
Protected resource
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

Figure 6-4 Risk-based access transaction flow

The following steps take place in RBA:

1. A Worklight mobile app accesses the Worklight server using WebSEAL, and attempts to start a Worklight adapter.
2. The adapter URL triggers a protected object policy (POP) in WebSEAL that is handled by the WebSEAL runtime security services EAS.

3. The runtime security services EAS calls the RBA runtime services to evaluate the request using the current context attributes for the device against the defined policy.
4. The RBA runtime services evaluate the request, and return a result of `permit with obligations` to the runtime security services EAS.
5. The runtime security services EAS returns the result to WebSEAL, requiring the user to step up their authentication level.
6. WebSEAL sends a `302 redirect` to the Worklight mobile app to the step-up authentication page.
7. The Worklight mobile app follows the redirect to the step-up authentication page. This page must be unprotected, so that the client can access the page without authentication.
8. The step-up authentication page is delivered to the client from the OTP EAI. Using an out-of-band mechanism, a OTP is sent to the user.
9. The user receives the OTP, enters this into the device, and submits it to begin the step-up authentication.
10. The OTP EAI evaluates the OTP and, if successful, raises the authentication level of the user for the remainder of their WebSEAL session.
11. WebSEAL raises the user's authentication level, and sends a `302 redirect` to the Worklight mobile app, to re-access the resource originally requested in step 1.
12. The Worklight mobile app follows the redirect to again request the protected resource.
13. The adapter URL triggers a POP in WebSEAL, which is handled by the WebSEAL runtime security services EAS.
14. The runtime security services EAS calls the RBA runtime services, passing the new user authentication level set by the successful OTP challenge.
15. The RBA runtime services evaluate the user authentication level with the risk score, and returns a `permit` result to the runtime security services EAS.
16. The runtime security services EAS passes the permit result to WebSEAL.
17. WebSEAL forwards the original request from the Worklight client to the Worklight adapter.

Each time an adapter invocation is attempted, the runtime security services EAS are triggered by the POP, and the RBA policy will be evaluated. Provided that the user has stepped up to the required authentication level in the same session, they will not be prompted to step up or provide a OTP until their session expires.

6.3.4 Policy authoring

RBA is driven through policy, which is authored using the Policy Generation Language, and added to the RBA policy store using the `manageRbaPolicy` administration task in WebSphere Application Server. When Tivoli Security Policy Manager is integrated with Tivoli Federated Identity Manager and IBM Security Access Manager for web, eXtensible Access Control Markup Language (XACML) policy can also be directly distributed.

The RBA policy does not bind business rules to application logic, which enables the creation of context-based access policies rather than traditional user and group access permissions. This is implemented using XACML. When a policy is authored using the Policy Generation Language, it is converted to XACML. The converted or manually-authored XACML is then stored in authorization service policy store.

Banking Company A defined the business rule requiring customers to provide an additional level of authentication for transfers greater than $200.

The corresponding Worklight adapter `POST` request will contain the following post-data elements:

Adapter	The Worklight adapter name.
Procedure	The function to be run in the adapter.
Parameters	A URL-encoded JSON array of the input parameters for the function.

The policy, triggered by the POP, must match all of the following conditions:

- A Worklight URL is requested.
- The `BankingApprovals` adapter is invoked.
- The `doTransfer` function is called.
- The parameter passed is the transfer amount.
- If the parameter is greater than 200, `permit with obligations`, otherwise simply `permit`.
- If the authentication level of the user is 3, having already authenticated using a OTP, no further step up is required.

An extract of the XACML RBA policy to support this business rule is shown in Example 6-22.

Example 6-22 Sample XACML policy for Worklight adapter invoke transactions

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet:PolicySet
xmlns:PolicySet="urn:oasis:names:tc:xacml:2.0:policy:schema:os" ...>
...
...
  <ns5:PolicySet  ... PolicySetId="RPS:RBA_WORKLIGHT99">
      <ns5:Obligations>
         <ns5:Obligation FulfillOn="Permit" ObligationId="otp"/>
      </ns5:Obligations>
      <ns5:Rule Effect="Permit" RuleId="RBA_WORKLIGHT99">
         <ns5:Condition>
            <ns5:Apply FunctionId="...function:and">
               <ns5:Apply FunctionId="... :function:string-regexp-match">
                  <ns5:AttributeValue
DataType="...#string">/worklight/.*</ns5:AttributeValue>
               </ns5:Apply>
               <ns5:Apply FunctionId="... :function:string-equal">
                  <ns5:EnvironmentAttributeDesignator
AttributeId="post-data:adapter" ... />
                  <ns5:AttributeValue
DataType="...#string">BankingApprovals</ns5:AttributeValue>
               </ns5:Apply>
               <ns5:Apply FunctionId="... :function:string-equal">
                  <ns5:EnvironmentAttributeDesignator
AttributeId="post-data:procedure" ... />
                  <ns5:AttributeValue
DataType="...#string">doTransfer</ns5:AttributeValue>
               </ns5:Apply>
               <ns5:Apply FunctionId="... :function:integer-less-than">
                  <ns5:EnvironmentAttributeDesignator
AttributeId="urn:ibm:security:isam:authentication:level" ... />
                  <ns5:AttributeValue
DataType="...#integer">3</ns5:AttributeValue>
               </ns5:Apply>
```

```
              <ns5:Apply FunctionId="... :function:integer-greater-than">
                 <ns5:EnvironmentAttributeDesignator
AttributeId="urn:ibm:security:wl:parameter" ... />
                 <ns5:AttributeValue
DataType="...#integer">99</ns5:AttributeValue>
              </ns5:Apply>
            </ns5:Apply>
        </ns5:Condition>
      </ns5:Rule>
  </ns5:PolicySet>
  ...
...
</PolicySet:PolicySet>
```

By using RBA to manage context-based access to your Worklight adapters, you can rapidly change your policy without requiring modifications to the Worklight mobile app. The logic to provide multiple context-based access policies is more manageable and scalable than requiring multiple and potentially overlapping security tests on Worklight adapter.

6.3.5 Policy attachment

In addition to configuring the RBA services for IBM Security Access Manager and Tivoli Federated Identity Manager, the resource paths for the Worklight application must be protected. A WebSEAL POP is used to trigger the runtime security services EAS and start the RBA process.

The **pdadmin** command-line utility can be used for both software and web gateway appliance versions of IBM Security Access Manager or the LMI for the web gateway appliance. In both instances, a POP must be created and attached to the adapter URI.

Alternatively, the POP can be attached at the root level for all Worklight applications, services, and requests. However, it is not necessary unless you also want to apply RBA policies to requests in addition to adapters.

In Example 6-23 you create a new POP called *RBA pop*, accepting the default options of the creation. However, additional coarse-grained authorization requirements can be added. The POP is then attached to the WebSEAL object space, only for adapters run from the `WLAppName` application on Android and iPhone devices.

The level of access enables us to control specific applications and devices without needing to evaluate the RBA policy for all URLs. The adapter name, function, and parameters are evaluated in the RBA policy itself.

Example 6-23 Sample POP to trigger risk-based access for Worklight applications

```
pdadmin sec_master> pop create rbapop
pdadmin sec_master> pop attach
/WebSEAL/localhost-default/worklight/apps/services/api/WLAppName/android/query
rbapop
pdadmin sec_master> pop attach
/WebSEAL/localhost-default/worklight/apps/services/api/WLAppName/iphone/query
rbapop
```

Alternatively, to protect all adapter requests across all Worklight applications, a single POP attachment can be performed at the `/WebSEAL/localhost-default/worklight/apps/services/api/` level. To apply RBA policy for all operations of all applications, the policy can instead be attached at the Worklight root junction level of `/WebSEAL/localhost-default/worklight`.

6.3.6 Policy evaluation

During policy evaluation, context-based attributes provided during the request, or collected previously, can be used to determine the risk score. In addition, at run time, you can manipulate attribute data or query extended sources, such as databases or directories, to enrich the context data. When the attributes have been collected, weighted, combined, and evaluated, a risk score is determined, and a decision made against the policy.

Attribute collector service

The attribute collector service is a small JavaScript file that is run locally on the device to retrieve context attributes about the device that is used for access, such as the device location, operating system, and custom attributes that can be retrieved using Apache Cordova (PhoneGap).

This JavaScript file can be included in the main HTML file of the Worklight application, and either run from the device or loaded dynamically from the Tivoli Federated Identity Manager server using a script `include`. The attribute collector service is found under the `https://isam.fqdn.com/FIM/sps/ac/js/info.js` path. A sample of common device attributes collected is shown in Example 6-24.

Example 6-24 Sample device and location attributes

```
if (typeof device === 'undefined') {
   // Cordova is not present
} else {
   jsonObj['deviceModel'] =  String(device.model);
   jsonObj['deviceCordova'] =  String(device.cordova);
   jsonObj['deviceUuid'] =  String(device.uuid);
   jsonObj['deviceVersion'] =  String(device.version);
   jsonObj['devicePlatform'] =  String(device.platform);
   jsonObj['geoCurrentPosition'] =  String(geolocation.getCurrentPosition);
   jsonObj['language'] = String(navigator.systemLanguage);
}
```

The device attributes and the default set of attributes provided by the attribute collector service can also be used as parameters in the Policy Generation Language and XACML policies.

Policy information point for Worklight adapters

A policy information point (PIP) can be run during the policy evaluation to manipulate incoming or existing attributes and can also be used to retrieve additional data from other sources to provide further context-based attributes.

When a Worklight adapter is started, an HTTP POST request is made by the Worklight client, containing the following HTML Form elements:

Adapter	The Worklight adapter name.
Procedure	The function to be run in the adapter.
Parameters	A URL-encoded JSON array of the input parameters for the function.

The adapter and procedure can easily be used in the RBA policy evaluation. However, the URL-encoded JSON array needs to be decoded and extracted before the underlying values can be evaluated against a policy.

To do this, a custom PIP for handling Worklight adapter requests is used. The segment of sample code in Example 6-25 demonstrates the URL decode and retrieval of a specific adapter parameter based on the index configuration.

For Banking Company A, the first parameter of their `doTransfer` function contains the transfer amount, as shown in "Worklight server adapter" on page 123. Therefore, the configuration for the PIP must extract the first parameter (zero indexed) to retrieve the transfer amount.

Example 6-25 Custom PIP to extract Worklight adapter parameters

```
...
//The attribute ID of WL Adapater parameters
static final String WL_PIP_ATTRIBUTE = "urn:ibm:security:wl:parameter";

//Config parameter for which WL Adapater attribute to retrieve
private int _parameterIndex;
...

private List<Attribute> getAttributes(RequestContext context, String attributeId,
String dataType, String issuer) {
   ...
   List<Attribute> attrList = new LinkedList<Attribute>();
   // URL decode, then separate as a JSON Array (e.g."[1,2,3]")
   String decodedParameters = URLDecoder.decode(encodedParameters, "UTF-8");
   JSONArray ja = new JSONArray(decodedParameters);

   //Add the configured WL adapter parameter to the attribute list
   attrList.add(AttributeFactory.createAttribute(WL_PIP_ATTRIBUTE, dataType,
ja.getString(_parameterIndex)));
   }
   ...
   return attrList;
}
```

Policy decision result

Upon evaluation of an RBA policy, a decision is returned to WebSEAL, which acts as the policy enforcement point (PEP). The following RBA policy decisions are possible:

- Permit. Allow the request to continue with no additional authentication, and forward the request to the protected resource.
- Deny. Do not allow the request to continue, and return an error to the client.
- Permit with Obligations. Allow the request to continue only when the obligations have been satisfied.

Banking Company A uses OTP authentication in their obligations to step up the user authentication level. The `permit with obligations` decision is returned to the client as a `302 redirect` to the step-up authentication page. The Worklight mobile app must handle the redirect and step-up request directly, because WebSEAL as the PEP will not forward the request to Worklight until the step up is successful.

6.3.7 Step-up authentication

When a `permit with obligations` access decision is returned to the Worklight mobile app, the app must handle the step-up authentication directly, because no requests are forwarded to the Worklight server until the obligation has been fulfilled.

Depending on the Tivoli Federated Identity Manager configuration, a number of mechanisms can be used to deliver the OTP to the user, such as SMS, email, Google Authenticator, and so on. Ideally, the OTP should not be delivered to the same device that is being used to make the transaction.

If the same device is the only option for OTP delivery, a two-factor or additional out-of-band validation, such as requiring a user PIN that is also known to the server, is appended to the OTP entered by the user and validated by the OTP EAI.

No Worklight server configuration is required, because the challenge handler in the Worklight mobile app is responsible for all operations during the step-up authentication using OTP.

Worklight client challenge handler

A challenge handler is created for a custom realm. It can be combined with an existing IBM Security Access Manager challenge handler and realm, or as a stand-alone challenge handler only used for step up.

The sample code extract in Example 6-26 is combined with the existing HTTP header authentication realm used in “Worklight server realm” on page 108. The challenge handler detects a custom response from WebSEAL, representing a login page or a step-up authentication in the `isCustomResponse` function.

If a step-up request is detected, the application opens a browser window inside the application, showing the step-up authentication page. The page contains the form to input and submit the OTP, which is hosted on the Tivoli Federated Identity Manager server.

When the OTP step up is successful, the browser detects the `302 redirect` back to the adapter start URL, and then closes itself, and the regular mobile app continues.

Example 6-26 Simple IBM Security Access Manager RBA Challenge Handler

```
var REALM_ISAM = ‘HeaderAuthRealm';
var RBA_STEPUP_ISAM = 'Access Manager for e-business Step Up Login';
var LOGIN_FORM_ISAM = 'pkmslogin.form';
var FIM_REDIRECT = '/FIM/demo/unprotected/generatelogin.jsp';
var ISAM_ADAPTER_QUERY_URL =
https://isam.fqdn.com/worklight/apps/services/api/WLAppName;

var RESPONSE_TYPE_NORMAL = 0;
var RESPONSE_TYPE_LOGIN = 1;
var RESPONSE_TYPE_STEPUP = 2;

var STEPUP_SUCCESS_DROID = ISAM_ADAPTER_QUERY_URL + '/android/query';
var STEPUP_SUCCESS_IOS = ISAM_ADAPTER_QUERY_URL + '/iphone/query';

var isamRealmChallengeHandler = WL.Client.createChallengeHandler(REALM_ISAM);

var lastRequestURL;

isamRealmChallengeHandler.isCustomResponse = function(response) {
    if(isWebSealLoginPage(response.responseText)) {
        isamRealmChallengeHandler.lastResponseDetails = {
            type: RESPONSE_TYPE_LOGIN,
            token: 'Unknown'
        };
        return true;
    }
    else if(isWebSealStepupPage(response.responseText)) {
        isamRealmChallengeHandler.lastResponseDetails = {
            type: RESPONSE_TYPE_STEPUP,
            response:response.responseText
        };
        return true;
    }
    else if (isTFIMLoginRedirect(response.responseText)) {
        isamRealmChallengeHandler.lastResponseDetails = {
            type: RESPONSE_TYPE_LOGIN,
            token: 'Unknown'
        };
        return true;
    }

    isamRealmChallengeHandler.lastResponseDetails = {
        type: RESPONSE_TYPE_NORMAL
    };

    //Need to also check for errors and handle as appropriate

    //This response is a worklight server response, handle it normally
    return false;
};

isamRealmChallengeHandler.handleChallenge = function(response) {
    if(isamRealmChallengeHandler.lastResponseDetails.type == RESPONSE_TYPE_LOGIN) {
        showLoginScreen(isamRealmChallengeHandler.lastResponseDetails.token, false);
    }
    else if(isamRealmChallengeHandler.lastResponseDetails.type ==
RESPONSE_TYPE_STEPUP) {
        handleStepupRequest(isamRealmChallengeHandler.lastResponseDetails.response);
    }
    else {
        //Unknown last response. Display an error.
    }
};

function isWebSealStepupPage(data) {
    return data.indexOf(RBA_STEPUP_ISAM) >= 0;
}

function isTFIMLoginRedirect(data) {
    return data.indexOf(FIM_REDIRECT) >= 0;
}

function isWebSealLoginPage(data) {
    return data.indexOf(LOGIN_FORM_ISAM) >= 0;
```

```
}

function showLoginScreen() {
   $("#index").hide();
   $("#authPage").show();
}

function handleStepupRequest(response) {
   //Use popup window in app.
   //Could instead use custom div in the app.
   var splitFront = response.split('window.location = "');
   var splitBack = splitFront[1].split("\"");
   var url = splitBack[0];

   var ref = window.open(url, '_blank', 'location=yes');

   ref.addEventListener('loadstop', function(event) {
      if(event.url == STEPUP_SUCCESS_DROID || event.url == STEPUP_SUCCESS_IOS) {
         ref.close();
         isamRealmChallengeHandler.submitSuccess();
      }
   });
}
```

To maintain a consistent user interface within the Worklight mobile app, the same control logic can be used to show and hide a custom `div` or `page` packaged with the application, to enable input and posting of an OTP to complete the step-up authentication.

6.4 Worklight adapter single sign-on

In addition to providing user authentication and identity assertion to the Worklight server, IBM Security Access Manager can also be used to propagate the authenticated user's identity to systems from which data is retrieved using Worklight adapters. When a Worklight adapter is invoked to the target the back-end system, the request appears as any other client request appears. The Worklight adapter must be able to provide the authenticated user's identity using a mechanism supported by the back-end system.

Products such as SAP, PeopleSoft, and Microsoft SharePoint do not support LTPA. Therefore, this particular user identity assertion token cannot be used to retrieve data directly from these systems as the Worklight authenticated user.

To achieve SSO for these systems, deploy IBM Security Access Manager as a web reverse proxy to take advantage of existing integration solutions offered for those products. In this way, the protected data can be accessed and made available to the Worklight adapter.

Authentication process flow

The Worklight server makes the request through the WebSEAL server, passing the session cookie of the authenticated user on the appropriate junction for the back-end data source.

This can be the same WebSEAL server that is accessed by the client device, or a different WebSEAL server inside the DMZ. The inside server must be configured to share sessions with the external one using fail-over or Session Management Server, or configured for LTPA authentication between WebSEAL servers.

In Figure 6-5, the same WebSEAL server instance used by the Worklight mobile app to access the Worklight server is used for SSO to the back-end system.

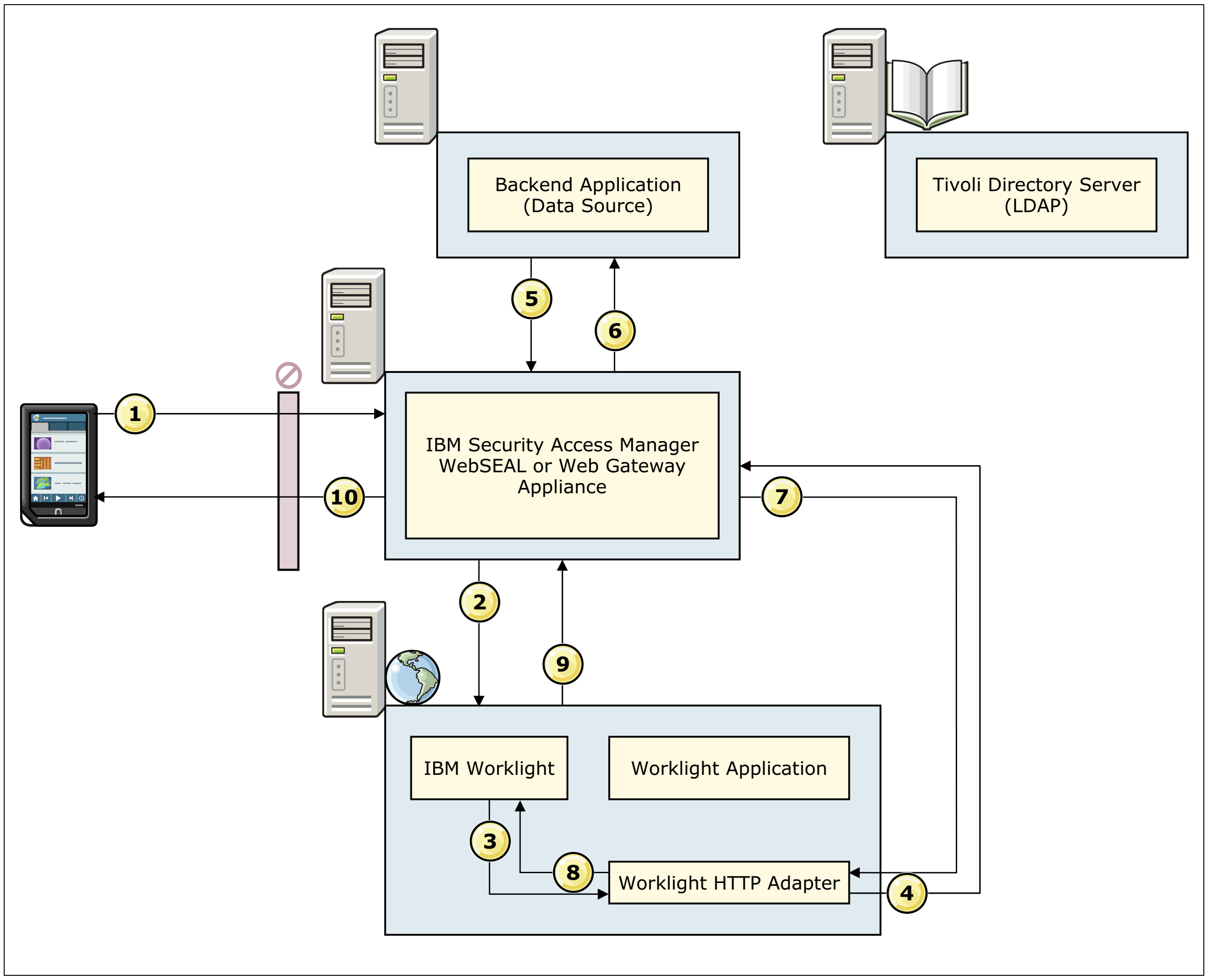

Backend Application (Data Source)
Tivoli Directory Server (LDAP)
5
6
1
IBM Security Access Manager WebSEAL or Web Gateway Appliance
10
7
2
9
IBM Worklight
Worklight Application
3
8
Worklight HTTP Adapter
4

Figure 6-5 Authentication process flow

Authentication includes the following steps:

1. A Worklight mobile app attempts to start a Worklight adapter, and has already passed all of the required security tests. The connection is made using WebSEAL as the web reverse proxy.
2. The user is authenticated, and the request is forwarded to the Worklight server.
3. The Worklight server checks the security test of the adapter and, if successful, enables the invocation.
4. The adapter adds the session ID parameter from the `invoke` request as a cookie HTTP header representing the WebSEAL session ID in the outgoing adapter request.
5. The WebSEAL server validates the passed session cookie, and forwards the request to the back-end server, providing authentication data to achieve SSO as the Worklight user.
6. The back-end system returns the data for the authenticated user to WebSEAL.
7. WebSEAL applies any necessary filtering, and then forwards the response back to the Worklight adapter.

8. The Worklight adapter parses the response, extracts the returned data, and returns the JSON data elements to the Worklight server.
9. The Worklight server forwards the adapter invocation result to WebSEAL.
10. WebSEAL returns the invocation result of the adapter to the Worklight client, to parse and process it as necessary.

Worklight Server adapter

The connection policy of the adapter must reference the WebSEAL server's protocol, domain (host), and port, because all connections are made using WebSEAL as the reverse proxy. The junction path of the adapter request will determine to which back-end data source to connect and authenticate.

A connection policy for WebSEAL is shown in Example 6-27.

Example 6-27 IBM Security Access Manager connection policy for a Worklight adapter

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wl:adapter xmlns:wl="http://www.worklight.com/integration" ...>
   <connectivity>
      <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
         <protocol>https</protocol>
         <domain>isam.fqdn.com</domain>
         <port>443</port>
      </connectionPolicy>
      <loadConstraints maxConcurrentConnectionsPerNode="1"/>
   </connectivity>
</wl:adapter>
```

The adapter running on the Worklight server will access the back-end system via WebSEAL. Therefore, when the connections is opened, the adapter must provide authentication information in a format that WebSEAL is configured to accept. The path can be provided by the Worklight client as an input parameter, or defined in the adapter. The path represents the junction path of the WebSEAL server, which corresponds to the target back-end system.

If the WebSEAL server has been configured to provide its session cookie to the Worklight mobile app on every response, the cookie can be captured in the app, and passed to the adapter as an input parameter for use in the adapter. In Example 6-28, the use of the WebSEAL session ID is shown.

Example 6-28 HTTP adapter Single Sign On using WebSEAL session ID

```
function getBackendResource(webSEALSessionID, path) {

   var webSEALCookieToSend = { "Cookie" : webSEALSessionID };

   var input = {
      method : 'get',
      returnedContentType : 'html',
      path : path,
      headers: webSEALCookieToSend
   };

   return WL.Server.invokeHttp(input);
}
```

If LTPA authentication was used from WebSEAL to Worklight, or an additional security test generated an LTPA token and added it to the authenticated user's identity, the same LTPA token can be used for authentication to WebSEAL when configured appropriately.

Example 6-29 shows how to retrieve the LTPA token from the user's session attributes, and then insert the token for authentication to WebSEAL.

Example 6-29 HTTP adapter SSO using an LTPA token

```
function getBackendResource(path) {

   var attributes = WL.Server.getActiveUser().attributes;
   var webSEALCookieToSend = "LtpaToken=" + attributes.get('LtpaToken');

   var input = {
      method : 'get',
      returnedContentType : 'html',
      headers: {"Cookie": webSEALCookieToSend},
      path : path
   };

   return WL.Server.invokeHttp(input);
}
```

6.5 Integrated security solution

Chapter 2, "Business scenario used in this book" on page 19 examines the requirements of Banking Company A, which needed to provide secure, convenient access to their employees and customers using Worklight mobile apps.

By integrating their Worklight environment with IBM Security Access Manager, they were able to meet the requirements using supported capabilities of both products, with little need for custom development.

6.5.1 Solution architecture

The solution architecture implemented by Banking Company A takes advantage of IBM Security Access Manager for Web and Tivoli Federated Identity Manager to provide the integrated security solution for Worklight.

Figure 6-6 shows the solution architecture.

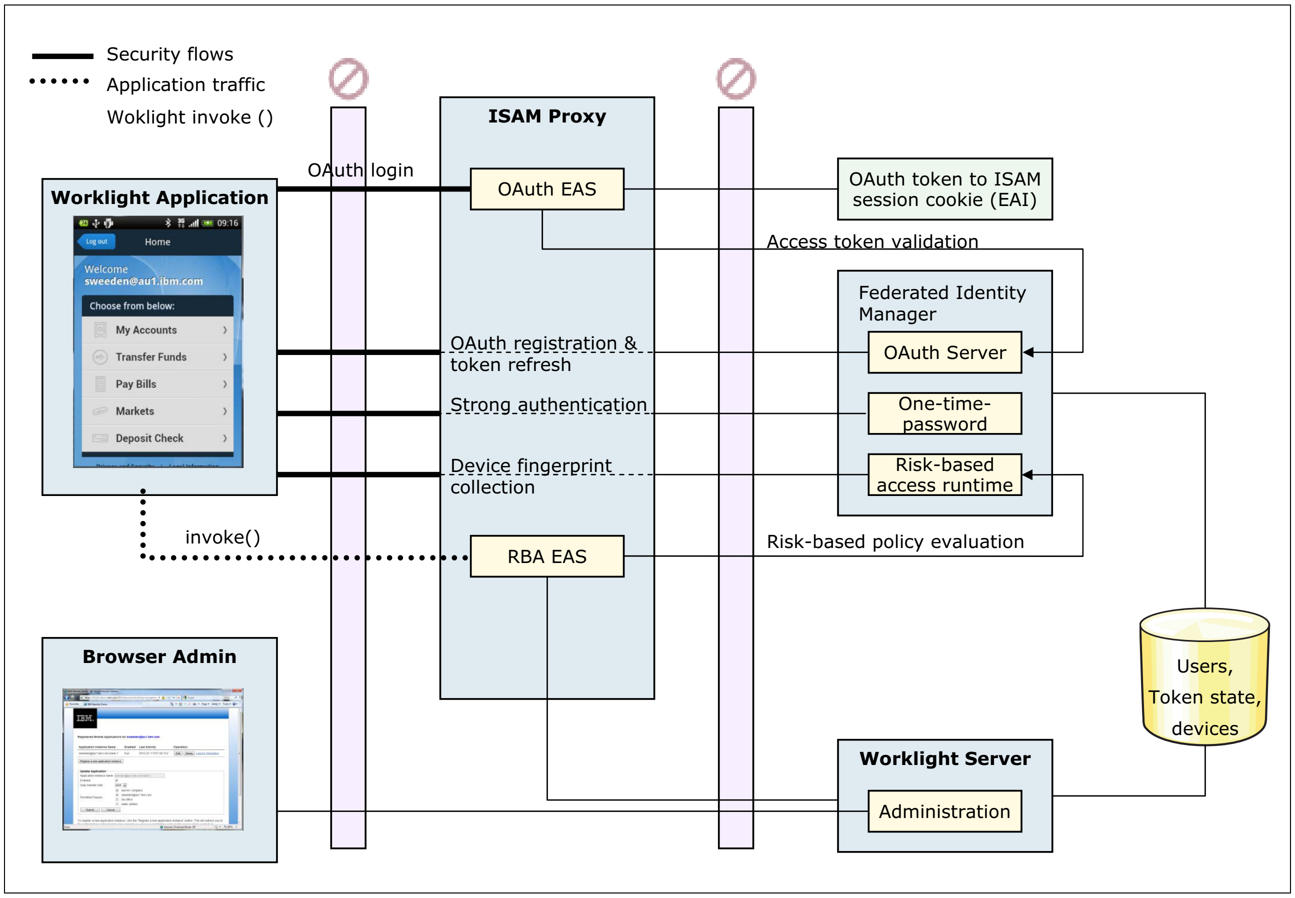


Figure 6-6 IBM Security Access Manager integration using OAuth authentication and RBA

IBM Security Access Manager authentication

By taking advantage of the integration options with IBM Security Access Manager, Banking Company A was able to simplify directory integration, delegate user authentication, and propagate user identity and attributes to the Worklight server.

Using an OAuth pattern as demonstrated in 6.2.2, "Tivoli Federated Identity Manager OAuth single sign-on" on page 111, Banking Company A provided the capability for their customers to add, suspend, restore, and delete mobile device access to their accounts without needing to show user names and passwords. Customers can manage their mobile device access through their traditional internet banking or, if necessary, have a bank employee disable access on a per-device basis.

The following custom IBM Security Access Manager components are configured to use the OAuth pattern:

- OAuth EAS provides an authorization policy enforcement point for sessionless OAuth transactions, such as registration and token validation. The Worklight mobile app calls the Worklight Server adapter to complete OAuth process flow.
- OAuth EAI performs token mediation, allowing the exchange of an OAuth access token for a valid WebSEAL session for the corresponding user. This enables subsequent requests to the Worklight server, to take advantage of the user identity and attributes for the user authenticated to IBM Security Access Manager.

In addition, Banking Company A was able to implement SSO to back-end data systems using the identity of the Worklight mobile app user, as described in 6.4, “Worklight adapter single sign-on” on page 132.

Risk-based access

Strong authentication using OTP support was implemented using the integration described in 6.3, “Risk-based access” on page 121. This provides reassurance and secure access with identity validation, to detect and enforce transfer policies for banking transactions.

The following IBM Security Access Manager components are configured for the Strong Authentication with OTP:

- WebSEAL acts as the PEP for authorization decisions, acts as the POP trigger for fine-grained authorization, and provides the authentication and OTP framework. After successful authentication, it then provides SSO to Worklight.
- RTSS EAS provides the runtime capability to check for fine-grained authorization access policy for resources protected by WebSEAL, and delegates as required.
- RBA PIP for Worklight adapters extracts indexed values from an HTTP-encoded JSON parameter set of Worklight adapters. These values can then be used in access policy for evaluation.
- Tivoli Federated Identity Manager acts as the policy decision point (PDP), and hosts the authorization service to evaluate context-based access policy, combining attributes passed to it from the RTSS EAS, RBA PIP for Worklight adapters, and the attribute collector service.
- OTP EAI adds the capability for challenging for and validating OTP which provides step up authentication for high value transactions.

The RBA policies for Banking Company A are easily applied across all of the Worklight adapters used in their mobile apps, and the context-based attribute decisions provide fine-gained authorization and risk scoring to enforce policy decisions in the DMZ.

6.5.2 Conclusion

Banking Company A deployed an IBM Security Access Manager for Cloud and Mobile solution, enabling them to achieve their mobile business drivers:

- Reusing their secure platform foundation
- Increasing staff productivity
- Providing secure and easy access for customers

They combined the features and strengths of IBM Security Access Manager for Web and IBM Tivoli Federated Identity Manager to provide enhanced threat detection and prevention capabilities, policy governance, and enforcement for the Worklight mobile environment.

7

Integration with IBM WebSphere DataPower

The IBM WebSphere DataPower Integration Appliance is a security and integration gateway appliance, built for simplified deployment and hardened security, bridging multiple protocols and performing conversions at wire speed.

The WebSphere DataPower Integration Appliance XI52 is part of the DataPower Appliance family, which also includes service gateways, business-to-business (B2B) gateways and caching technology.

The IBM WebSphere DataPower Service Gateway XG45 is a functional subset of the DataPower Integration Appliance XI52. Therefore, from a security perspective, much of this chapter can conceptually also be applied to the DataPower Service Gateway XG45.

This chapter demonstrates how the DataPower Integration Appliance XI52 can be used in the DMZ (a firewall configuration for securing local area networks) of your enterprise to protect Worklight mobile application (app) traffic.

Note: A sample Worklight application (`simpleProject.zip`) and a sample DataPower configuration (`DPMobile.zip`) have been provided with this chapter.

See Appendix A, "Additional material" on page 159 for information about how to obtain these sample modules.

7.1 Introduction to the DataPower Appliances

For service-oriented architecture (SOA), the IBM WebSphere DataPower SOA Appliances help secure, integrate, and optimize access to web, mobile, and application programming interface (API) workloads.

The main advantage of the DataPower Appliances is that they are easy to integrate into a network architecture. In addition, you can use the virtual appliance to accomplish the same basic functions of the hardware-based appliance. The only difference is that you will not have the same level of security assurance as the hardware.

When used as the security layer provider, the DataPower Appliances can run many critical security tasks by offloading them from the application server.

The DataPower SOA Appliances provide the following key features:

- Enable new workloads for securing mobile, web, and API management, consolidating and simplifying enterprise infrastructure
- Provide authorization, authentication, and auditing (known as AAA) support
- Provide application-level security as an integral part of the user interaction
- Help customers meet compliance requirements, serving as a governance policy enforcement point
- Can implement an enterprise single sign-on (SSO) functions using Lightweight Third Party Authentication (LTPA) tokens
- Simplify integration to multiple back-end applications, supporting a wide array of protocols, including protocol bridging
- Provide capabilities to enable Representational State Transfer (REST) access to back-end SOA services
- Optimize back-end systems via robust and application-aware load balancing support
- Improve total cost of ownership (TCO), because it is configurable and no programming is needed
- Unload Hypertext Transfer Protocol Secure (HTTPS) session handling from the web server
- Act as a web service, mobile web service, and Extensible Markup Language (XML) firewall
- Also available as a virtual appliance

For a complete listing of WebSphere DataPower SOA Appliance products, see the following website:

http://www.ibm.com/software/products/us/en/datapower/

7.2 DataPower Integration Appliance XI52 overview

In the discussion of integrating the DataPower Appliances and Worklight, the DataPower Integration Appliance XI52 is used as an example. However, because the DataPower Service Gateway XG45 is a functional subset of the DataPower Integration Appliance XI52 relative to security, all of the descriptions in this section also apply to the DataPower Service Gateway XG45.

The DataPower Integration Appliance XI52 is a network appliance that is built for web service deployments, governance, quick integrations, and hardened security.

The XI52 protects against several vulnerabilities by acting as a proxy. It includes many essential security functions:

- Web service access control authentication
- AAA
- XML encryption and digital signature
- Web Services Security (WS-Security)
- Content-based routing

As shown in Table 7-1, the DataPower Integration Appliance XI52 offers an advanced approach to threat reduction and security for web and mobile access.

Table 7-1 The DataPower Integration Appliance XI52 security features

Feature	Description
Web application firewall and gateway	► Protects against XSS, SQL injection, XML vulnerabilities. ► Offers extended security functions beyond those of an XML firewall. The extended functions include web service access control (AAA), XML encryption and digital signature, WS-Security, and content-based routing. ► Supports JavaScript Object Notation (JSON) schema validation and JSON payload protection. ► Supports HTTP header (including cookie) signature and encryption.
XML and JSON denial-of-service protection	► Validates incoming requests and documents malformed and malicious traffic (for example, use the schema validation to make sure the message is valid). ► Gain access to valuable post-attack forensics. ► Controls the low-byte XML and JSON messages that can bypass your traditional perimeter protection. ► Deploys service level monitor (SLM), service level agreement (SLA) to regulate the incoming messages.
Field-level message security	Selectively shares information of entire messages or of individual XML fields.
Access control for web services	Enables secure access to web services-based applications for your internal and external clients.
Fine-grained authorization	Offers fine-grained authorization that interrogates individual requests to determine whether they can be allowed through.

Banking Company A wants to connect its mobile apps to its enterprise data, and needs a mobile gateway for security. Banking Company A can use the DataPower Appliances to perform tasks including validation, authentication, and authorization.

For Banking Company A, which has a Worklight server, the DataPower Appliances can offer the following functions:

- Provide enhanced form-based authentication support for easy and quick integration with Worklight applications running on mobile devices
- Include a ready-to-use configuration pattern as a reverse proxy and security policy enforcement point in front of the Worklight Server
- Provide fine-grained authorization and authentication with a centralized policy enforcement
- Provide data transformation and enable connectivity

7.3 Integrating WebSphere DataPower with Worklight

In a typical business-to-consumer (B2C) scenario, security requirements are more demanding, because you must expose your information technology (IT) infrastructure to external devices that are located out of your intranet zone.

With mobile devices, there is a big variation regarding the device models, mobile operating systems, and regions from where the requests come. In addition, your user base will be more volatile, and it is not possible to control whether a device is rooted or not safe. In such scenarios, it makes sense to have a component acting as the security gateway front-end to your network and infrastructure, a first point of contact enforcing security regarding mobile app access to your internal corporate systems.

Making the Worklight server accessible to external user-owned devices opens your IT infrastructure to inbound connections that are potential security exposures if not controlled properly. So a secure gateway must be introduced to protect the existing IT infrastructure, and restrict access to only those back-end services that are required by your mobile app.

Although a B2C scenario exposes additional security risks, a business-to-employee (B2E) scenario also exposes security risks. As you mobile-enable your enterprise, there are several aspects of security to consider. When protecting mobile app traffic that is coming from your customer and employee devices into your network, you want to protect the traffic from being altered, authenticate these users, and authorize their access to applications.

This is where you can take advantage of the security features offered by the DataPower Integration Appliance XI52 to protect access to your enterprise systems. Using this appliance as the secure gateway creates what is essentially a self-contained security infrastructure for the mobile apps.

7.3.1 Patterns for integration

Enterprise topologies generally include designating different zones of protection, so that specific processing can be secured and optimized. There are several ways the DataPower Integration Appliance can be used in the DMZ, or in other zones within your network, to protect enterprise resources.

As you start to build out Worklight applications to be delivered to the devices of your customers and employees, the following patterns can be applied to mobile traffic.

The DMZ

The following patterns can be used in the DMZ:

- Reverse Proxy Pattern. The DataPower Integration Appliance is a front-end reverse proxy and security gateway for your mobile apps, built with and deployed on a Worklight Server.
- REST Service Façade Pattern. The DataPower Integration Appliance protects REST endpoints in the DMZ without using Worklight adapters for integration.

Intranet

The following patterns can be used in the intranet zone:

- Mediation Pattern. The DataPower Integration Appliance provides protocol and message transformation, and security for enterprise services that are integrated via Worklight adapters, and that enable connectivity to your existing SOA infrastructure.
- Federated SSO with Authorization Pattern. The DataPower Integration Appliance authorizes authenticated access to OAuth protected resources.

The rest of this chapter focuses on the Reverse Proxy Pattern to integrate the DataPower Integration Appliance security capabilities with the Worklight server.

7.3.2 Reverse Proxy Pattern

The DataPower Integration Appliance functions as a network security gateway, ensuring only authorized access from the mobile app to the internal corporate systems.

In the following sections, the DataPower Multi-Protocol Gateway (MPGW) service will be used as a proxy and secure access for Worklight mobile apps. HTTP basic authentication or HTTP form-based login will be used between the mobile client and the DataPower Integration Appliance which, in turn, will authenticate the user and generate an LTPA SSO token to be used by the back-end Worklight server running on a WebSphere Application Server run time. Figure 7-1 illustrates a typical request flow.

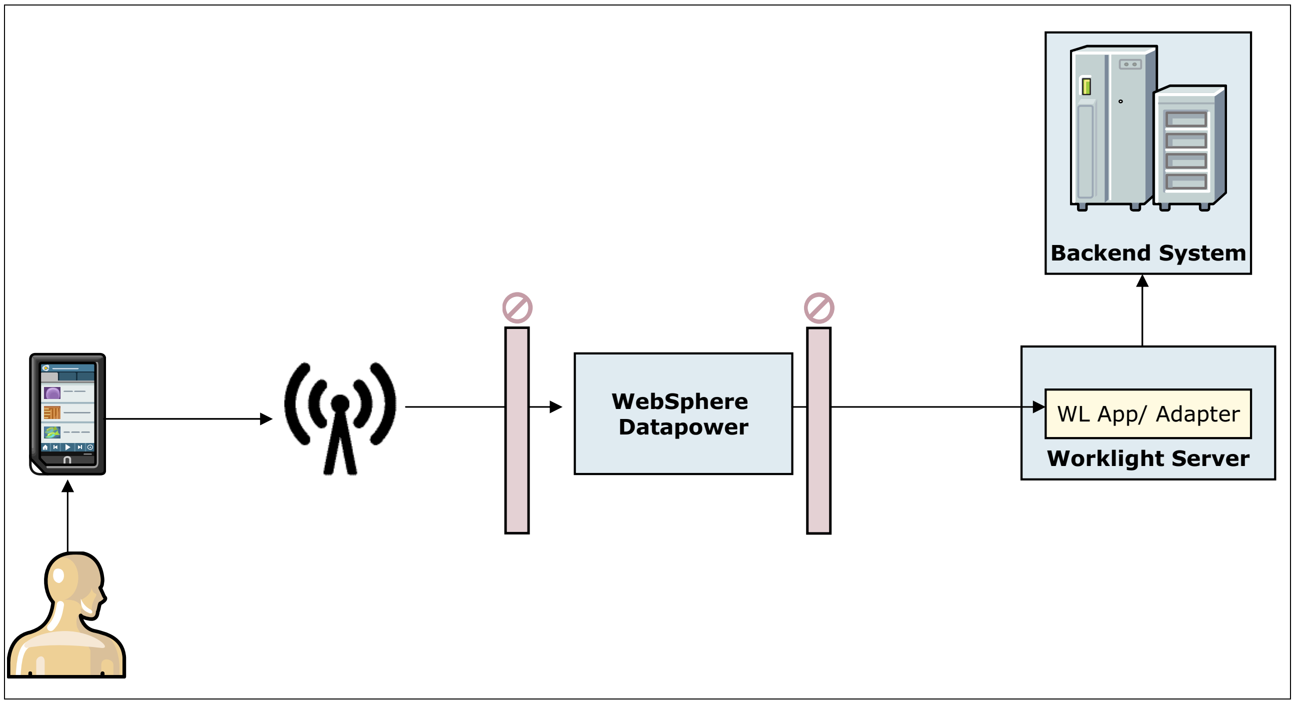


Figure 7-1 DataPower Appliances authenticate the user and protect the communication to the Worklight Server

When a user sends a request from the mobile app, the request is routed to the DataPower Appliance, which is in the DMZ. If the request does not have user authentication credentials associated with it, the appliance challenges the mobile app to supply the user credentials (for example, user ID and password) and then authenticates the request against a user registry.

When the request is authenticated, the appliance injects an LTPA token or HTTP header (identifying the user) into the request, and sends that request downstream to the Worklight server that has been configured to trust that the DataPower Appliance can assert the user's identity. See Figure 7-2 for a a typical interaction between the user and application, and the Worklight server.

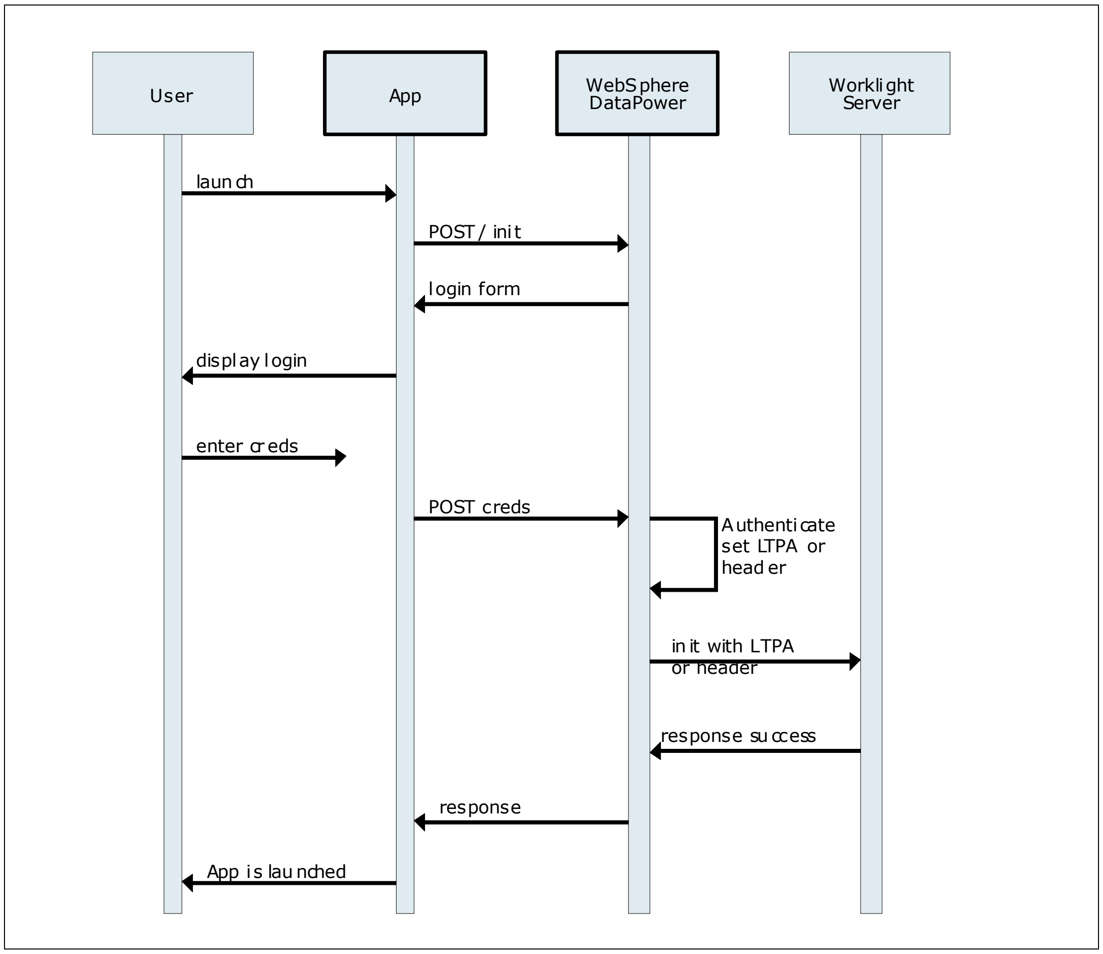


Figure 7-2 A typical interaction between the user/application and the Worklight server

To establish this pattern, follow these steps:

1. Install and configure a Worklight environment.
2. Test the install with a simple application without the DataPower Integration Appliance acting as the reverse proxy, to make sure your application logic works correctly.
3. Configure a MPGW on the DataPower Integration Appliance to act as a proxy for the mobile app. The configuration of the gateway for the basic authentication or HTML form-based login is described in the following sub-steps:
   - Use basic authentication for user authentication with an AAA policy and, if the user is successfully authenticated, generate an SSO LTPA token for the Worklight server running on WebSphere Application Server.

– Use HTML form-based login with an AAA policy and, if the user is successfully authenticated, generate an SSO LTPA token for the Worklight server running on WebSphere Application Server.

4. Testing with a reverse proxy involves the following actions on the Worklight server and the mobile app:

   – Updating the security configuration on the Worklight server to enable validation of the LTPA token that will be sent by the DataPower Appliance.

   – Updating the mobile app mobile security test configuration to use the LTPA authentication method, as configured in the Worklight server. This configuration requires the application to authenticate immediately upon startup.

     In addition, the mobile app also has to be updated to handle the authentication challenge that will be generated by the DataPower Appliance.

7.3.3 Configuring the Worklight Server

The Worklight server used by the mobile app(s) has to be configured to accept the LTPA token that will be sent after the request has been authenticated by the DataPower Appliance. The `authenticationConfig.xml` file on the server must be modified to include `WASLTPARealm`, its login module, and security test. Use the `WASLTPARealm` to indicate that all of the applications within this realm will use LTPA tokens for their SSO needs, as shown in Example 7-1.

Example 7-1 authenticationConfig.xml

```
<securityTests>
   <mobileSecurityTest name="WASTest-securityTest">
      <testDeviceId provisioningType="none"/>
      <testUser realm="WASLTPARealm"/>
   </mobileSecurityTest>
   <webSecurityTest name="WASTest-web-securityTest">
      <testUser realm="WASLTPARealm"/>
   </webSecurityTest >
</securityTests>

<realms>
   <!-- For WebSphere -->
   <realm name="WASLTPARealm" loginModule="WASLTPAModule">

<className>com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator</className>
      <parameter name="login-page" value="/login.html"/>
      <parameter name="error-page" value="/loginError.html"/>
   </realm>
</realms>

<loginModules>
   <!-- For WebSphere -->
   <loginModule name="WASLTPAModule">
      <className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
   </loginModule>
</loginModules>
```

Note: The `login.html` and **`loginError.html`** files should already exist in the root of the Worklight server or console web archive (WAR) file. Furthermore, the `login.html` file should contain the string `j_security_check`, because the challenge handler code will search for that string upon initial initialization.

Optionally, you should include a “success” HTML file that is requested upon successful authentication.

The Worklight server or enterprise console application must be restarted after the configuration changes are made.

7.3.4 Configuring the mobile application

The mobile app has to be configured to accept and respond to the authentication challenge from the DataPower Appliance. Worklight provides the mobile app with a challenge handler mechanism that, in this case, will be used to ask the user for a user ID and password, and then submit these user credentials to the DataPower Appliance for authentication.

On the mobile app, a set of JavaScript functions corresponding to the Worklight client-side challenge handler APIs must be defined to handle challenges related to `WASLTPARealm`. See Example 7-2 for a JavaScript challenge handler that can be used by the mobile app.

Example 7-2 JavaScript challenge handler used by the mobile app

```
function showLoginScreen() {
   $("#index").hide();
   $("#authPage").show();
}

function showMainScreen() {
   $("#authPage").hide();
   $("#index").show();
}

var myChallengeHandler = WL.Client.createChallengeHandler("WASLTPARealm");
var lastRequestURL = '/worklight/apps/services/www/SimpleApp/mobilewebapp/';


myChallengeHandler.isCustomResponse = function(response) {
   console.log(response.responseText);
   // A normal login form has been returned
   var findError = response.responseText.search("Form login authentication failure
| Login invalid.");
   if (findError >= 0) {
      return true;
   }
   // A normal login form has been returned
   var findLoginForm = response.responseText.search("j_security_check");
   if (findLoginForm >= 0) {
      return true;
   }
   // This response is a worklight server response, handle it normally
   return false;
};
```

```
myChallengeHandler.handleChallenge = function(response) {
   showLoginScreen();
};

myChallengeHandler.handleFailure = function(response) {
   console.log("Error during WL authentication.");
};

myChallengeHandler.submitLoginFormCallback = function(response) {
   var isCustom = myChallengeHandler.isCustomResponse(response);

   if (isCustom) {
      myChallengeHandler.handleChallenge(response);
   } else {
      // hide the login screen, we are logged in
      showMainScreen();
      myChallengeHandler.submitSuccess();
   }
};

// When the login button is pressed, submit a login form
$("#loginButton").click(function() {
   var reqURL = "/j_security_check";

   var options = {
      method : "POST"
   };
   options.parameters = {
      j_username : $("#username").val(),
      j_password : $("#password").val(),
      originalUrl : lastRequestURL,
      login : "Login"
   };
   options.headers = {};
   myChallengeHandler.submitLoginForm(reqURL, options,
myChallengeHandler.submitLoginFormCallback);
});
```

Note: This code can be placed in a separate JavaScript file (for example, `auth.js`) and referenced from your main HTML file.

The form parameters submitted (**`j_username`**, **`j_password`**, and URL **`j_security_check`**) must match the expected parameters in the DataPower HTML Forms Login policy object.

The *lastRequestURL* variable references an HTML file from the mobile web version of the Worklight project (you will need to change it to match your environment). If you created a "success" HTML file and deployed it into the Worklight WAR file, you could reference that file instead. After successful authentication of the credential by the DataPower Integration Appliance, it will re-issue the request based on the URL in the **`originalURL`** parameter.

This JavaScript challenge handler example requires the mobile app to display a form to the user so that the user ID and password information can be collected.

Example 7-3 shows a simple example of a mobile app form.

Example 7-3 Mobile app Login form

```
<div id= "authPage">
   You are not authenticated.
   <div id= "loginForm">
      Username:<br/>
      <input type= "text" id= "username"autocorrect= "off"autocapitalize= "off" 
value= "admin"/><br/>
      Password:<br/>
      <input type= "password" id= "password"autocorrect= "off"autocapitalize= 
"off" value= "admin"/><br/>
      <input type= "button" id= "loginButton" value= "Login" />
   </div>
</div>

<div id= "index">
   You are authenticated.
</div>

<script src="js/auth.js"></script>
```

In addition to the challenge handler and the login form, the mobile app has to be configured to participate in the Worklight security mechanisms by indicating which Worklight security test will govern this mobile app's requests. After authentication by the DataPower Appliance, the mobile app requests will have an LTPA token, created by the appliance, sent along with the requests to the Worklight server. Therefore the mobile app must be configured to use the Worklight security test that handles `WASLTPARealm`.

This is accomplished by adding a `securityTest` attribute to the `application-descriptor.xml` file of the Worklight application project (using Worklight Studio) to use the `WASTest-securityTest` security test that was created earlier in the `authenticationConfig.xml` file on the Worklight server. Example 7-4 shows an iPad example.

Example 7-4 Securing an iPad application

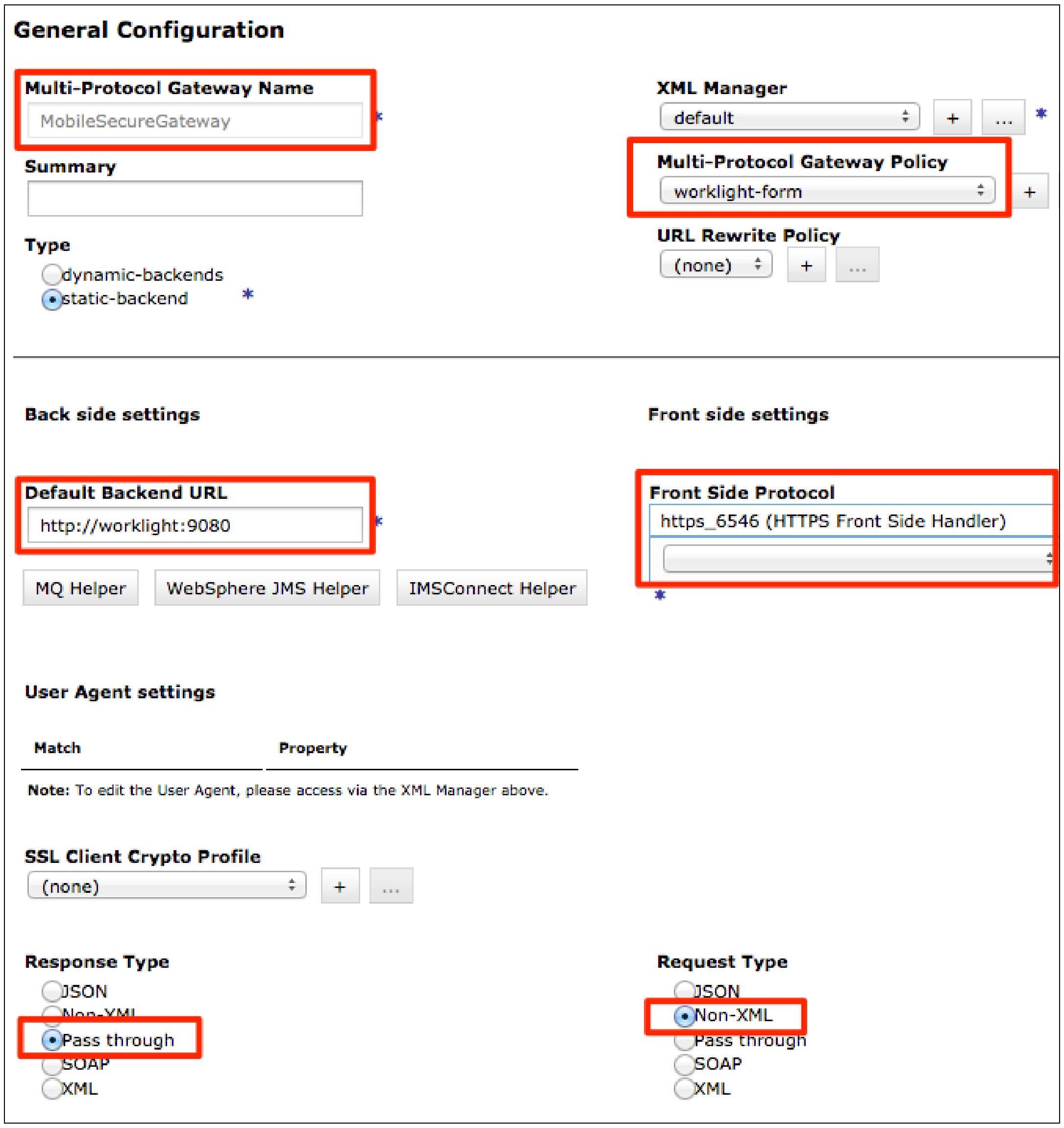
```
<ipad bundleId= "com.Datapower" securityTest= "WASTest-securityTest" version= 
"1.0">
   <worklightSettings include= "true"/>
   <security>
      <encryptWebResources enabled= "false"/>
      <testWebResourcesChecksum enabled= "false" ignoreFileExtensions= "png, jpg, 
jpeg, gif, mp4, mp3"/>
   </security>
</ipad>
```

7.3.5 Configuring the multiprotocol gateway on the DataPower Appliance

Next, configure a MPGW to end the secure network connection, and also to enforce the authentication of an application. The steps and illustrations represent a subset of the configuration screens found in the DataPower Integration Appliance GUI that are necessary to configure the MPGW for a mobile app. To configure the MPGW service, the front-side and the back-end URLs have to be configured.

General configuration and front-side handler

Start with the General Configuration tab. Figure 7-3 highlights the items that you will select.



General Configuration

Multi-Protocol Gateway Name: MobileSecureGateway *

Summary:

Type: dynamic-backends [✓ static-backend] *

XML Manager: default *

Multi-Protocol Gateway Policy: worklight-form

URL Rewrite Policy: (none)

Back side settings

Default Backend URL: http://worklight:9080 *

Front side settings

Front Side Protocol: https_6546 (HTTPS Front Side Handler) *

User Agent settings

Match	Property

Note: To edit the User Agent, please access via the XML Manager above.

SSL Client Crypto Profile: (none)

Response Type: JSON Non-XML [✓ Pass through] SOAP XML

Request Type: JSON [✓ Non-XML] Pass through SOAP XML

Figure 7-3 General Configuration tab

These selections include the following details:

- Multi-Protocol Gateway Name. Provide a name for your gateway.
- Multi-Protocol Gateway Policy. The Policy tab enables additional configuration. In this case, select the plus sign (**+**) to add a new processing policy (**worklight-form**), which will also contain the AAA policy.
- Default Backend URL. Specify the address and port of the Worklight server hosted on WebSphere Application Server. Given the user credentials, an LTPA token will be sent to the back-end. Consider using a Secure Sockets Layer (SSL) port. However, for this example, an HTTP port is used.
- Front Side Protocol. For the type of use case described, we use an HTTPS handler. For this style of interaction, the user credential will be passed between the mobile app and the

DataPower Appliance. Therefore, the transport should be protected using the SSL protocol.

- Set Response Type. Select **Pass through**. This enables all HTTP traffic to pass through the appliance.
- Set Request Type. Select **Non-XML**. This enables all HTTP requests to be handled by appliance.

Note: You will create the MPGW policy later in this section.

When creating an HTTPS Front Side Handler, there are additional parameters to configure (see Figure 7-4 and Figure 7-5 on page 151):

- Name. Provide a name for the handler (HTTPS).
- Port Number. Choose an available port (note: this port is shown in the DMZ).
- Allowed Methods and Versions. Enable Get Method. This enables support for HTTP `Get`.
- SSL Proxy. Specify an SSL Reverse Proxy profile. This provides the identity for the SSL server on the DataPower Appliance.

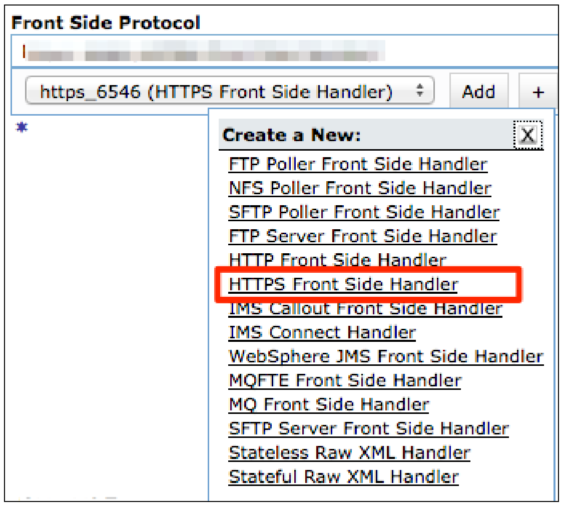
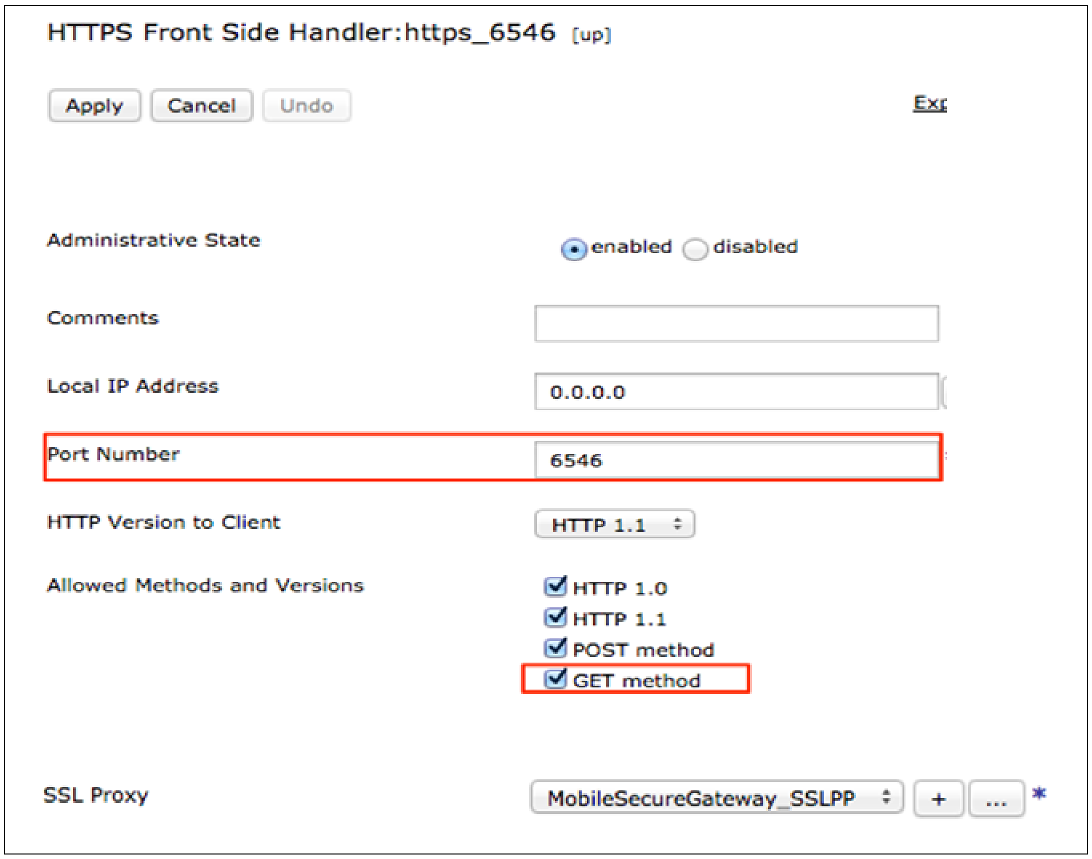

Front Side Protocol
https_6546 (HTTPS Front Side Handler)
Add
+
Create a New:
X
FTP Poller Front Side Handler
NFS Poller Front Side Handler
SFTP Poller Front Side Handler
FTP Server Front Side Handler
HTTP Front Side Handler
HTTPS Front Side Handler
IMS Callout Front Side Handler
IMS Connect Handler
WebSphere JMS Front Side Handler
MQFTE Front Side Handler
MQ Front Side Handler
SFTP Server Front Side Handler
Stateless Raw XML Handler
Stateful Raw XML Handler

Figure 7-4 Select HTTPS (SSL) Front Side Handler


HTTPS Front Side Handler:https_6546 [up]

Administrative State	[✓ enabled] disabled
Comments	
Local IP Address	0.0.0.0
Port Number	6546
HTTP Version to Client	HTTP 1.1
Allowed Methods and Versions	✓ HTTP 1.0 ✓ HTTP 1.1 ✓ POST method ✓ GET method
SSL Proxy	MobileSecureGateway_SSLPP

Figure 7-5 Configure HTTPS Front Side Handler

Before creating the MPGW service policy, you first need to create the AAA policy objects, which will be referenced from the MPGW service policy.

AAA policy

Creating an AAA policy involves a series of actions that become part of the policy rules:

- Extract Identity
- Authenticate
- Map Identity
- Extract Resource
- Map Resource
- Authorize
- Audit and Post-Process.

Only the actions needed for a specific AAA policy are selected.

Figure 7-6 shows an overview of the AAA processing flow that is run by the DataPower Appliance.

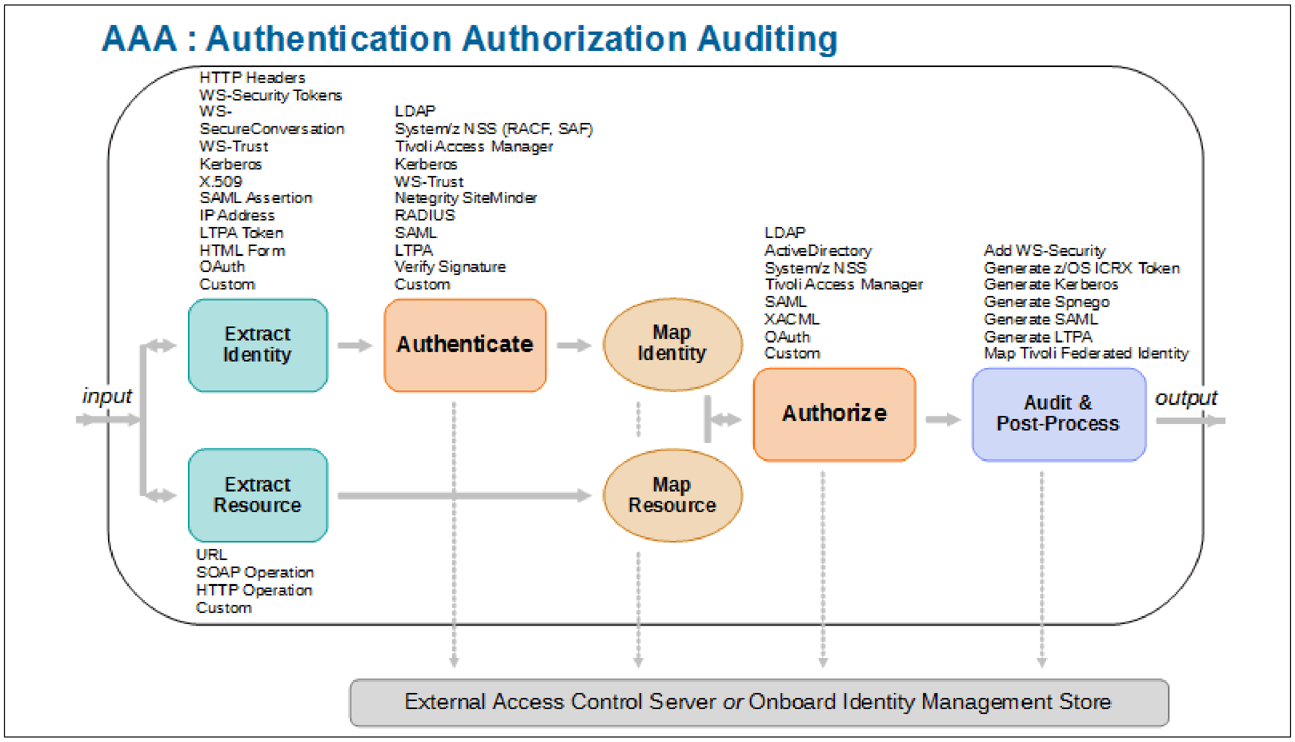


Figure 7-6 AAA processing flow

In this use case, several different AAA polices are created:

- BasicAuth2LTPA. Extract the user credentials from an HTTP basic authentication header and create an LTPA token if the user credentials are authenticated.
- Form2LTPA. Extract the user credentials from a Login form and create an LTPA token if the user credentials are authenticated.
- VerifyLTPA. Verify that the LTPA token is valid.

You can create these AAA policies by entering **AAA Policy** in the vertical navigation bar and clicking the resulting entry (Figure 7-7).

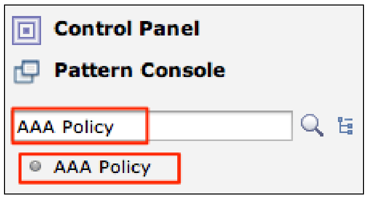
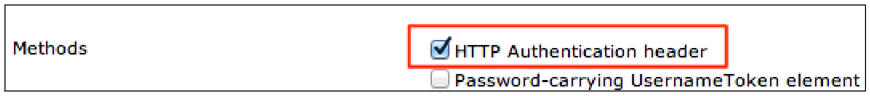


Figure 7-7 AAA Policy

BasicAuth2LTPA

To extract the user credentials from an HTTP basic authentication header, and then create an LTPA token if the user credentials are authenticated, follow these steps:

1. Extract Identity. For Method, select **HTTP Authentication header** (as shown in Figure 7-8 on page 153).


Methods

☑ HTTP Authentication header
☐ Password-carrying UsernameToken element

Figure 7-8 HTTP basic authentication

2. Authenticate. Choose the authentication method. If the Worklight server running on WebSphere Application Server is using LDAP, configure the same LDAP here.

> **Tip:** For testing purposes, you can use the DataPower Appliance AAA security file located at `store:///AAAInfo.xml`, or create a new AAA policy XML file. Make sure that the <aaa:Output Credential> tag uses the appropriate format for WebSphere Application Server, for example:
> `file-based registry: uid=<user>,o=defaultWIMFileBasedRealm`

3. Extract Resource. For Resource Information, select **URL Sent by Client**.
4. Authorize. Allow any authenticated client.
5. Postprocess. Perform the following actions:
   a. Generate an LTPA Token.
   b. Specify **LTPA Token Expiry**, **LTPA Key File**, and **LTPA Key File Password**.

Form2LTPA

To extract the user credentials from a Login form, and then create an LTPA token if the user credentials are authenticated, follow these steps:

1. Extract Identity. For Method, select **HTML Form-based Authentication**, and create an HTML forms-based login policy (Figure 7-9).

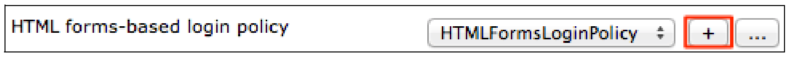
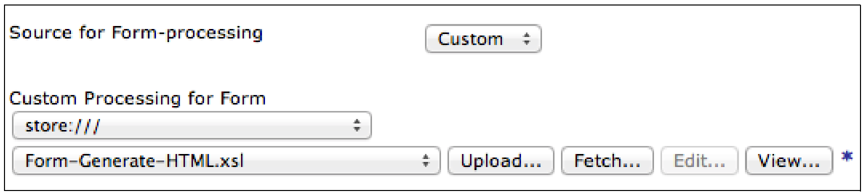

HTML forms-based login policy: HTMLFormsLoginPolicy

Figure 7-9 Form-based authentication

> **Tip:** See the following IBM developerWorks® article for more detail on how to create an HTML Form Policy:
> `http://www.ibm.com/developerworks/websphere/library/techarticles/1208_poon1/1208_poon1.html#html`
>
> One difference between the process described in that article and this example is the source for the login and error pages.

2. Select Source for Form-processing. Select **Custom**.
3. Configure Custom Processing for Form. Choose `store:///Form-Generate-HTML.xsl` (Figure 7-10).


Source for Form-processing: Custom

Custom Processing for Form
store:///
Form-Generate-HTML.xsl

Figure 7-10 Custom form-processing in the HTML form policy

Note: Leave the Login form properties at their default value of `j_username`, `j_password`, and `/j_security_match`, which will match the parameters submitted by the mobile HTML login form.

4. Authenticate. Choose the authentication method. If the Worklight server running on WebSphere Application Server is using LDAP, configure the same LDAP here.

Tip: For testing purposes, you can use the DataPower AAA security file located at `store:///AAAInfo.xml`, or create a new AAA policy XML file. Make sure that the <aaa:Output Credential> tag uses the appropriate format for WebSphere Application Server, for example:

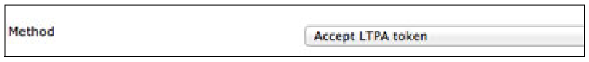
```
file-based registry: uid=<user>,o=defaultWIMFileBasedRealm
```

5. Extract Resource. For Resource Information, select **URL Sent by Client**.
6. Authorize. Allow any authenticated client.
7. Postprocess. Perform the following actions:
   a. Generate an LTPA Token.
   b. Specify **LTPA Token Expiry**, **LTPA Key File**, and **LTPA Key File Password**.

VerifyLTPA

To verify that the LTPA token is valid, follow these steps:

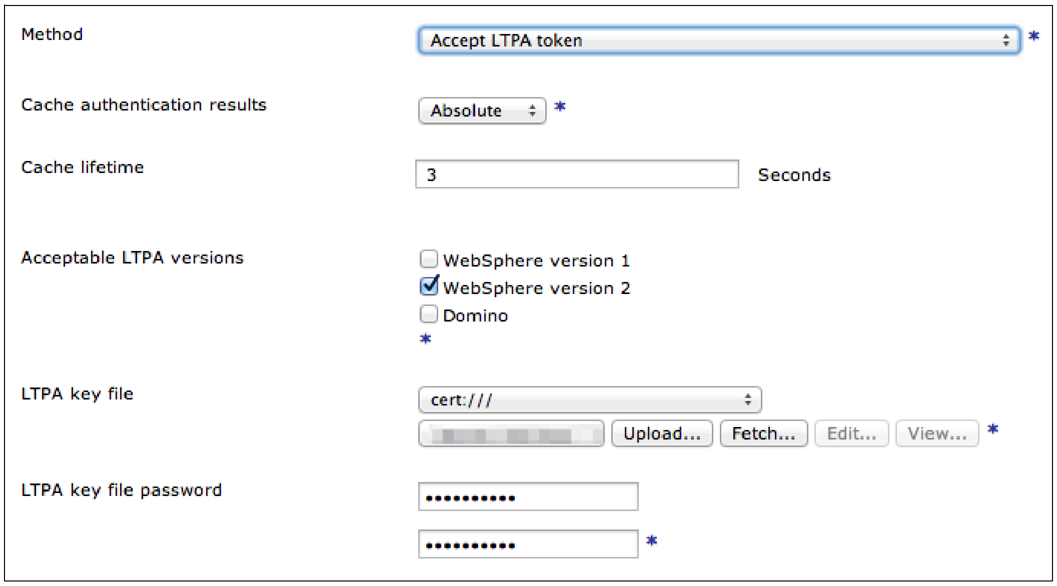
1. Extract Identity. For Method, select **Accept LTPA token** (Figure 7-11).



Method: Accept LTPA token

Figure 7-11 LTPA Token Authentication

2. Authenticate. Choose **Accept an LTPA Token**, and specify **LTPA Token Versions**, **LTPA Key File**, and **LTPA Key File Password** (Figure 7-12).



Method	Accept LTPA token *
Cache authentication results	Absolute *
Cache lifetime	3 Seconds
Acceptable LTPA versions	☐ WebSphere version 1 ☑ WebSphere version 2 ☐ Domino *
LTPA key file	cert:/// *
LTPA key file password	••••••••••• ••••••••••• *

Figure 7-12 Verify an LTPA token

3. Extract Resource. For Resource Information, select **URL Sent by Client**.
4. Authorize. Allow any authenticated client.

Processing policy

The next step is to configure the processing policies to be applied to the traffic. To do so, follow these steps:

1. Select the General tab, and then click the plus sign (+) and select **Multi-Protocol Gateway Policy**.
2. In the policy section, configure a set of rules to be applied based on what actions the reverse proxy needs to enforce.

   The DataPower Appliance enables the administrator to configure a policy, name it, and then reuse it. This policy will handle both mobile web and mobile app traffic.

   The following authentication options can be used in the processing policy:

   – Option 1 is HTTP basic authentication.
   – Option 2 is forms-based authentication.

Option 1: Use HTTP basic authentication

To use HTTP basic authentication, follow these steps:

1. Create a Policy Name. Enter a policy name (`worklight-basicauth`), and click **Apply Policy**.
2. Click **New Rule** (see Figure 7-13 on page 156). Repeat this for the following rules, in order:

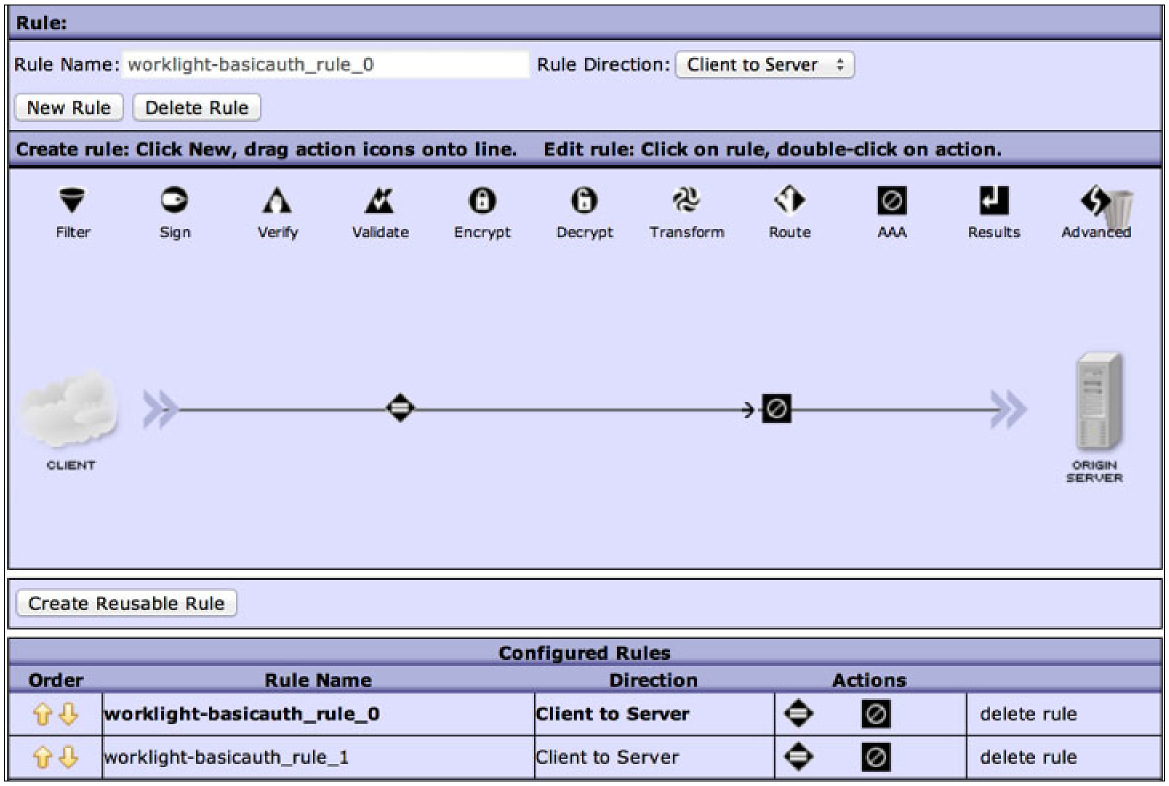
   a. Rule 1. Because the policy will be applied to each request, the order of the rules needs to ensure that we first verify if an LTPA token exists in the HTTP request (`worklight-basicauth_rule_0`). If there is no token, move to the next rule and authenticate the user:

      i. Direction: “Client to Server”
      ii. Match: type = http, HTTP Header Tag = Cookie, HTTP Value Match=*LtpaToken*
      iii. AAA: VerifyLTPA
      iv. Input: INPUT
      v. Output: OUTPUT

   b. Rule 2. Handle the authentication of the user if the LTPA token does not exist (`worklight-basicauth_rule_1`):

      i. Direction: “Client to Server”
      ii. Match: type = url, match pattern = *
      iii. AAA: BasicAuth2LTPA
      iv. Input: INPUT
      v. Output: OUTPUT

Figure 7-13 shows the form for creating or editing a rule.


Rule:
Rule Name: worklight-basicauth_rule_0
Rule Direction: Client to Server
New Rule
Delete Rule
Create rule: Click New, drag action icons onto line.
Edit rule: Click on rule, double-click on action.
Filter
Sign
Verify
Validate
Encrypt
Decrypt
Transform
Route
AAA
Results
Advanced
CLIENT
ORIGIN SERVER
Create Reusable Rule

Configured Rules

Order	Rule Name	Direction	Actions	
⇧⇩	**worklight-basicauth_rule_0**	**Client to Server**		delete rule
⇧⇩	worklight-basicauth_rule_1	Client to Server		delete rule

Figure 7-13 MPGW Style Policy for HTTP Basic Authentication Support

Option 2: Use HTML form-based login

To use HTML form-based login, follow these steps:

1. Create a Policy Name. Enter a policy name (`worklight-form`), and then click **Apply Policy**.
2. Click **New Rule** (see Figure 7-14 on page 157). Repeat this for the following rules, in order:
   a. Rule 1. Verify if an LTPA Token exists in the HTTP request (`worklight-form_rule_0`):
      i. Direction: "Client to Server"
      ii. Match: type = http, HTTP Header Tag = Cookie, HTTP Value Match=*LtpaToken*
      iii. AAA: VerifyLTPA
      iv. Input: INPUT
      v. Output:OUTPUT
   b. Rule 2. Handle the authentication of the user if an LTPA token does not exist (`worklight-form_rule_2`):
      i. Direction: "Client to Server"
      ii. Match: type = url, match pattern =*/j_security_check
      iii. Advanced: "Convert Query Parameter to XML" (choose the defaults for other selections)
      iv. AAA: Form2LTPA
      v. Input: INPUT
      vi. Output:OUTPUT

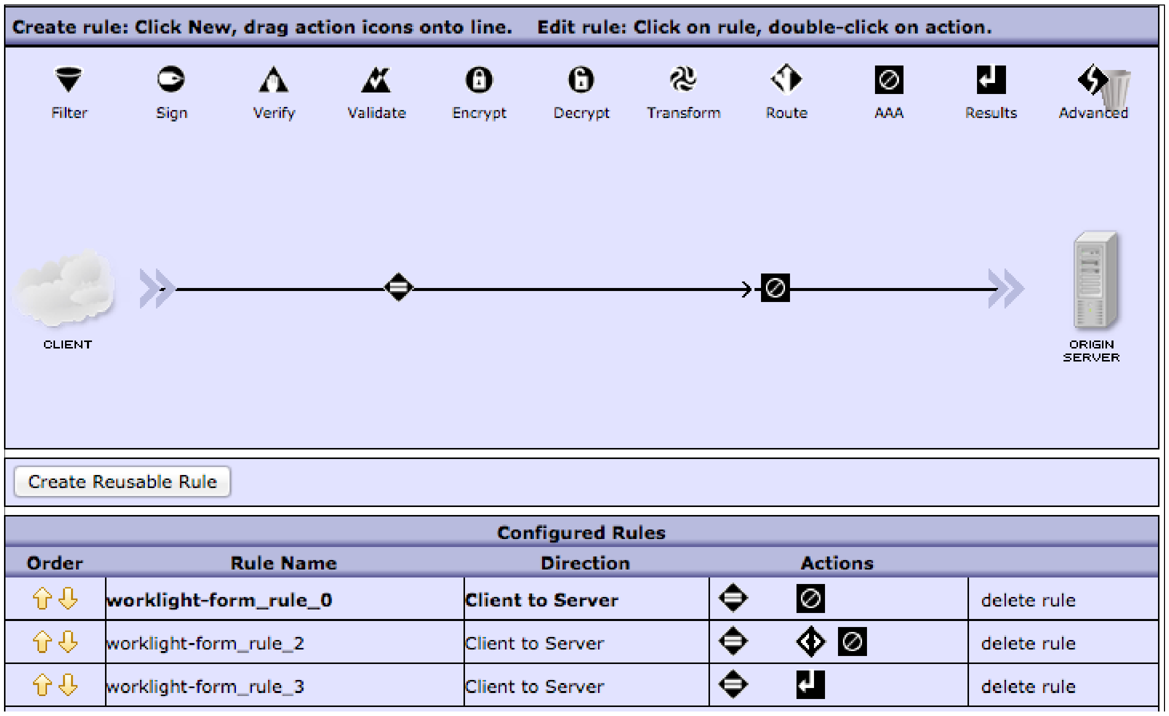
c. Rule 3. Match all rule (`worklight-form_rule_3`):

   i. Direction: "Client to Server"
   ii. Match: type = url, match pattern = *
   iii. Input: INPUT
   iv. Output: OUTPUT

Tip: This rule is shown here for this specific use case. However, you might want to divide rule 3 to further protect requests (for example, jpg or css) that do not require authentication (anonymous access) versus requests that require authentication. For example, you might want to replace Rule 3 with two new rules:

- The first rule is for accessing resources that do not need protection (unauthenticated or anonymous access).
- The second rule is for protecting resources that do require a valid credential. However, if this rule ever gets invoked, it would mean that the LTPA token was missing, and perhaps this rule should then reject the request.

Figure 7-14 shows the MPGW-style policy.



Order	Rule Name	Direction	Actions	
⇧⇩	**worklight-form_rule_0**	**Client to Server**		delete rule
⇧⇩	worklight-form_rule_2	Client to Server		delete rule
⇧⇩	worklight-form_rule_3	Client to Server		delete rule

Figure 7-14 MPGW-style policy for HTML form-based login support

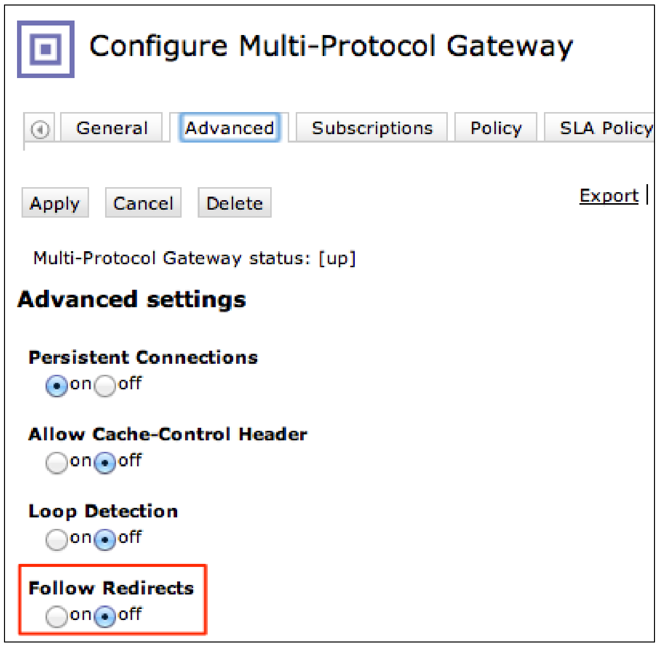
Completing the MPGW configuration

After you have completed the options on the General tab, there are additional options to select on the Advanced tab.

- Persistent Connections
- Allow Cache-Control Header
- Loop Detection
- Follow Redirects

  In this instance, select **off** for this option. This prevents the DataPower Integration Appliance back-end user agent from resolving redirects from the back-end. Web applications typically require a client browser to resolve redirects, so that they can maintain the context for `directory`, along with setting the LTPA cookie on the client.

Figure 7-15 shows these additional options on the Advanced tab.



Multi-Protocol Gateway status: [up]

Advanced settings

Persistent Connections
[✓ on] off

Allow Cache-Control Header
on [✓ off]

Loop Detection
on [✓ off]

Follow Redirects
on [✓ off]

Figure 7-15 Advanced tab

You are now ready to test the mobile app. Make sure that you create both a mobile environment and device-specific environment (IPhone/IPad or Android) to test the functionality.

7.3.6 Conclusion

In summary, Worklight mobile app traffic can be protected by using the DataPower Appliance's secure gateway functionality to enforce authentication on the DataPower Appliance device. Next, forward the credentials (header or LTPA token) downstream to Worklight Server to establish the user identity as part of the mobile traffic. This technique uses the strength of the DataPower Appliance functionality to protect the enterprise systems from unauthorized access.

A

Additional material

This book refers to additional material that can be downloaded from the Internet, as described in the following sections.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server:

`ftp://www.redbooks.ibm.com/redbooks/SG248179`

Alternatively, you can go to the IBM Redbooks website:

`ibm.com``/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG24-8179.

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material `.zip` file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *Extending your business to mobile devices with IBM Worklight*, SG24-8117
- *Enabling Mobile Apps with IBM Worklight Application Center*, REDP-5005

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, drafts, and additional materials, on the Redbooks website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Redbooks

Securing Your Mobile Business with IBM Worklight

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Securing Your Mobile Business with IBM Worklight

Apply Worklight security features to your mobile applications

Integrate Worklight with IBM Security Access Manager

Learn by example with practical scenarios

The IBM Worklight mobile application platform helps you to develop, deploy, host, and manage mobile enterprise applications. It also enables companies to integrate security into their overall mobile application lifecycle.

This IBM Redbooks publication describes the security capabilities offered by Worklight to address mobile application security objectives.

The book begins with an overview of IBM MobileFirst and its security offerings. The book also describes a business scenario illustrating where security is needed in mobile solutions, and how Worklight can help you achieve it.

This publication then provides specific, hands-on guidance about how to integrate Worklight with enterprise security. It also provides step-by-step guidance to implementing mobile security features, including *direct update*, *remote disable*, and *encrypted offline cache*. Integration between Worklight and other IBM security technologies is also covered, including integration with IBM Security Access Manager and IBM WebSphere DataPower.

This Redbooks publication is of interest to anyone looking to better understand mobile security, and to learn how to enhance mobile security with Worklight.

SG24-8179-00 ISBN 073843888X