



Red Hat JBoss Enterprise Application Platform 6.4

Security Architecture

Security Architecture Guide

Red Hat JBoss Enterprise Application Platform 6.4 Security Architecture

Security Architecture Guide

Legal Notice

Copyright © 2017 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This document focuses on the high level concepts of security within JBoss EAP 6 and what components exist to implement those concepts. This document focuses on what and why and much less on how, meaning specifics on how to configure a specific scenario will be housed in other documents. When completing this document, readers should have a solid conceptual understanding of the components of security within JBoss EAP 6, as well as how those components fit together.

Table of Contents

CHAPTER 1. OVERVIEW OF GENERAL SECURITY CONCEPTS	4
1.1. AUTHENTICATION	4
1.2. AUTHORIZATION	4
1.3. AUTHENTICATION AND AUTHORIZATION IN PRACTICE	4
1.4. ENCRYPTION	4
1.5. SSL/TLS AND CERTIFICATES	5
1.6. SINGLE SIGN ON (SSO)	5
1.6.1. Third-Party SSO Implementations	6
1.6.2. Claims-Based Identity	7
1.7. LDAP	8
CHAPTER 2. HOW RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6 HANDLES SECURITY OUT OF THE BOX	9
2.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM-SPECIFIC CONCEPTS	9
2.1.1. Core Services, Subsystems, and Profiles	9
2.1.2. Management Interfaces	9
2.1.3. Security Realms	10
2.1.4. Security Domains	10
2.1.5. Using Security Realms and Security Domains	11
2.1.6. Security Auditing	11
2.1.7. Security Mapping	11
2.1.8. JMX	11
2.1.9. Role-Based Access Control	12
2.1.10. Declarative Security and JAAS	14
2.2. CORE MANAGEMENT AUTHENTICATION	15
2.2.1. Default Security	15
2.2.1.1. Local and Remote Client Authentication with the Native Interface	15
2.2.1.2. Local and Remote Client Authentication with the HTTP Interface	16
2.2.2. Advanced Security	16
2.2.2.1. Updating the Management Interfaces	16
2.2.2.2. Adding Outbound Connections	17
2.2.2.3. Adding RBAC to the Management Interfaces	17
2.2.2.4. Using LDAP with the Management Interfaces	19
2.2.2.5. JAAS and the Management Interfaces	20
2.3. SECURITY SUBSYSTEM	20
2.3.1. Password Vault System	20
2.3.2. Security Domains	20
2.3.2.1. Login Modules	21
2.3.2.2. Password Stacking	23
2.3.2.3. Password Hashing	23
2.3.3. Security Management	24
2.3.3.1. Deep Copy Mode	24
2.3.4. Additional Components	24
2.3.4.1. JASPI	24
2.3.4.2. JACC	24
2.3.4.3. About Fine Grained Authorization and XACML	24
2.3.4.4. SSO	25
CHAPTER 3. ADDITIONAL USECASES FOR SSO WITH RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM	26
3.1. BROWSER-BASED SSO USING SAML	26
3.1.1. Identity Provider Initiated Flow	26

3.1.2. Global Logout	27
3.2. DESKTOP-BASED SSO	27
3.3. SSO USING STS	27
CHAPTER 4. EXAMPLE SCENARIOS	29
4.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM OUT OF THE BOX	29
4.1.1. Core Management Authentication Out of the Box	29
4.1.1.1. Security	29
4.1.1.2. How it works	29
4.1.2. Security Subsystem Out of the Box	30
4.1.2.1. Security	30
4.1.2.2. How it Works	30
4.2. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM WITH HTTPS AND RBAC ADDED TO THE MANAGEMENT INTERFACES	30
4.2.1. Security	30
4.2.2. How it works	30
4.3. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM WITH AN UPDATED SECURITY SUBSYSTEM INCLUDING HTTPS	31
4.3.1. Security	32
4.3.2. How it works	32
4.4. SSO FOR WEB APPLICATIONS ON RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM	32
4.4.1. Security	32
4.4.2. How it works	33
4.5. MULTIPLE RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM INSTANCES AND MULTIPLE APPLICATIONS USING BROWSER-BASED SSO WITH SAML	34
4.5.1. Security	34
4.5.2. How it works	34
4.6. USING LDAP WITH THE MANAGEMENTREALM	36
4.6.1. Security	36
4.6.2. How it works	36
4.7. USING DESKTOP SSO (VIA KERBEROS) TO PROVIDE SSO FOR WEB APPLICATIONS	37
4.7.1. Security	37
4.7.2. How it works	37

CHAPTER 1. OVERVIEW OF GENERAL SECURITY CONCEPTS

Before digging into how JBoss EAP 6 handles security, it is important to understand a few basic security concepts.

1.1. AUTHENTICATION

Authentication refers to identifying a subject and verifying the authenticity of the identification. The most common authentication mechanism is a username and password combination, but other mechanisms, such as shared keys, smart cards, or fingerprints are also used for Authentication. When in the context of Java Enterprise Edition declarative security, the result of a successful authentication is called a *principal*.

1.2. AUTHORIZATION

Authorization refers to a way of specifying access rights or defining access policies. A system can then implement a mechanism to utilize those policies to permit or deny access to resources for the requestor. In many cases, this is implemented by matching a principal with a set of actions or places they are or are not allowed to access, sometimes referred to as a role.

1.3. AUTHENTICATION AND AUTHORIZATION IN PRACTICE

Though Authentication and Authorization are distinct concepts, they are very often linked. Many modules written to handle authentication also handle authorization and vice-versa.

Example

The application *MyPersonalSoapbox* provides the ability to post and view messages. Principals with the *Talk* role are allowed to post messages and view other posted messages. Users who have not logged in have the *Listen* role and are allowed to view posted messages. *Suzy*, *Adam*, and *Bob* use the application. *Suzy* and *Bob* can authenticate with their username and password, but *Adam* does not have a username and password yet. *Suzy* has the *Talk* role, but *Bob* has no roles (neither *Talk* nor *Listen*). When *Suzy* authenticates, she may post and view messages. When *Adam* uses *MyPersonalSoapbox*, he cannot log in, but he can still see posted messages. When *Bob* logs in, he cannot post any messages nor can he view any other posted messages.

Suzy is both authenticated and authorized. *Adam* has not authenticated, but he is authorized (with the *Listen* role) to view messages. *Bob* is authenticated, but has no authorization (no roles).

1.4. ENCRYPTION

Encryption refers to encoding sensitive information by applying mathematical algorithms to it. Data is secured by converting (or encrypting) it to an encoded format. In order to read the data again, the encoded format must be converted back (or decrypted) to the original format. Encryption can be applied to simple string data in files or databases, or even on data sent across communications streams.

Examples of encryption include:

- LUKS can be used to encrypt Linux filesystem disks.
- The blowfish or AES algorithms can be used to encrypt data stored in Postgres databases.
- The HTTPS protocol encrypts all data via SSL/TLS before transferring it from one party to another.

- When a user connects from one server to another using the Secure Shell (SSH) protocol, all of the communication is sent in an encrypted tunnel.

1.5. SSL/TLS AND CERTIFICATES

Secure Sockets Layer (SSL)/Transport Layer Security (TLS) encrypts network traffic between two systems. This occurs by using a symmetric key which is exchanged between and only known by those two systems. To ensure a secure exchange of the symmetric key, SSL/TLS makes use of Public Key Infrastructure (PKI), a method of encryption that utilizes a key pair. A key pair consists of two separate but matching cryptographic keys - a public key and a private key. The public key is shared with any party and is used to encrypt data, and the private key is kept secret and is used to decrypt data that has been encrypted using the public key.

When a client requests a secure connection to exchange symmetric keys, a handshake phase takes place before secure communication can begin. During the SSL/TLS handshake, the server passes its public key to the client in the form of a certificate. The certificate contains the identity of the server (its URL), the public key of the server, and a digital signature that validates the certificate. The client then validates the certificate and makes a decision about whether the certificate is trusted or not. If the certificate is trusted, the client generates the symmetric key for the SSL/TLS connection, encrypts it using the public key of the server, and sends it back to the server. The server decrypts the symmetric key, using its private key, and further communication between the two machines over this connection is encrypted using the symmetric key.

There are two basic kinds of certificates: **Self-Signed Certificates** and **Authority-Signed Certificates**. A self-signed certificate uses its own private key to sign itself, and that signature is unverified (not connected to any chain of trust). An authority-signed certificate is a certificate that is issued to a party by a certificate authority and is signed by that certificate authority (e.g. Verisign, CAcert, RSA and many others). The certificate authority is essentially verifying the authenticity of the holder of the certificate.

Self-Signed certificates can be faster and easier to generate and require less infrastructure to manage, but they can be difficult for clients to verify their authenticity since no third party has confirmed their authenticity. This inherently makes them less secure. Authority-signed certificates can take more effort to setup initially, but are far easier for clients to verify their authenticity (i.e. a chain of trust has been created since a third party has confirmed the authenticity of the holder of the certificate).

1.6. SINGLE SIGN ON (SSO)

Single Sign On (SSO) allows principals authenticated to one resource to implicitly authorize access to other resources. If a set of distinct resources are secured by SSO, a user is only required to authenticate the first time they access any of the secured resources. Upon successful authentication, the roles associated with the user are stored and used for authorization of all other associated resources. This allows the user to access any additional authorized resources without re-authenticating.

If the user logs out of a resource, or a resource invalidates the session programmatically, all persisted authorization data is removed, and the process starts over. In the case of a resource session timeout, the SSO session is not invalidated if there are other valid resource sessions associated with that user. SSO may be used for authentication and authorization on both web applications as well as on desktop applications. In some cases, a single SSO implementation can integrate with both.

Within SSO, there are a few common terms used to describe different concepts and parts of the system:

Identity Management

Identity Management (IDM) refers to the idea of managing principals and their associated authentication, authorization, and privileges across one or more systems or domains. The term *Identity and Access Management (IAM)* is sometimes used to describe this same concept.

Identity Provider

An *identity provider (IDP)* is the authoritative entity responsible for authenticating an end user and asserting the identity for that user in a trusted fashion to trusted partners.

Identity Store

An identity provider needs an *identity store* to retrieve users' information. This information will be used during the authentication and authorization process. Identity stores can be any type of repository: a database, LDAP, properties file, etc.

Service Provider

A *service provider* relies on an identity provider to assert information about a user via an electronic user credential, leaving the service provider to manage access control and dissemination based on a trusted set of user credential assertions.

Clustered and Non-Clustered SSO

Non-clustered SSO limits the sharing of authorization information to applications on the same virtual host. In addition, there is no resiliency in the event of a host failure. In a *clustered SSO* scenario, data can be shared between applications in multiple virtual hosts, and is therefore resilient to failover. In addition, clustered SSO is able to receive requests from a load balancer.

1.6.1. Third-Party SSO Implementations

Kerberos

Kerberos is a network authentication protocol for client/server applications. It allows authentication across a non-secure network in a secure way, using secret-key symmetric cryptography.

Kerberos uses security tokens called tickets. To use a secured service, users need to obtain a ticket from the Ticket Granting Service (TGS), which is a service running on a server on their network. After obtaining the ticket, users request a Service Ticket (ST) from an Authentication Service (AS), which is another service running on the same network. Users then use the ST to authenticate to the desired service. The TGS and the AS both run inside an enclosing service called the Key Distribution Center (KDC).

Kerberos is designed to be used in a client-server desktop environment, and is not usually used in web applications or thin client environments. However, many organizations already use a Kerberos system for desktop authentication, and prefer to reuse their existing system rather than create a second one for their web applications. Kerberos is an integral part of Microsoft Active Directory, and is also used in many Red Hat Enterprise Linux environments.

SPNEGO

Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) provides a mechanism for extending a Kerberos-based SSO environment for use in web applications.

When an application on a client computer, such as a web browser, attempts to access a protected page on a web server, the server responds that authorization is required. The application then requests a service ticket from the Kerberos Key Distribution Center (KDC). After the ticket is obtained, the application wraps it in a request formatted for SPNEGO, and sends it back to the web application, via the browser. The web container running the deployed web application unpacks the request and authenticates the ticket. Upon successful authentication, access is granted.

SPNEGO works with all types of Kerberos providers, including the Kerberos service included in Red Hat Enterprise Linux and the Kerberos server which is an integral part of Microsoft Active Directory.

Microsoft Active Directory

Microsoft Active Directory (AD) is a directory service developed by Microsoft to authenticate users and computers in a Microsoft Windows domain. It is included as part of Microsoft Windows Server. The computer running Microsoft Windows Server controlling the domain is referred to as the domain controller. Red Hat Enterprise Linux can integrate with Active Directory domains as can Red Hat Identity Management, Red Hat JBoss Enterprise Application Platform, and other Red Hat Products.

Active Directory relies on three core technologies which work together:

1. Lightweight Directory Access Protocol (LDAP), for storing information about users, computers, passwords, and other resources.
2. Kerberos, for providing secure authentication over the network.
3. Domain Name Service (DNS) for providing mappings between IP addresses and host names of computers and other devices on the network.

1.6.2. Claims-Based Identity

One way of implementing SSO is by using a *claims-based identity* system. A claims-based identity system allows systems to pass around identity information, but abstracts that information into two components: a *claim* and an *issuer* (or authority). A claim is statement that one subject (e.g. user, group, application, organization) makes about another. That claim (or set of claims) is then packaged into a token (or set of tokens) and issued by a provider. Claims-based identity allows individual secured resources to implement SSO without having to know everything about a user.

Security Token Service (STS)

A *Security Token Service (STS)* is an authentication service that issues security tokens to clients for use in authenticating and authorizing users for secured applications (web services or EJBs). A client attempting to authenticate against an application (service provider) secured with STS will be redirect to a centralized STS authenticate and issued a token. If successful, that client will reattempt to access the service provider, providing their token in along with the original request. That service provider will validate the token from the client with the STS and proceed accordingly. This same token may be reused by the client against other web services or EJBs that are connected to the STS. The concept of a centralized STS that can issue, cancel, renew and validate security tokens, and specifies the format of security token request and response messages is known as *WS-Trust*.

Browser-Based SSO

In *browser-based SSO*, one or more web applications, known as service providers, are connected to a centralized identity provider in a hub and spoke architecture. The identity provider (IDP) acts as the central source (hub) for identity and role information by issuing claim statements (via SAML) to service providers (spokes). Requests may be issued when a user attempts to access a service provider or if a user attempt to authenticate directly with the identity provider. These are known as SP-initiated and IDP-initiated flows respectively, and will both issue the same claim statements.

SAML

Security Assertion Markup Language (SAML) is a data format that allows two parties, usually an identity provider and a service provider, to exchange authentication and authorization information. A SAML token is a type of token issued by a STS or IDP and can be used to enable SSO. A resource secured by SAML (SAML service provider) will redirect users to the SAML identity provider (a type of STS or IDP) to obtain a valid SAML token before authenticating and authorizing that user.

Desktop-Based SSO

Desktop-Based SSO enables service providers and desktop domains (e.g. Active Directory or Kerberos) to share a principal. In practice, this allows users to login on their computer using their domain credentials and then have service providers re-use that principal during authentication (without having to re-authenticate), thus providing SSO.

1.7. LDAP

Lightweight Directory Access Protocol (LDAP) is a protocol for storing and distributing directory information across a network. This directory information includes information about users, hardware devices, access roles and restrictions, and other information.

In Lightweight Directory Access Protocol (LDAP), the Distinguished Name (DN) uniquely identifies an object in a directory. Each distinguished name must have a unique name and location from all other objects, which is achieved using a number of attribute-value pairs (AVPs). The AVPs define information such as common names, organization unit, among others. The combination of these values results in a unique string required by the LDAP.

Some common implementations of LDAP include Red Hat Directory Server, OpenLDAP, Microsoft Active Directory, IBM Tivoli Directory Server, Oracle Internet Directory, 389 Directory Server, and others.

CHAPTER 2. HOW RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6 HANDLES SECURITY OUT OF THE BOX

There are two components that ship with JBoss EAP 6 and above that relate to security: Core Management Authentication and the Security Subsystem. These two components are based on the general security concepts discussed in the [overview section](#), but they also incorporate some JBoss EAP-specific concepts in their implementation discussed in the in the [Section 2.1, “Red Hat JBoss Enterprise Application Platform-Specific Concepts”](#) section.

2.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM-SPECIFIC CONCEPTS

In addition to the general security concept covered in the [Chapter 1, *Overview of General Security Concepts*](#) section, it’s important to understand some of the concepts specific to JBoss EAP and JBoss EAP security.

2.1.1. Core Services, Subsystems, and Profiles

JBoss EAP 6 is built on the concept of modular classloading. Each API or service provided by JBoss EAP is implemented as a module, which is loaded and unloaded on demand. The core services are services which are always loaded on server startup and are required to be running prior to starting any additional subsystems.

A subsystem is an additional set of capabilities added to the core server by an extension. Some example subsystems include: a subsystem that provides servlet handling capabilities, a subsystem provides an EJB container, and a subsystem provides JTA.

A profile is a named list of subsystems, along with the details of each subsystem’s configuration. A profile with a large number of subsystems results in a server with a large set of capabilities. A profile with a small, focused set of subsystems will have fewer capabilities but a smaller footprint. By default, JBoss EAP 6 comes with several pre-defined profiles (e.g. default, full, ha, full-ha). In these profiles, the Management Interfaces and the associated security realms are loaded as core services.

2.1.2. Management Interfaces

JBoss EAP 6 offers two different management interfaces (APIs) for interacting with and editing its configuration: The Management Console (HTTP API) and the Management CLI (Native API). Both of these interfaces expose the functionality of the core management of JBoss EAP. In essence, these interfaces offer two ways to access the same core management system.

The Management Console is a web-based administration tool for JBoss EAP 6 . It may be used to start and stop servers, deploy and undeploy applications, tune system settings, and make persistent modifications to the server configuration. The Management Console also has the ability to perform administrative tasks, with live notifications when any changes require the server instance to be restarted or reloaded. In a Managed Domain, server instances and server groups in the same domain can be centrally managed from the Management Console of the domain controller.

The Management Command Line Interface (CLI) is a command line administration tool for JBoss EAP 6. The Management CLI may be used to start and stop servers, deploy and undeploy applications, configure system settings, and perform other administrative tasks. Operations can be performed in batch mode, allowing multiple tasks to be run as a group. The Management CLI may also connect to the

Domain Controller in a managed domain to execute management operations on the domain. The Management CLI can perform all tasks that the web-based administration tool can perform as well as many other lower level operations that are unavailable to the web-based administration tool.



NOTE

In addition to the clients that ship with JBoss EAP 6, other clients can be written to invoke the management interfaces over either the HTTP or native interfaces using the APIs included with JBoss EAP 6.

2.1.3. Security Realms

A security realm is effectively an identity store of usernames, passwords, and group membership information that can be used when authenticating users in EJBs, Web applications, and the Management Interface. Initially, JBoss EAP 6 comes pre-configured with two security realms by default: *ManagementRealm* and *ApplicationRealm*. Both of these security realms use the filesystem to store mappings between users and passwords, and users and group membership information and use a digest mechanism by default when authenticating.

A digest mechanism is simply an authentication mechanism that authenticates the user by making use of one-time, one-way hashes comprised of various pieces of information including information stored in the users/passwords mapping property file. This allows JBoss EAP to authenticate users without sending any passwords in plain text over the network.

A script is included with the JBoss EAP 6 install which enables administrators to add users to both realms. When users are added in this way, the username and hashed password are stored in the corresponding users/passwords properties file. When a user attempts to authenticate, JBoss EAP sends back a one-time use number (nonce) to the client. The client then generates a one-way hash using their username, password, nonce and a few other fields, and then sends back to JBoss EAP the username, nonce and one-way hash. JBoss EAP then looks up the user's pre-hashed password and uses it along with the provided username and nonce and few other fields to generate another one-way hash in the same manner. If all the same information is used (e.g. correct password) on both sides then hashes will match and the user is authenticated.

Though security realms use the digest mechanism by default, they may be reconfigured to use other authentication mechanisms as well. On startup, the management interfaces determine which authentication mechanisms will be enabled based on what authentication mechanisms are configured in *ManagementRelam*.

Security realms are not involved in any authorization decisions, however they can be configured to load a user's group membership information which can subsequently be used to make authorization decisions. After a user has been authenticated, a second step will occur to load the group membership information based on the username.

By default, the *ManagementRealm* is used during authentication and authorization for the management interfaces. The *ApplicationRealm* is a default realm made available for web applications and EJBs to use when authenticating and authorizing users.

2.1.4. Security Domains

A security domain is a set of Java Authentication and Authorization Service (JAAS) declarative security configurations which one or more applications use to control authentication, authorization, auditing, and mapping. Three security domains are included by default: *jboss-ejb-policy*, *jboss-web-policy*, and *other*.

jboss-ejb-policy and *jboss-web-policy* are the default authorization mechanisms that are used if an application's configured security domain has none. Those security domains along with *other* are also used internally within JBoss EAP for authorization and are therefore required for correct operation.

A security domain consists of configurations for authentication, authorization, security mapping, and auditing. Security domains are part of the JBoss EAP 6 security subsystem and are managed centrally by the domain controller or standalone server. Users can create as many security domains as needed to accommodate application requirements.

2.1.5. Using Security Realms and Security Domains

Both security realms and security domains can be used as a part of securing web applications deployed to JBoss EAP. When deciding if either should be used, it's important to understand the difference between the two.

Web applications and EJB deployments can only use Security Domains directly. They perform the actual authentication and authorization via login modules using the identity information passed from an identity store. Security domains can be configured to use Security Realms for identity information (e.g. *other* allows applications to specify a security realm to use for authentication and getting authorization information), but they may also be configured to use external identity stores. Web applications and EJB deployments cannot be configured to directly use Security Realms for authentication. The security domains are also part of the security subsystem and are therefore loaded after core services.

Only the core management (e.g. the management interfaces) and the EJB remoting end points can use the Security Realms directly. They are identity stores that provide authentication as well as authorization information. They are also a core service and are loaded before any subsystems are started. The out of the box Security Realms (*ManagementRealm* and *ApplicationRealm*) use a simple file-based authentication mechanism, but they can be configured to use other mechanisms.

2.1.6. Security Auditing

Security auditing refers to triggering events, such as writing to a log, in response to an event that happens within the security subsystem. Auditing mechanisms are configured as part of a security domain, along with authentication, authorization, and security mapping details. Auditing uses provider modules to control the way that security events are reported. JBoss EAP ships with several security auditing providers, but custom ones may be used as well. In addition, the core management of JBoss EAP also has its own security auditing and logging functionality which is configure separately and is not part of the security subsystem.

2.1.7. Security Mapping

Security mapping adds the ability to combine authentication and authorization information after the authentication or authorization happens, but before the information is passed to your application. Roles (authorization), principals (authentication), or credentials (attributes which are not principals or roles) may all be mapped. Role Mapping is used to add, replace, or remove roles to the subject after authentication. Principal mapping is used to modify a principal after authentication. Credential (attribute) mapping is used to convert attributes from an external system to be used by your application, and vice versa.

2.1.8. JMX

Java Management Extensions (JMX) provides a way to remotely trigger JDK and application management operations. The Management API of JBoss EAP 6 is exposed as JMX Managed Beans. These Managed Beans are referred to as *core mbeans* and access to them is controlled and filtered exactly the same as the underlying Management API itself.



NOTE

Prior to JBoss EAP 6, the management functionality was primarily JMX based, meaning the management functionality relied on these JMX exposed beans to perform operations. With JBoss EAP 6, the core management does not rely on them to perform operations. JMX exposed beans are now just alternative mechanism (in addition to the native and http interfaces) to access and perform management operations.

2.1.9. Role-Based Access Control

Role-Based Access Control (RBAC) is a mechanism for specifying a set of permissions for management users. It allows multiple users to share responsibility for managing JBoss EAP 6 servers without each of them requiring unrestricted access. By providing a *separation of duties* for management users, JBoss EAP 6 makes it easy for an organization to spread responsibility between individuals or groups without granting unnecessary privileges. This ensures the maximum possible security of your servers and data while still providing flexibility for configuration, deployment, and management.

Role-Based Access Control in JBoss EAP 6 works through a combination of role permissions and constraints. Seven predefined roles are provided that each have different fixed permissions. Each management user is assigned one or more roles, which specify what the user is permitted to do when managing the server.

Role-Based Access Control is supported by JBoss EAP 6.2 and above, but is disabled by default.

Standard Roles

JBoss EAP 6 provides seven predefined user roles: *Monitor*, *Operator*, *Maintainer*, *Deployer*, *Auditor*, *Administrator*, and *SuperUser*. Each of these roles has a different set of permissions and is designed for specific use cases. The *Monitor*, *Operator*, *Maintainer*, *Administrator*, and *SuperUser* role each build successively upon each other, with each having more permissions than the previous. The *Auditor* and *Deployer* roles are similar to the *Monitor* and *Maintainer* roles respectively but have some additional special permissions and restrictions.

Monitor

Users of the Monitor role have the fewest permissions and can only read the current configuration and state of the server. This role is intended for users who need to track and report on the performance of the server. Monitors cannot modify server configuration nor can they access sensitive data or operations.

Operator

The Operator role extends the Monitor role by adding the ability to modify the runtime state of the server. This means that Operators can reload and shutdown the server as well as pause and resume JMS destinations. The Operator role is ideal for users who are responsible for the physical or virtual hosts of the application server so they can ensure that servers can be shutdown and restarted corrected when needed. Operators cannot modify server configuration or access sensitive data or operations.

Maintainer

The Maintainer role has access to view and modify runtime state and all configuration except sensitive data and operations. The Maintainer role is the general purpose role that doesn't have access to sensitive data and operation. The Maintainer role allows users to be granted almost complete access to administer the server without giving those users access to passwords and other sensitive information. Maintainers cannot access sensitive data or operations.

Administrator

The Administrator role has unrestricted access to all resources and operations on the server except the audit logging system. The Administrator role has access to sensitive data and operations. This

role can also configure the access control system. The Administrator role is only required when handling sensitive data or configuring users and roles. Administrators cannot access the audit logging system and cannot change themselves to the Auditor or SuperUser role.

SuperUser

The SuperUser role has no restrictions and has complete access to all resources and operations of the server including the audit logging system. This role is equivalent to the administrator users of earlier versions of JBoss EAP 6 (6.0 and 6.1). If RBAC is disabled, all management users have permissions equivalent to the SuperUser role.

Deployer

The Deployer role has the same permissions as the Monitor, but can modify configuration and state for deployments and any other resource type enabled as an application resource.

Auditor

The Auditor role has all the permissions of the Monitor role and can also view (but not modify) sensitive data, and has full access to the audit logging system. The Auditor role is the only role other than SuperUser that can access the audit logging system. Auditors cannot modify sensitive data or resources. Only read access is permitted.

Permissions

What each role is allowed to do is defined by what permissions it has. Not every role has every permission. Notably SuperUser has every permission and Monitor has the least. Each permission can grant read and/or write access for a single category of resources. The categories are: runtime state, server configuration, sensitive data, the audit log, and the access control system.

Table 2.1. Permissions of Each Role

	Monitor	Operator	Maintainer	Deployer	Auditor	Administrator	SuperUser
Read Config and State	X	X	X	X	X	X	X
Read Sensitive Data ²					X	X	X
Modify Sensitive Data ²						X	X
Read/Modify Audit Log					X		X
Modify Runtime State		X	X	X ¹		X	X

	Monitor	Operator	Maintainer	Deployer	Auditor	Administrator	SuperUser
Modify Persistent Config			X	X ¹		X	X
Read/Modify Access Control						X	X

¹ permissions are restricted to application resources.

² What resources are considered to be "sensitive data" are configured using Sensitivity

Constraints

Constraints are named sets of access-control configuration for a specified list of resources. The RBAC system uses the combination of constraints and role permissions to determine if any specific user can perform a management action.

Constraints are divided into three classifications:

Application Constraints

Application Constraints define sets of resources and attributes that can be accessed by users of the Deployer role. By default the only enabled Application Constraint is core which includes deployments, deployment overlays. Application Constraints are also included (but not enabled by default) for datasources, logging, mail, messaging, naming, resource-adapters and security. These constraints allow Deployer users to not only deploy applications but also configure and maintain the resources that are required by those applications.

Sensitivity Constraints

Sensitivity Constraints define sets of resources that are considered *sensitive*. A sensitive resource is generally one that is either secret, like a password, or one that will have serious impact on the operation of the server, like networking, JVM configuration, or system properties. The access control system itself is also considered sensitive. The only roles permitted to write to sensitive resources are Administrator and SuperUser. The Auditor role is only able to read sensitive resources. No other roles have access.

Vault Expression Constraint

The Vault Expression constraint defines if reading or writing vault expressions is consider a sensitive operation. By default both reading and writing vault expressions is a sensitive operation.

2.1.10. Declarative Security and JAAS

Declarative security is a method to separate security concerns from application code by using the container to manage security. The container provides an authorization system based on either file permissions or users, groups, and roles. This approach is usually superior to programmatic security, which gives the application itself all of the responsibility for security. JBoss EAP 6 provides declarative security via security domains in the Security Subsystem.

Java Authentication and Authorization Service (JAAS) is a declarative security API which consists of a set of Java packages designed for user authentication and authorization. The API is a Java

implementation of the standard Pluggable Authentication Modules (PAM) framework. It extends the Java EE access control architecture to support user-based authorization. The JBoss EAP 6 security subsystem is actually based on the JAAS API.

Since JAAS is the foundation for the security subsystem, authentication is performed in a pluggable fashion. This permits Java applications to remain independent from underlying authentication technologies, such as Kerberos or LDAP, and allows the security manager to work in different security infrastructures. Integration with a security infrastructure is achievable without changing the security manager implementation. Only the configuration of the authentication stack JAAS uses needs to be changed.

2.2. CORE MANAGEMENT AUTHENTICATION

Core Management Authentication is responsible for securing the Management Interfaces (HTTP and Native) for the core management functionality using the *ManagementRealm*. It is built into the core management and is enabled and configured as a core service by default. It is only responsible for securing the Management Interfaces.

2.2.1. Default Security

By default, the Core Management Authentication secures each of the Management Interfaces (HTTP and Native) in two different forms: local clients and remote clients, both of which are configured using the *ManagementRealm* security realm by default. These defaults may be configured differently or replaced entirely. In addition to securing the Management Interfaces, the HTTP and Native interfaces may each be disabled.



NOTE

Out of the box, the Management Interfaces are configured to use simple access controls, which does not use roles. As a result, all users by default (when using simple access controls) have the same privileges as the SuperUser Role, which essentially has access to everything.

2.2.1.1. Local and Remote Client Authentication with the Native Interface

The Native Interface (CLI) can be invoked on the same host as the running JBoss EAP instance (local) or from another machine with the `jboss-cli` script (remote). When attempting to connect via the Native Interface, JBoss EAP presents the client with a list of available SASL authentication mechanisms (e.g. local jboss user, BASIC, etc). The client chooses its desired authentication mechanism and attempts to authenticate with the JBoss EAP instance. If it fails, it retries with any remaining mechanisms or stops attempting to connect. Local clients have the option to use the *local jboss user* authentication mechanism. This security mechanism is based on the client's ability to access the local filesystem. It simply validates that the user attempting to log in actually has access to the local filesystem on the same host as the JBoss EAP instance.

This authentication mechanism happens in four steps:

1. The client sends a message to the server which includes a request to authenticate using *local jboss user*.
2. The server generates a one-time token, writes it to a unique file, and sends a message to the client with the full path of the file.
3. The client reads the token from the file and sends it to the server, verifying that it has local access to the filesystem.

4. The server verifies the token and then deletes the file.

This form of authentication is based on the principle that if physical access to the filesystem is achieved, other security mechanisms are superfluous. The reasoning being that if a user has local filesystem access, that user has enough access to create a new user or otherwise thwart other security mechanism put in place. This is sometimes referred to as *Silent Authentication* since it allows the local user the ability to access the Management CLI without username or password authentication.

This functionality is enabled as a convenience, and to assist local users running Management CLI scripts without requiring additional authentication. It is considered a useful feature given that access to the local configuration typically also gives the user the ability to add their own user details or otherwise disable security checks.

The Native Interface can also be accessed from other servers (or even the same server) as a remote client. When accessing the Native Interface as a remote client, clients will not be able to authenticate using *local jboss user* and will be forced another authentication mechanism (e.g. DIGEST). If a local client fails to authenticate via *local jboss user* it will automatically fall back and attempt to use the other mechanisms as a remote client.



NOTE

The Management CLI may also be invoked from other servers (or even the same server) using the HTTP interface as apposed to the native interface. All HTTP connections (CLI or otherwise) are considered to be remote and **NOT** covered by local interface authentication.

2.2.1.2. Local and Remote Client Authentication with the HTTP Interface

The HTTP Interface can be invoked by clients on the same host as the same host as the running JBoss EAP instance (local) or by clients from another machine (remote). Despite allowing both local and remote clients to access the HTTP interface, all clients accessing the HTTP interface, are treated as remote connections.

When a client attempts to connect to the HTTP management interfaces, JBoss EAP sends back an HTTP response with a status code of 401 (requires authentication) and a set of headers that list the supported authentication mechanisms (e.g. Digest, GSSAPI, etc). The header for Digest also includes the nonce generated by JBoss EAP. The client then looks at the headers and chooses which authentication method to use and sends an appropriate response. In the case where the client chooses Digest, it prompts the user for their username and password. The client then uses the supplied fields (e.g. username and password), the nonce, and a few other pieces of information to generate a one-way hash. The client then sends the one-way hash, username, and nonce back to JBoss EAP as a response. JBoss EAP then takes that information, generates another one-way hash, compares to the two, and authenticates the user based on the result.

2.2.2. Advanced Security

There are a number of ways to change the default configuration of Management Interfaces as well as the Authentication/Authorization mechanisms to affect how it is secured.

2.2.2.1. Updating the Management Interfaces

In addition to modifying the Authentication and Authorization mechanisms, JBoss EAP 6 allows administrators to update the configuration of the Management Interface itself. There are a number of options:

Configuring the Management Interfaces to use HTTPS

Configuring the JBoss EAP management console for communication only via HTTPS provides increased security. All network traffic between the client and management console is encrypted, which reduces the risk of security attacks such as a man-in-the-middle attack. Anyone administering a JBoss EAP instance has greater permissions on that instance than non-privileged users, and using HTTPS helps protect the integrity and availability of that instance. When configuring HTTPS with JBoss EAP 6, authority-signed certificates are preferred over self-signed certificates since they provide a chain of trust. Self-signed certificates are still permitted but not recommended.

Using 2-way SSL/TLS

2-way SSL/TLS authentication, also known as client authentication, authenticates both the client and the server using SSL/TLS certificates. This provides assurance that not only is the server who it says it is, but the client is also who it says it is.

Using Distinct Network Interfaces for HTTP and HTTPS Traffic

The Management Interface can listen on distinct network interfaces for HTTP and HTTPS connections. For instance, an administrator may want to configure an external interface to listen for HTTPS traffic only while the internal-facing interface can accept HTTP traffic. If a server listens for HTTP and HTTPS traffic on the same interface, HTTPS requests received by the HTTP listener are automatically redirected to the HTTPS port. When distinct interfaces are used for HTTP and HTTPS traffic, no redirection is performed when an HTTPS request is received by the HTTP listener.

Disabling Management Interfaces

In certain instances, such as with managed domains or when using other management clients such as JBoss Operations Network, administrators may wish disable the HTTP interface, the Native interface, or the Web Console. Each of these interfaces may be disabled or removed entirely.



NOTE

The native interface is used by JBoss EAP 6 when running in domain mode for several purposes, including communication with slave domain controllers. As a result, the native interface should not be disabled when running in domain mode.

Updating or Creating a New Security Realm

The default security realm can be updated or replaced with a new security realm.

2.2.2.2. Adding Outbound Connections

Some security realms connect to external interfaces, such as an LDAP server. An outbound connection defines how to make this connection. A pre-defined connection type, *ldap-connection*, sets all of the required and optional attributes to connect to the LDAP server and verify the credential.

2.2.2.3. Adding RBAC to the Management Interfaces

By default the *Role-Based Access Control* (RBAC) system is disabled. It is enabled by changing the provider attribute from *simple* to *rbac*. This can be done using the `jboss-cli` script. When RBAC is disabled or enabled on a running server, the server configuration must be reloaded before it takes effect.

When RBAC is enabled for the management interfaces, the role assigned to a user determines the resources to which they have access and what operations they can conduct with a resource's attributes. Only users of the *Administration* or *SuperUser* role can view and make changes to the access control system.

**WARNING**

Enabling RBAC without having users and roles properly configured could result in administrators being unable to login to the management interfaces.

RBAC's Effect on the Management Console (Web Console)

In the management console some controls and views are disabled (greyed out) or not visible at all depending on the permissions of the role to which the user has been assigned.

If the user does not have read permissions to a resource attribute, that attribute will appear blank in the console. For example, most roles cannot read the username and password fields for datasources.

If the user has read permissions but does not have write permissions to a resource attribute, that attribute will be disabled (greyed-out) in the edit form for the resource. If the user does not have write permissions to the resource, then the edit button for the resource will not appear.

If a user does not have permissions to access a resource or attribute (it is *unaddressable* for that role), it will not appear in the console for that user. An example of that is the access control system itself which is only visible to a few roles by default.

The management console also provides an interface for the following common RBAC tasks:

- View and configure what roles are assigned to (or excluded from) each user
- View and configure what roles are assigned to (or excluded from) each group
- View group and user membership per role.
- Configure default membership per role.
- Create a scoped role

**NOTE**

Constraints **cannot** be configured in the Management Console at this time.

RBAC's Effect on the Management CLI or API (HTTP and Native)

Users of the `jboss-cli` script or management API will encounter slightly different behavior in the API when RBAC is enabled.

Resources and attributes that cannot be read are filtered from results. If the filtered items are addressable by the role, their names are listed as `filtered-attributes` in the response-headers section of the result. If a resource or attribute is not addressable by the role, it is not listed.

Attempting to access a resource that is not addressable will result in a resource not found error.

If a user attempts to write or read a resource that they can address but lack the appropriate write or read permissions, a *Permission Denied* error is returned.

The management CLI can perform all of same RBAC tasks as the management console as well as a few additional tasks:

- Enable and disable RBAC
- Change permission combination policy
- Configuring Application Resource and Resource Sensitivity Constraints

RBAC's Effect on JMX Managed Beans

Role-Based Access Control applies to JMX in three ways:

1. The Management API of JBoss EAP 6 is exposed as JMX Managed Beans. These Managed Beans are referred to as *core mbeans* and access to them is controlled and filtered exactly the same as the underlying Management API itself.
2. The JMX subsystem is configured with write permissions being *sensitive*. This means only users of the *Administrator* and *SuperUser* roles can make changes to that subsystem. Users of the *Auditor* role can also read this subsystem configuration.
3. By default Managed Beans registered by deployed applications and services (non-core mbeans) can be accessed by all management users, but only users of the *Maintainer*, *Operator*, *Administrator*, and *SuperUser* roles can write to them.

RBAC Authentication

Role-Based Access Control works with the standard authentication providers that are included with JBoss EAP 6 :

Username/Password

Users are authenticated using a username and password combination which is verified according to the settings of the *ManagementRealm*, which has the ability to use a local properties file or LDAP.

Client Certificate

Using the Trust Store.

Local User

`jboss-cli` script authenticates automatically as *Local User* if the server that is running on the same machine. By default *Local User* is a member of the *SuperUser* group.

Regardless of which provider is used, JBoss EAP is responsible for the assignment of roles to users. However when authenticating with the *ManagementRealm* or an LDAP server, those systems can supply user group information. This information can also be used by JBoss EAP to assign roles to users.

2.2.2.4. Using LDAP with the Management Interfaces

JBoss EAP 6 includes several authentication and authorization modules which allow an LDAP server to be used as the authentication and authorization authority for web and EJB applications.

To use an LDAP directory server as the authentication source for the Management Console, Management CLI, or Management API, the following must be done:

1. Create an outbound connection to the LDAP server.
2. Create an LDAP-enabled security realm or update an existing security realm to use LDAP.
3. Reference the new security realm in the Management Interface.

The LDAP authenticator operates by first establishing a connection to the remote directory server. It then performs a search using the username, which the user passed to the authentication system, to find the fully-qualified distinguished name (DN) of the LDAP record. A new connection to the LDAP server is established, using the DN of the user as the credential and password supplied by the user. If this authentication to the LDAP server is successful, the DN is verified to be valid.

Once an LDAP-enabled security realm is created, it can be referenced by the management interface. The management interface will then use the security realm for authentication. JBoss EAP 6 can also be configured to use an outbound connection to a LDAP server using 2-way SSL/TLS for authentication in the Management Interface and CLI.

2.2.2.5. JAAS and the Management Interfaces

JAAS can be used to secure the management interfaces. When using JAAS for the management interfaces, the security realm must be configured to use a security domain. This introduces a dependency between core services and the subsystems. In addition, while SSL/TLS is not required to use JAAS to secure the management interfaces, it is heavily recommend that administrators enable SSL/TLS to avoid accidentally transmitting sensitive information in an unsecured manner.



NOTE

When JBoss EAP 6 instances are configured to run in **ADMIN_ONLY** mode, using JAAS to secure the management interfaces is not supported. For more information on **ADMIN_ONLY** mode, please see section *Reference of Switches and Arguments to pass at Server Runtime* of the [Administration and Configuration Guide](#).

2.3. SECURITY SUBSYSTEM

The security subsystem provides security infrastructure for applications and is based on the JAAS API. The subsystem uses a security context associated with the current request to expose the capabilities of the authentication manager, authorization manager, audit manager, and mapping manager to the relevant container.

The authentication and authorization managers handle authentication and authorization. The mapping manager handles adding, modifying, or deleting information from a principal, role, or attribute before passing the information to your application. The auditing manager allows users to configure provider modules to control the way that security events are reported.

In most cases, administrators should only need to focus on setting up and configuring security domains in regards to updating the configuration of the security subsystem. Outside of security domains, the only security element that may need to be changed is whether to use *deep-copy-subject-mode*. In the rare case where security elements do require changes, those configuration changes (as well as *deep-copy-subject-mode*) may be found in the [security management](#) portion of the security subsystem.

2.3.1. Password Vault System

JBoss EAP 6 has a Password Vault to encrypt sensitive strings, store them in an encrypted keystore, and decrypt them for applications and verification systems. In plain-text configuration files, such as XML deployment descriptors, it is sometimes necessary to specify passwords and other sensitive information. The JBoss EAP Password Vault can be used to securely store sensitive strings for use in plain-text files.

2.3.2. Security Domains

Security domains are configured centrally either at the domain controller or on the standalone server. When security domains are used, an application may be configured to use a security domain lieu of individually configuring security. This allows users and administrators to leverage [declarative security](#).

Example

One common scenario that benefits from this type of configuration structure is the process of moving applications between testing and production environments. If an application has its security individually configured, it may need to be updated every time its promoted to a new environment (e.g. from a testing environment to a production environment). If that application instead used a security domain, the JBoss EAP instances in the individual environments would have their security domains properly configured for the current environment, allowing the application to rely on the container to provide the proper security configuration (via the security domain).

2.3.2.1. Login Modules

JBoss EAP 6 includes several bundled login modules suitable for most user management which are configured within a security domain. The security subsystem offers some core login modules that can read user information from a relational database, an LDAP server, or flat files. In addition to these core login modules, JBoss EAP 6 provides other login modules that provide user information and functionality for very customized needs.

Summary of the login modules commonly used

Ldap Login Module

Ldap login module is a LoginModule implementation that authenticates against an LDAP server. The security subsystem connects to the LDAP server using connection information (i.e. a bindDN that has permissions to search both the baseCtxDN and rolesCtxDN trees for the user and roles) provided using a JNDI initial context. When a user attempts to authenticate, the LDAP login module connects to the LDAP server, and passes the user's credentials to the LDAP server. Upon successful authentication, an InitialLDAPContext is created for that user within JBoss EAP, populated with the user's roles.

LdapExtended Login Module

The LdapExtended (org.jboss.security.auth.spi.LdapExtLoginModule) login module searches for the user to bind, as well as the associated roles, for authentication. The roles query recursively follows DN's to navigate a hierarchical role structure. The LoginModule options include whatever options are supported by the chosen LDAP JNDI provider supports.

UsersRoles Login Module

UsersRoles login module is a simple login module that supports multiple users and user roles loaded from Java properties files. The primary purpose of this login module is to easily test the security settings of multiple users and roles using properties files deployed with the application.

Database Login Module

The Database login module is a Java Database Connectivity-based (JDBC) login module that supports authentication and role mapping. This login module is used if user name, password and role information are stored in a relational database. This works by providing a reference to logical tables containing Principals and Roles in the expected format.

Certificate Login Module

Certificate login module authenticates users based on X509 certificates. A typical use case for this login module is CLIENT-CERT authentication in the web tier. This login module only performs authentication and must be combined with another login module capable of acquiring authorization roles to completely define access to a secured web or EJB components. Two subclasses of this login module, CertRolesLoginModule and DatabaseCertLoginModule extend the behavior to obtain the authorization roles from either a properties file or database.

Identity Login Module

Identity login module is a simple login module that associates a hard-coded user name to any subject authenticated against the module. It creates a `SimplePrincipal` instance using the name specified by the `principal` option. This login module is useful when a fixed identity is required to be provided to a service. This can also be used in development environments for testing the security associated with a given principal and associated roles.

RunAs Login Module

RunAs login module is a helper module that pushes a run as role onto the stack for the duration of the login phase of authentication, then pops the run as role from the stack in either the commit or abort phase. The purpose of this login module is to provide a role for other login modules that must access secured resources in order to perform their authentication (for example, a login module that accesses a secured EJB). RunAs login module must be configured ahead of the login modules that require a run as role established.

Client Login Module

Client login module (`org.jboss.security.ClientLoginModule`) is an implementation of `LoginModule` for use by JBoss clients when establishing caller identity and credentials. This creates a new `SecurityContext`, assigns it a principal and a credential and sets the `SecurityContext` to the `ThreadLocal` security context. Client login module is the only supported mechanism for a client to establish the current thread's caller. Both stand-alone client applications, and server environments (acting as JBoss EJB clients where the security environment has not been configured to use the JBoss EAP security subsystem transparently) must use Client login module.



WARNING

This login module does not perform any authentication. It merely copies the login information provided to it into the server EJB invocation layer for subsequent authentication on the server. Within JBoss EAP 6, this is only supported for the purpose of switching a user's identity for in-JVM calls. This is **NOT** supported for remote clients to establish an identity.

SPNEGO Login Module

The SPNEGO login module (`org.jboss.security.negotiation.spnego.SPNEGOLoginModule`) is an implementation of `LoginModule` that establishes caller identity and credentials with a KDC. The module implements SPNEGO (Simple and Protected GSSAPI Negotiation mechanism) and is a part of the JBoss Negotiation project. This authentication can be used in the chained configuration with the `AdvancedLdap` login module to allow cooperation with an LDAP server. Web applications must also enable the `NegotiationAuthenticator` within the application in order to use this login module.

RoleMapping Login Module

The RoleMapping login module supports mapping roles, that are the end result of the authentication process, to one or more declarative roles. For example, if the authentication process has determined that the user *John* has the roles *IdapAdmin* and *testAdmin*, and the declarative role defined in the `web.xml` or `ejb-jar.xml` file for access is *admin*, then this login module maps the *admin* roles to *John*. The RoleMapping login module must be defined as an optional module to a login module configuration as it alters mapping of the previously mapped roles.

Remoting Login Module

The Remoting login module checks if the request that is currently being authenticated was received over the Remoting connection. In cases where the request was received via the remoting interface, that request is associated with the identity created during the authentication process.

Realm Direct Login Module

The Realm Direct login module allows for the use of an existing security realm to be used in making authentication and authorization decisions. When configured, this module will look up identity information using the referenced realm for making authentication decisions and mapping user roles. For example, the pre-configured *other* security domain that ships with JBoss EAP 6 has a Realm Direct login module. If no realm is referenced in this module, the *ApplicationRealm* security realm is used by default.

Custom Modules

In cases where the login modules bundled with the JBoss EAP security framework do not meet the needs of the security environment, a custom login module implementation may be written. The AuthenticationManager requires a particular usage pattern of the Subject principals set. A full understanding of the JAAS Subject class's information storage features and the expected usage of these features are required to write a login module that works with the AuthenticationManager.

In addition, the *Unauthenticated Identity* login module option is also commonly used. There are certain cases where requests are not received in an authenticated format. *unauthenticatedIdentity* is a login module configuration option that assigns a specific identity (*guest*, for example) to requests that are made with no associated authentication information. This can be used to allow unprotected servlets to invoke methods on EJBs that do not require a specific role. Such a principal has no associated roles and so can only access either unsecured EJBs or EJB methods that are associated with the unchecked permission constraint.

2.3.2.2. Password Stacking

Multiple login modules can be chained together in a stack, with each login module providing both the credentials verification and role assignment during authentication. This works for many use cases, but sometimes credentials verification and role assignment are split across multiple user management stores.

Consider the case where users are managed in a central LDAP server but application-specific roles are stored in the application's relational database. The password-stacking module option captures this relationship.

To use password stacking, each login module should set the *password-stacking* attribute to *useFirstPass*, which is located in the *<module-option>* section. If a previous module configured for password stacking has authenticated the user, all the other stacking modules will consider the user authenticated and only attempt to provide a set of roles for the authorization step.

When password-stacking option is set to *useFirstPass*, this module first looks for a shared user name and password under the property names *javax.security.auth.login.name* and *javax.security.auth.login.password* respectively in the login module shared state map.

If found, these properties are used as the principal name and password. If not found, the principal name and password are set by this login module and stored under the property names *javax.security.auth.login.name* and *javax.security.auth.login.password* respectively.

2.3.2.3. Password Hashing

Most login modules must compare a client-supplied password to a password stored in a user management system. These modules generally work with plain text passwords, but can be configured to support hashed passwords to prevent plain text passwords from being stored on the server side. JBoss

EAP 6 supports the ability to configure the hashing algorithm, encoding, and character set as well as when the user password and store password are hashed.



IMPORTANT

Red Hat JBoss Enterprise Application Platform Common Criteria certified configuration does not support hash algorithms weaker than SHA-256.

2.3.3. Security Management

The security management portion of the security subsystem is used to override the high-level behaviors of the security subsystem. Each setting is optional. It is unusual to change any of these settings except for deep copy subject mode.

2.3.3.1. Deep Copy Mode

If deep copy subject mode is disabled (the default), copying a security data structure makes a reference to the original, rather than copying the entire data structure. This behavior is more efficient, but is prone to data corruption if multiple threads with the same identity clear the subject by means of a flush or logout operation.

If deep copy subject mode is enabled, a complete copy of the data structure along with and all its associated data is made (as long as they are marked cloneable). This is more thread-safe, but less efficient.

2.3.4. Additional Components

2.3.4.1. JASPI

Java Authentication SPI for Containers (JASPI or JASPIC) is a pluggable interface for Java applications and is defined in [JSR-196](#). In addition to JAAS authentication, JBoss EAP 6 also allows for JASPI authentication to be used. JASPI authentication is configured using login modules in a security domain and those modules may be stacked. The web-based management console does not expose the configuration of JASPI authentication modules. In addition, applications deployed to JBoss EAP 6 require a special authenticator to be configured in their deployment descriptor to use JASPI security domains.

2.3.4.2. JACC

Java Authorization Contract for Containers (JACC) is a standard which defines a contract between containers and authorization service providers, which results in the implementation of providers for use by containers. It was defined in [JSR-115](#) and has been part of the core Java EE specification since version 1.3.

JBoss EAP 6 implements support for JACC within the security functionality of the security subsystem.

2.3.4.3. About Fine Grained Authorization and XACML

Fine Grained Authorization caters to the changing requirements and multiple variables involved in the decision making process, which becomes the basis of providing authorization for accessing a module. Hence, the process of Fine Grained Authorization is complex in itself.

**NOTE**

The XACML bindings (web, ejb) are not supported in JBoss EAP 6.

JBoss EAP uses XACML as a medium to achieve Fine Grained Authorization. XACML provides standards based solution to the complex nature of achieving Fine Grained Authorization. XACML defines a policy language and an architecture for decision making. The XACML architecture includes a Policy Enforcement Point (PEP), which intercepts any requests in a normal program flow, then asks a Policy Decision Point (PDP) to make an access decision based on the policies associated with the PDP. The PDP evaluates the XACML request created by the PEP and runs through the policies to make one of the following access decisions:

PERMIT

The access is approved.

DENY

The access is denied.

INDETERMINATE

There is an error at the PDP.

NOTAPPLICABLE

There is some attribute missing in the request or there is no policy match.

XAMCL also has the following features:

- Oasis XACML v2.0 library
- JAXB v2.0 based object model
- ExistDB Integration for storing/retrieving XACML Policies and Attributes

2.3.4.4. SSO

JBoss EAP 6 provides out of the box support for both clustered and non-clustered SSO via the web and Infinispan subsystems. This requires the following:

- A configured security domain which handles authentication and authorization.
- The *SSO* infinispan replication cache. It is present in the *ha* and *full-ha* profiles for a managed domain, or by using the *standalone-ha* or *standalone-full-ha* profile for a standalone server.
- The *web* cache-container and *SSO* replication cache within it must each be present.
- The *web* subsystem needs to be configured to use SSO.
- Each application which will share the SSO information must be configured to use the same security domain.

CHAPTER 3. ADDITIONAL USECASES FOR SSO WITH RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

In addition to the out of the box functionality, JBoss EAP 6 supports additional use cases for Single Sign On including SAML for browser-based SSO, desktop-based SSO, and SSO via a secure token service.

3.1. BROWSER-BASED SSO USING SAML

In a browser-based SSO scenario, one or more web applications (service providers) are connected to a centralized identity provider in a hub and spoke architecture. The identity provider (IDP) acts as the central source (hub) for identity and role information to all the service providers or spokes. When an unauthenticated user attempts to access one of the service providers, that user is instead redirected to an identity provider to perform the authentication. The identity provider authenticates the user, issues a SAML token specifying the role of the principal, and redirects them back to the requested service provider. From there, that SAML token is used across all of the associated service providers to determine the user's identity and access. This specific method of using SSO starting at the service providers is known as a service provider-initiated flow.

Like many SSO systems, JBoss EAP utilizes both identity providers (IDP) and a service providers (SP). Both of these components are enabled to be run within JBoss EAP instances and work in conjunction with the JBoss EAP security subsystem. SAML v2, FORM based web application security, and HTTP/POST and HTTP/Redirect Bindings are also utilized to implement SSO.

To create an identity provider, a security domain is created (e.g. *idp-domain*) in an JBoss EAP instance with an authentication and authorization mechanism defined (e.g. LDAP, database, etc) to serve as the identity store. A web application (e.g. *IDP.war*) is configured to use additional modules (*org.picketlink*) required for running an IDP in conjunction with *idp-domain* and is deployed to that same JBoss EAP instance. *IDP.war* will serve as an identity provider. To create a service provider, a security domain is created (e.g. *sp-domain*) that uses **SAML2LoginModule** as an authentication mechanism. A web application (e.g. *SP.war*) is configured to use additional modules (*org.picketlink*) and contains a service provider valve that uses *sp-domain*. *SP.war* is deployed to an JBoss EAP instance where *sp-domain* is configured and is now a service provider. This process can be replicated for one or more service providers (e.g. *SP2.war*, *SP3.war*, etc) and across one or more JBoss EAP instances.

3.1.1. Identity Provider Initiated Flow

In most SSO scenarios, the SP starts the flow by sending an authentication request to the IDP, which in turns sends an SAML response to SP with a valid assertion. This is know as a service provider (SP)-initiated flow. But the SAML 2.0 specs also defines another flow, called identity provider (IDP)-initiated or Unsolicited Response flow. In this scenario, the service provider does not initiate the authentication flow to receive a SAML response from the IDP. The flow instead starts on the IDP-side. Once authenticated, the user can choose a specific service provider from a list and then get redirected to the service provider's URL. No special configuration is necessary to enable this flow.

Walkthrough

1. User accesses the IDP.
2. The IDP seeing that there is neither SAML request nor response, assumes an IDP-initiated flow scenario using SAML.
3. The IDP challenges the user to authenticate.

4. Upon authentication, the IDP shows the hosted section where the user gets a page that links to all the SP applications.
5. The user chooses an SP application.
6. The IDP redirects the user to the service provider with a SAML assertion in the query parameter, SAML response.
7. The SP checks the SAML assertion and provides access.

3.1.2. Global Logout

A Global Logout initiated at one service provider logs out the user from the Identity Provider (IDP) and all the service providers. If a user attempts to access secured portions of any SP or IDP after performing a global logout, they will be forced to re-authenticate.

3.2. DESKTOP-BASED SSO

A desktop-based SSO scenario enables a principal to be shared across both the desktop (usually governed by an [Active Directory](#) or [Kerberos](#) server) as well as a set of web applications (service providers). In this case, the desktop identity provider will also serve as the identity provider for the web applications.

In a typical setup, the user logs into a desktop which is governed by the Active Directory domain. The user then uses a web browser to access a web application that uses JBoss Negotiation hosted on the JBoss EAP. The web browser transfers the sign on information from the local machine of the user to the web application. JBoss EAP uses background GSS messages with the Active Directory or any Kerberos Server to validate the user. This enables the user to achieve a seamless SSO into the web application.

To setup a desktop-based SSO as an identity provider for a web application, a security domain is created that connects to the identity provider server. A NegotiationAuthenticator is then added as a valve to the desired web application and JBoss Negotiation is added to the SP container's class path. Alternatively, an IDP can be setup similarly to the browser-based SSO scenario, but using the desktop-based SSO provider as an identity store.

3.3. SSO USING STS

JBoss EAP 6 and above offers several login modules for service providers to connect to an STS. It can also run a Security Token Service (*PicketLinkSTS*). More specifically, the *PicketLinkSTS* defines several interfaces to other Security Token Services and provide extension points. Implementations can be plugged in via configuration, and the default values can be specified for some properties via configuration. This means that the *PicketLinkSTS* generates and manages the security tokens, but does not issue tokens of a specific type. Instead, it defines generic interfaces that allows multiple token providers to be plugged in. As a result, it can be configured to deal with various types of token, as long as a token provider exists for each token type. It also specifies the format of the security token request and response messages.

The following are the steps in which the security token requests are processed when using the JBoss EAP Security Token Service:

1. A client sends a security token request to *PicketLinkSTS*.
2. *PicketLinkSTS* parses the request message, generating a JAXB object model.

3. *PicketLinkSTS* reads the configuration file and creates the *STSTConfiguration* object, if needed. Then it obtains a reference to the *WSTrustRequestHandler* from the configuration and delegates the request processing to the handler instance.
4. The request handler uses the *STSTConfiguration* to set default values when needed (for example, when the request doesn't specify a token lifetime value).
5. The *WSTrustRequestHandler* creates the *WSTrustRequestContext*, setting the JAXB request object and the caller principal it received from *PicketLinkSTS*.
6. The *WSTrustRequestHandler* uses the *STSTConfiguration* to get the *SecurityTokenProvider* that must be used to process the request based on the type of the token that is being requested. Then it invokes the provider, passing the constructed *WSTrustRequestContext* as a parameter.
7. The *SecurityTokenProvider* instance process the token request and stores the issued token in the request context.
8. The *WSTrustRequestHandler* obtains the token from the context, encrypts it if needed, and constructs the WS-Trust response object containing the security token.
9. *PicketLinkSTS* dictates the response generated by the request handler and returns it to the client.

An STS Login Module (e.g. *STSIssuingLoginModule*, *STSTValidatingLoginModule*, *SAML2STSTLoginModule*, etc) is typically configured as part of the security setup of a JEE container to use a Security Token Service for authenticating users. The STS may be collocated on the same container as the Login Module or be accessed remotely through Web Service calls or another technology.

CHAPTER 4. EXAMPLE SCENARIOS

One way of understanding how JBoss EAP security and its components work together is to view in context of real scenarios. The following sections cover several generalized but realistic situations that involve the various JBoss EAP security components and configuration options. They will focus on how an application or set of applications is secured as well as how the management interfaces are secured.

4.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM OUT OF THE BOX

This scenario shows how JBoss EAP and a sample application are secured when no configuration changes are made after the initial install. An application (*sampleApp1.war*) is deployed and configured to use container-based security.



4.1.1. Core Management Authentication Out of the Box

4.1.1.1. Security

The core management authentication offers two pre-configured security realms by default: *ManagementRealm* and *ApplicationRealm*. These realms use property files to store the usernames, passwords, and roles. The *ManagementRealm*, which is used to store the authentication information the management APIs, also defines and enables local authentication for users using the CLI interface on the same host as the JBoss EAP instance. The *ApplicationRealm* is also pre-configured to store authentication and authorization information, but for use with other applications besides the management APIs. In addition, *ApplicationRealm* does come pre-configured with local authentication enabled but is not commonly used.

4.1.1.2. How it works

For this scenario, the following users have been added to a default installation of JBoss EAP:

Table 4.1. Users

Username	Password	Roles	Security Realm
Susan	<i>Testing123!</i>		ManagementRealm
Sarah	<i>Testing123!</i>	sample	ApplicationRealm
Frank	<i>Testing123!</i>		ApplicationRealm

On startup, the JBoss EAP instance loads both the *ManagementRealm* and *ApplicationRealm* security realms.

If *Susan* attempts to access either of the management APIs (HTTP or CLI), she is required to authenticate. Her username, password, and roles must match an entry in the *ManagementRealm* security realm. By default, no roles are required to access the management APIs. *Sarah* and *Frank* would not be able to access the management APIs since they are not in the *ManagementRealm* security realm.

4.1.2. Security Subsystem Out of the Box

4.1.2.1. Security

The security subsystem also comes pre-configured with three security domains: *other*, *jboss-web-policy*, and *jboss-ejb-policy*. The *other* security domain performs authentication and authorization by delegating to the realm specified in the login module (uses *ApplicationRealm* by default).

Additional information about both the default security realms as well as the default security domains can be found in the [Security Realms](#) and [Security Domains](#) sections.

4.1.2.2. How it Works

The application *sampleApp1.war* has two html files (*/hello.html* and */secure/hello.html*) and uses basic HTTP authentication to secure the path */secure/**. It uses the *other* security domain and requires the role of *sample*. Since the *other* security domain defers to *ApplicationRealm* for its authentication and authorization information, refer to the users in the *Users* table from the [previous section](#).

When *Sarah* requests */hello.html*, she is able to view the page without authenticating. When *Sarah* tries to request */secure/hello.html*, she is prompted to enter her username and password. She is then able to view */secure/hello.html* after successfully logging in. *Frank* and *Susan* (or any user) can access */hello.html*, but neither can access */secure/hello.html* (*Frank* does not have the *sample* role and *Susan* is not in the *ApplicationRealm* security realm).

4.2. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM WITH HTTPS AND RBAC ADDED TO THE MANAGEMENT INTERFACES

This scenario shows how JBoss EAP is secured when HTTPS is enabled for the management interfaces, and RBAC is added to the *ManagementRealm* security realm.



4.2.1. Security

JBoss EAP provides support for using HTTPS with the management interfaces, which includes the Management Console. HTTPS can be enabled by configuring the *secure-interface* element, adding a *secure-port* to the *http-interface* section of the management interface, and configuring an existing or new security realm with the desired settings (e.g. server-identities, protocol, keystore, alias, etc). This enables the management interfaces to use SSL/TLS for all HTTP traffic. For more background information on HTTPS and securing the management interfaces in general, see the [Section 2.2.2, “Advanced Security”](#) section.

JBoss EAP also offers the ability to enable [RBAC](#) on the management interfaces using a couple of different authentication schemes. This example uses username/password authentication with the existing property files used in the *ManagementRealm*. RBAC is enabled by setting the *provider* attribute to *rbac* for the management interface and updating the *ManagementRealm* with the desired users and roles.

4.2.2. How it works

For this scenario, the following users have been added to the existing security realms:

Table 4.2. Management Users

Username	Password	Security Realm
Suzy	<i>Testing123!</i>	<i>ManagementRealm</i>
Tim	<i>Testing123!</i>	<i>ManagementRealm</i>
Emily	<i>Testing123!</i>	<i>ManagementRealm</i>
Zach	<i>Testing123!</i>	<i>ManagementRealm</i>
Natalie	<i>Testing123!</i>	<i>ManagementRealm</i>

Based on group membership, the users have also been mapped to the following RBAC roles:

Table 4.3. RBAC Roles

Username	RBAC Role
Suzy	SuperUser
Tim	Administrator
Emily	Maintainer
Zach	Deployer
Natalie	Monitor

On startup, JBoss EAP loads the *ManagementRealm* (with the RBAC configuration) and management interfaces as part of the core services, which also starts the *http-interface* (configured for HTTPS) for the management interfaces. Users now access the management interfaces via HTTPS, and RBAC has also been enabled and configured. If RBAC is not enabled, any user in the *ManagementRealm* security realm is considered a *SuperUser* and has unlimited access. Since RBAC has been enabled, each user is now restricted based on the roles they have. *Suzy*, *Tim*, *Emily*, *Zach*, and *Natalie* all have different roles which are showing in the table above. For example, only *Suzy* and *Tim* can read and modify access control information. *Suzy*, *Tim*, and *Emily* can modify runtime state and other persistent configuration information. *Zach* can also modify runtime state and other persistent configuration information, but only related to application resources. *Suzy*, *Tim*, *Emily*, *Zach*, and *Natalie* can all read configuration and state information, but *Natalie* cannot update anything. For more details on each of the roles and how JBoss EAP handles RBAC, see the [Section 2.1.9, “Role-Based Access Control”](#) and [Section 2.2.2.3, “Adding RBAC to the Management Interfaces”](#) sections.

4.3. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM WITH AN UPDATED SECURITY SUBSYSTEM INCLUDING HTTPS

This scenario shows how a sample application running on JBoss EAP is secured when a new security domain is added, and HTTPS is enabled. An application (sampleApp2.war) is deployed that is configured to use container-based security (sample-domain) and HTTPS.



4.3.1. Security

JBoss EAP provides support for using HTTPS for use with applications, which is handled using the *web subsystem*. A new connector for HTTPS is added to the *web subsystem* and is configured with the desired settings (e.g. protocol, port, keystore, etc). Once the configuration is saved, web applications can start accepting HTTPS traffic on the configured port. A new security domain has also been added called *sample-domain* that uses the *IdentityLoginModule* for authentication. *sampleApp2.war* is configured to use *sample-domain* to authenticate users.

4.3.2. How it works

A security domain (*sample-domain*) has been created and configured to use the *IdentityLoginModule*. The following credentials have been configured in the login module:

Table 4.4. sample-domain Users

Username	Password	Roles
Vincent	<i>samplePass</i>	<i>sample</i>

On startup, JBoss EAP loads the core services, and then starts up the *web* and *security* subsystems which manages the HTTPS connections for all web applications and the *sample-domain* respectively. *sampleApp2.war* is then loaded which looks for *sample-domain* for providing authentication and authorization for its secured URLs. *sampleApp2.war* has two html files (*/hello.html* and */secure/hello.html*) and uses basic HTTP authentication to secure the path */secure/**. It uses the *sample-domain* security domain and requires the role of *sample*.

When *Vincent* requests */hello.html*, he is able to view the page without authenticating. When *Vincent* tries to request */secure/hello.html*, he is prompted to enter her username and password. He is then able to view */secure/hello.html* after successfully logging in. All other users can access */hello.html* without logging in, but none can access */secure/hello.html* since *Vincent* is the only user in *sample-domain*. This also applies to all traffic handled over HTTPS.

4.4. SSO FOR WEB APPLICATIONS ON RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

This scenario shows how web applications make use of both clustered and non-clustered SSO on JBoss EAP. Four JBoss EAP instances have are created (*EAP1*, *EAP2*, *EAP3*, and *EAP4*) with *EAP1* and *EAP2* operating in standalone mode, and *EAP3* and *EAP4* operating as a two node cluster. Two web applications (*sampleAppA* and *sampleAppB*) have been deployed to each of the four JBoss EAP instances.



4.4.1. Security

JBoss EAP provides support for both clustered and non-clustered SSO with web applications by using a combination of the *security*, *web*, and *infinispan* subsystems. The *security* subsystem provides a security domain for performing authentication and authorization while the *infinispan* and *web* subsystems help

with caching and distributing the SSO information between all the web applications on an JBoss EAP instance or across a JBoss EAP cluster. All four eap instances have a security domain (*sso-domain*) configured to use the *IdentityLoginModule*. *sampleAppA* and *sampleAppB* have been configured to use the *sso-domain* security domain to secure the path `/secure/*` and require the role of *sample* to access it. The following credentials have been configured in the *sso-domain* login module:

Table 4.5. sso-domain Users

Username	Password	Roles
Jane	<i>samplePass</i>	<i>sample</i>

All four JBoss EAP instances have also been configured to start up with either *standalone-full-ha* or *full-ha* profile, which adds the *infinispan* subsystem and other functionality needed for enabling SSO in this scenario. The web cache-container and SSO replicated-cache have also been added, and the *web* subsystem has been configured to use both them. *EAP1* and *EAP2* have configured their *web* subsystems for non-clustered SSO while the cluster containing *EAP3* and *EAP4* has been configured to use clustered SSO.



APPLICATION CLUSTERING VS. CLUSTERED SSO

There is a distinct difference between a clustered web application and clustered SSO. A clustered web application is one which is distributed across the nodes of a cluster to spread the load of hosting that application. In clustered applications (marked as distributable), all new sessions, and changes to existing sessions are replicated to other members of the cluster. Clustered SSO allows for replication of security context and identity information, regardless of whether or not the applications are themselves clustered. Although these technologies may be used together they are mutually exclusive and may be used independent of one other.

4.4.2. How it works

On startup, JBoss EAP loads the core services and then starts up the *security*, *web*, and *infinispan* subsystems which manage *sso-domain* and the associated caching for SSO information. *sampleAppA.war* and *sampleAppB.war* are loaded on all four JBoss EAP instances, each of which look for *sso-domain* for providing authentication and authorization.

If *Jane* attempts to access **`EAP1/sampleAppA/secure/hello.html`**, she will be asked to authenticate. After providing the correct information, she will be allowed to see **`EAP1/sampleAppA/secure/hello.html`**. *Jane's* session will then be added into the SSO caches used by the *web* and *infinispan* subsystems. If she then attempts to access **`EAP1/sampleAppB/secure/hello.html`**, she will not be asked to re-authenticate. *sampleAppB* running on *EAP1* will find her session using the *web* subsystem caches along with the *infinispan* subsystem and grant her access since she is already authenticated. If *Jane* then attempts to access either **`EAP2/sampleAppA/secure/hello.html`** or **`EAP2/sampleAppB/secure/hello.html`**, she will be asked to authenticate again since *EAP1* and *EAP2* do not share a cache.

If *Jane* attempts to access **`EAP3/sampleAppA/secure/hello.html`**, she will be asked to authenticate and her session will be stored in the SSO caches. These caches are stored across the entire cluster, so if *Jane* attempts to log into **`EAP3/sampleAppB/secure/hello.html`**, **`EAP4/sampleAppA/secure/hello.html`**, or **`EAP4/sampleAppB/secure/hello.html`**, she will not have to re-authenticate. Additionally, if either *EAP3* or *EAP4* are restarted, *Jane's* SSO information

will remain in the cache since the other JBoss EAP instance and the cluster remain running, hence preserving the cache. Similarly, if *Jane's* session is invalidated, it ripples across the cache and she will be asked to re-authenticate regardless of which application or server she tries to access in the cluster.

4.5. MULTIPLE RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM INSTANCES AND MULTIPLE APPLICATIONS USING BROWSER-BASED SSO WITH SAML

This scenario shows how multiple JBoss EAP are secured and browser-based SSO is added. Three separate (non-clustered) JBoss EAP instances are configured (*EAP1*, *EAP2*, and *EAP3*) and three sample applications (*sampleAppA.war*, *sampleAppB.war*, and *sampleAppC.war*) are configured to use browser-based SSO for authentication.



4.5.1. Security

JBoss EAP provides support for doing browser-based SSO via SAML with web applications as well as hosting an identity provider. To host an identity provider, a security domain (*idp-domain*) must be configured with an authentication mechanism defined. In this case, *idp-domain* is configured to use the **DatabaseLoginModule** as the authentication mechanism. The identity provider application (*IDP.war*) is deployed to that JBoss EAP instance (*EAP-IDP*), and configured to use *idp-domain* as its identity store. *IDP.war* uses the identity store in combination with functionality in the application to authenticate users as well as issue SAML tokens containing the users identity and role information. Three additional JBoss EAP instances (*EAP1*, *EAP2*, and *EAP3*) each host one distinct application that will serve as an individual service provider (*sampleAppA.war*, *sampleAppB.war*, and *sampleAppC* respectively). Each JBoss EAP instance has a security domain (*sso-domain*) configured to use the **SAML2LoginModule** as the authentication mechanism. Each application contains functionality to support SSO, use *IDP.war* as an identity provider, and use HTTP/POST binding for authentication. Each application is also configured to use *sso-domain* to secure the path */secure/** and supplies its own list of roles for handling authorization.

4.5.2. How it works

For this scenario, the following users have been added to the database used by the **DatabaseLoginModule** in the *idp-domain* security domain:

Table 4.6. idp-domain Users

Username	Password	Roles
Eric	<i>samplePass</i>	<i>all</i>
Amar	<i>samplePass</i>	<i>AB</i>
Brian	<i>samplePass</i>	<i>C</i>
Alan	<i>samplePass</i>	

On startup of *EAP-IDP*, the management interfaces start up, followed by the subsystems and deployed applications including the security subsystem (which includes *idp-domain*) and *IDP.war*. *idp-domain*

connects to the database and loads the usernames, passwords, and roles as configured in the **DatabaseLoginModule** login module. To prevent sensitive information from being stored in plaintext in the configuration of the **DatabaseLoginModule** login module password hashing is configured to obscure certain fields (e.g. password for the database). *IDP.war* uses *idp-domain* for authentication and authorization. *IDP.war* is also configured (via *jboss-web.xml*, *web.xml*, *picketlink.xml*, and *jboss-deployment-structure.xml*) to issue SAML tokens to authenticated users and supplies a simple login form for users to authenticate against. This allows it to serve as an identity provider.

On startup of *EAP1*, *EAP2* and *EAP3*, the management interfaces start up, followed by the subsystems and deployed applications including the security subsystem (which includes *sso-domain* on each instance) and *sampleAppA.war*, *sampleAppB.war*, and *sampleAppC.war* respectively. *sso-domain* is configured to use the **SAML2LoginModule** login module, but relies on the application to supply the proper SAML token. This means any application using *sso-domain* must handle properly connecting to an identity provider to obtain a SAML token. In this case, *sampleAppA*, *sampleAppB*, and *sampleAppC* are each configured (via *jboss-web.xml*, *web.xml*, *picketlink.xml*, and *jboss-deployment-structure.xml*) to obtain SAML tokens from an identity provider (*IDP.war*) using that identity providers login form.

Each application is also configured with its own set of allowed roles:

Table 4.7. Application/SP roles

Application/SP	Allowed Roles
<i>sampleAppA</i>	<i>all</i> , <i>AB</i>
<i>sampleAppB</i>	<i>all</i> , <i>AB</i>
<i>sampleAppC</i>	<i>all</i> , <i>C</i>

When an un-authenticated user attempts to access the URLs secured by *sso-domain* (i.e. */secure/**) of any application, that user is redirected to the login page at *IDP.war*. The user then authenticates using the form, is issued a SAML token containing their identity and role information. The user is then redirected back to the original URL, and their SAML token is presented to the application (service provider). The application ensures the SAML token is valid and then authorizes the user based on the roles provided by the SAML token and the ones configured in the application. Once a user is issued a SAML token, they will then use that token to be authenticated and authorized on the other applications secured by SSO using the same identity provider. Once the SAML token expires or becomes invalidated, the user will be required to obtain a new SAML token from the identity provider.

In this example, if Eric accesses ***EAP1/sampleAppA/secure/hello.html***, he is redirected to ***EAP-IDP/IDP/login.html*** to login. After successful login, he is issued a SAML token containing his user information and the role *all*, and redirected back to ***EAP1/sampleAppA/secure/hello.html***. His SAML token is then presented to *sampleAppA* to be checked and to authorize him based on his roles. Since *sampleAppA* allows the roles *all* and *AB* to access */secure/** and Eric has the role *all*, he is allowed to access ***EAP1/sampleAppA/secure/hello.html***.

If Eric then tries to access ***EAP2/sampleAppB/secure/hello.html***, since he is not already authenticated against that service provider, he is redirected again to *IDP.war* with an authentication request. Since Eric has already authenticated against the identity provider (*IDP.war*), he is then redirected back to ***EAP2/sampleAppB/secure/hello.html*** by *IDP.war* with a new SAML token for that service provider without having to re-authenticate. *sampleAppB* then simply checks his new SAML token and authorizes him based on his role. Since *sampleAppB* allows the roles *all* and *AB* to access

`/secure/*` and Eric has the role *all*, he also is allowed to access `EAP2/sampleAppB/secure/hello.html`. The same thing would apply if Eric were to try and access `EAP3/sampleAppC/secure/hello.html`.

If Eric were to return to `EAP1/sampleAppA/secure/hello.html` (or any URL under `EAP2/sampleAppB/secure/*` or `EAP3/sampleAppC/secure/*`) after his SAML token became invalidated via global logout, he would be redirected back to `EAP-IDP/IDP/login.html` to authenticate again and be issued a new SAML token.

If Amar attempted to access `EAP1/sampleAppA/secure/hello.html` or `EAP2/sampleAppB/secure/hello.html`, he would be directed through the same flow as Eric. If Amar attempted to access `EAP3/sampleAppC/secure/hello.html`, he would still be required to have or obtain a SAML token, but would be restricted from viewing `EAP3/sampleAppC/secure/hello.html` since his role *AB* only allows him to access `EAP1/sampleAppA/secure/*` and `EAP2/sampleAppB/secure/*`. Brian is in a similar situation, but he is only allowed to access `EAP3/sampleAppC/secure/*`. If Alan tries to access any service provider's secured area, he would still be required to have or obtain a SAML token, but would be restricted from seeing anything since he has no role. Each service provider also has unsecured area which any user (authorized or not) can view without obtaining a SAML token.

4.6. USING LDAP WITH THE *MANAGEMENTREALM*

This scenario shows the *ManagementRealm* using LDAP for securing the management interfaces. A JBoss EAP instance has been created (*EAP1*) and is running in standalone mode. The *ManagementRealm* on *EAP1* has also been updated to use LDAP as the authentication and authorization mechanism.



4.6.1. Security

JBoss EAP supports using LDAP (as well as Kerberos) for authentication in security realms. This is accomplished by updating the existing *ManagementRealm* to use **ldap** as the authentication type and creating an outbound connection to the LDAP server. This, in turn, changes the authentication mechanism from digest to BASIC / Plain, and will transmit usernames and passwords in the clear over the network by default.

4.6.2. How it works

The following users have been added to the LDAP directory:

Table 4.8. Management Users

Username	Password	Roles
Adam	<i>samplePass</i>	<i>SuperUser</i>
Sam	<i>samplePass</i>	

On startup, *EAP1* loads the core-services, including *ManagementRealm* and the management interfaces. *ManagementRealm* connects to the LDAP directory server and provides the authentication for the management interfaces as needed.

If *Adam* attempts to access a management interface, he will be forced to authenticate. His credentials will be passed to the *ManagementRealm* security realm, which will use the LDAP directory server for authentication. After he is authenticated, his roles will be passed back to the management interface and for authorization. Since *Adam* has the *SuperUser* role, he will be granted access to the management interface. If *Sam* attempts to access a management interface, he would be authenticated via LDAP, but would be denied access since he does not have the proper role of *SuperUser*.

4.7. USING DESKTOP SSO (VIA KERBEROS) TO PROVIDE SSO FOR WEB APPLICATIONS

This scenario shows how Kerberos can be used with JBoss EAP to provide SSO for web applications. A JBoss EAP instance has been created (*EAP1*) and is running in standalone mode. Two web applications (*sampleAppA* and *sampleAppB*) have been deployed to *EAP1*. Both the web applications and *EAP1* have been configured to authenticate using desktop-based SSO via Kerberos.



4.7.1. Security

JBoss EAP offers support for using Kerberos for SSO in web applications via SPNEGO and JBoss Negotiation. For more information on the specifics of Kerberos and SPNEGO please see the [Section 1.6.1, “Third-Party SSO Implementations”](#) section. To enable JBoss EAP and deployed web applications to use Kerberos for authentication, two security domains must be created. The first security domain (*host-domain*) is configured with the *kerberos* login module to connect to the kerberos server. This allows JBoss EAP to authenticate at the container level. A second security domain (*spnego-domain*) is created with two login modules. One uses the *spnego* login module to connect to *host-domain* to authenticate users. The second can use any other login module to load role information for use in authorization decisions (e.g. *UsersRoles* using properties files to map users to roles).

These two login modules also make use of *password-stacking* to map the users and roles in the authorization module with the users in the authentication login module. *EAP1* is configured with both *host-domain* and *spnego-domain*. Applications are then configured (via *web.xml* and *jboss-web.xml*) to use *spnego-domain* to perform authentication and get a user’s roles for authorization. Applications can also be configured with FORM authentication as a fallback authentication mechanism in case the security tokens cannot be passed from the OS to the browser. If FORM authentication is configured as a fallback, an additional security domain must be configured to support it. This security domain is independent of Kerberos and SPNEGO, and just has to support FORM authentication (i.e. username/password validation and roles). Each application is configured to use *spnego-domain* and to provide FORM authentication as a fallback. Each application is also configured to secure the path */secure/** and supplies its own list of roles for handling authorization.

4.7.2. How it works

The following users have been created in the Kerberos server:

Table 4.9. Kerberos Users

Username	Password
Brent	<i>samplePass</i>
Andy	<i>samplePass</i>

Username	Password
Bill	<i>samplePass</i>
Ron	<i>samplePass</i>

The following roles have mapped to the users via an additional module that is chained by setting the password-stacking option to *useFirstPass*:

Table 4.10. User Roles

Username	Roles
Brent	<i>all</i>
Andy	<i>A</i>
Bill	<i>B</i>
Ron	

The following roles have also been configured in each application:

Table 4.11. Application Roles

Application/SP	Allowed Roles
<i>sampleAppA</i>	<i>all, A</i>
<i>sampleAppB</i>	<i>all, B</i>

On startup, *EAP1* loads the core-services, followed by the *security* and other subsystems. *host-domain* establishes a connection to the Kerberos server and *spnego-domain* connects to *host-domain*. *sampleAppA* and *sampleAppB* are then deployed and connect to *spnego-domain* for authentication.

Brent has logged onto his computer that is secured with Kerberos. He then opens his browser and attempts to access **sampleAppA/secure/hello.html**. Since that is secured, authentication is required. *EAP1* directs the browser to send a request for a key to the Kerberos server (specifically the Kerberos Key Distribution Center which is configured on his computer). After a key is obtained by the browser, it is sent to *sampleAppA*. *sampleAppA* sends the ticket to *spnego-domain* where it is unpacked and authentication is performed by *host-domain* (in conjunction with the configured Kerberos server). Once the ticket is authenticated, *Brent's* role is passed back to *sampleAppA* to perform authorization. Since *Brent* has the *all* role, he will be able to access **sampleAppA/secure/hello.html**. If *Brent* tries to access **sampleAppB/secure/hello.html**, the same process will occur and he will be granted access (due to him having the *all* role). *Andy* and *Bill* would follow the same process as well, but with *Andy* only having access to **sampleAppA/secure/hello.html** and not **sampleAppB/secure/hello.html**. *Bill* would be the opposite, having access to

sampleAppB/secure/hello.html and not **sampleAppA/secure/hello.html**. *Ron* would pass authentication to either **sampleAppA/secure/hello.html** or **sampleAppB/secure/hello.html**, but would not be granted access to either since he has no role.

If *Andy* were to attempt to access **sampleAppA/secure/hello.html** from a computer not secured by Kerberos (e.g. a personal laptop connected to the office network), he would be directed to the FORM login page as a fallback login mechanism. His credentials would then be authenticated via the fallback security domain and then the process would continue with authorization.