JBoss Enterprise Application Platform Common Criteria Certification 6.2.2

Security Guide

For Use with Red Hat JBoss Enterprise Application Platform 6

Last Updated: 2017-11-17

JBoss Enterprise Application Platform Common Criteria Certification 6.2.2 Security Guide

For Use with Red Hat JBoss Enterprise Application Platform 6

Nidhi Chaudhary

Lucas Costi

Russell Dickenson

Sande Gilda

Vikram Goyal

Eamon Logue

Darrin Mison

Scott Mumford

David Ryan

Misty Stanley-Jones

Keerat Verma

Tom Wells

Nichola Moore

Nidhi Srinivas

Abstract

This book is a guide to securing Red Hat JBoss Enterprise Application Platform 6 and its patch releases.

Table of Contents

PART I. SECURITY FOR RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

CHAPTER 1. INTRODUCTION

1.1. ABOUT RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6 (JBOSS EAP 6)

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) is a fast, secure, powerful middleware platform built upon open standards, and compliant with the Java Enterprise Edition 6 specification. It integrates JBoss Application Server 7 with high-availability clustering, powerful messaging, distributed caching, and other technologies to create a stable and scalable platform.

The new modular structure allows for services to be enabled only when required, significantly increasing start up speed. The Management Console and Management Command Line Interface remove the need to edit XML configuration files by hand, adding the ability to script and automate tasks. In addition, it includes APIs and development frameworks that can be used to develop secure, powerful, and scalable Java EE applications quickly.

Report a bug

1.2. ABOUT SECURING JBOSS ENTERPRISE APPLICATION PLATFORM 6

Computer security is the all encompassing term given to the field of information technology that deals with securing the virtual environments that power the digital age. This can include data protection and integrity, application security, risk and vulnerability assessment and authentication and authorization protocols.

Computer data is an all important asset for most organizations. Data protection is vital and forms the core of most businesses. JBoss EAP 6 provides a multi-layered approach to security to take care of data at all stages.

Truly secure systems are the ones that are designed from the ground up with security as the main feature. Such systems use the principle of Security by Design. In such systems, malicious attacks and infiltration's are accepted as part and parcel of normal security apparatus and systems are designed to work around them.

Security can be applied at the operating system, middleware and application level. For more information about security at the operating system level as it applies to RHEL, refer to the Red Hat Enterprise Linux Security Guide.

In the coming chapters, you will read about the different levels and layers of security within JBoss EAP 6. These layers provides the infrastructure for all security functionality within the platform.

Report a bug

CHAPTER 2. SECURITY OVERVIEW

2.1. ABOUT DECLARATIVE SECURITY

Declarative security is a method to separate security concerns from your 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.

Report a bug

2.1.1. Java EE Declarative Security Overview

The J2EE security model is declarative in that you describe the security roles and permissions in a standard XML descriptor rather than embedding security into your business component. This isolates security from business-level code because security tends to be more a function of where the component is deployed than an inherent aspect of the component's business logic. For example, consider an Automated Teller Machine (ATM) that is to be used to access a bank account. The security requirements, roles and permissions will vary independent of how you access the bank account, based on what bank is managing the account, where the ATM is located, and so on.

Securing a J2EE application is based on the specification of the application security requirements via the standard J2EE deployment descriptors. You secure access to EJBs and web components in an enterprise application by using the **`ejb-jar.xml`** and **`web.xml`** deployment descriptors.

Report a bug

2.1.2. Security References

Both Enterprise Java Beans (EJBs) and servlets can declare one or more <security-role-ref> elements.

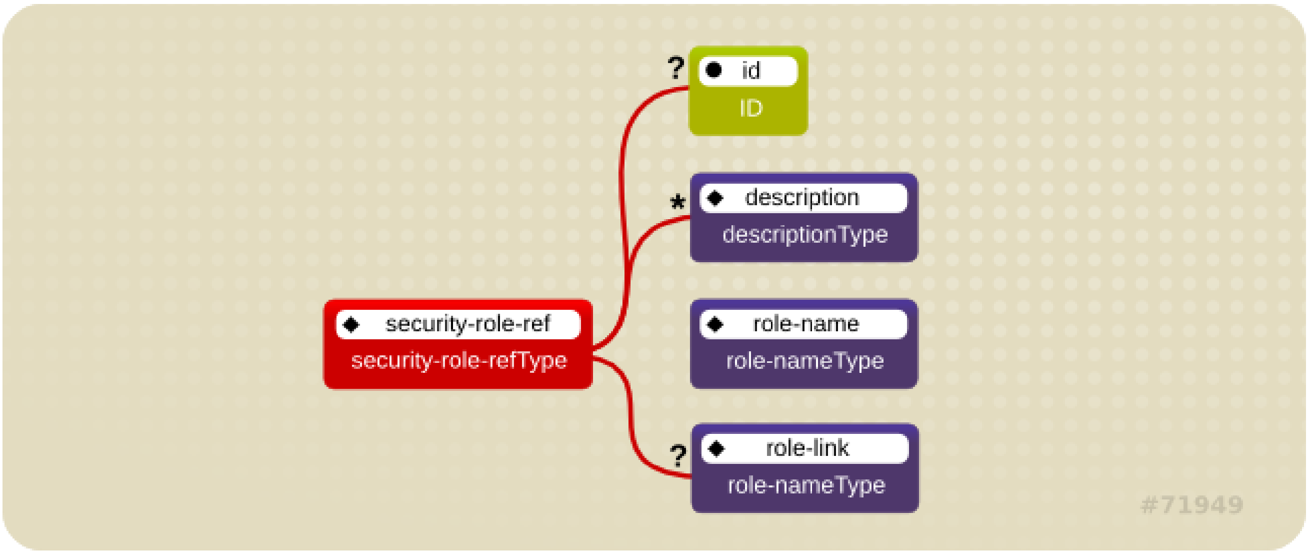


Figure 2.1. Security Roles Reference Model

This element declares that a component is using the <role-name> element's **`role-nameType`** attribute value as an argument to the **`isCallerInRole(String)`** method. By using the **`isCallerInRole`** method, a component can verify whether the caller is in a role that has been declared with a <security-

role-ref> or <role-name> element. The <role-name> element value must link to a <security-role> element through the <role-link> element. The typical use of **`isCallerInRole`** is to perform a security check that cannot be defined by using the role-based <method-permissions> elements.

Example 2.1. ejb-jar.xml descriptor fragment

```
    <!-- A sample ejb-jar.xml fragment -->
      <ejb-jar>
        <enterprise-beans>
          <session>
            <ejb-name>ASessionBean</ejb-name>
            ...
            <security-role-ref>
              <role-name>TheRoleICheck<role-name>
                <role-link>TheApplicationRole</role-link>
            </security-role-ref>
          </session>
        </enterprise-beans>
      ...
      </ejb-jar>
```

NOTE

This fragment is an example only. In deployments, the elements in this section must contain role names and links relevant to the EJB deployment.

Example 2.2. web.xml descriptor fragment

```
  <web-app>
    <servlet>
      <servlet-name>AServlet</servlet-name>
      ...
      <security-role-ref>
        <role-name>TheServletRole</role-name>
        <role-link>TheApplicationRole</role-link>
      </security-role-ref>
    </servlet>
      ...
  </web-app>
```

Report a bug

2.1.3. Security Identity

An Enterprise Java Bean (EJB) can specify the identity another EJB must use when it invokes methods on components using the <security-identity> element.

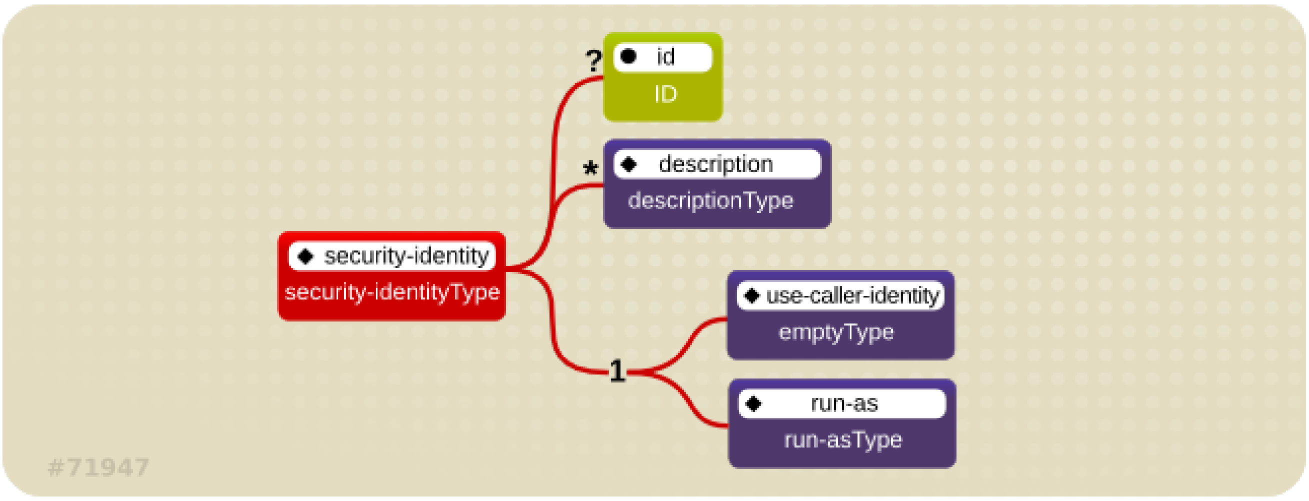


Figure 2.2. J2EE Security Identity Data Model

The invocation identity can be that of the current caller, or it can be a specific role. The application assembler uses the <security-identity> element with a <use-caller-identity> child element. This indicate that the current caller's identity should be propagated as the security identity for method invocations made by the EJB. Propagation of the caller's identity is the default used in the absence of an explicit <security-identity> element declaration.

Alternatively, the application assembler can use the <run-as> or <role-name> child element to specify that a specific security role supplied by the <role-name> element value must be used as the security identity for method invocations made by the EJB.

Note that this does not change the caller's identity as seen by the **`EJBContext.getCallerPrincipal()`** method. Rather, the caller's security roles are set to the single role specified by the <run-as> or <role-name> element value.

One use case for the <run-as> element is to prevent external clients from accessing internal EJBs. You configure this behavior by assigning the internal EJB <method-permission> elements, which restrict access to a role never assigned to an external client. EJBs that must in turn use internal EJBs are then configured with a <run-as> or <role-name> equal to the restricted role. The following descriptor fragment describes an example<security-identity> element usage.

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>ASessionBean</ejb-name>
            <!-- ... -->
            <security-identity>
                <use-caller-identity/>
            </security-identity>
        </session>
        <session>
            <ejb-name>RunAsBean</ejb-name>
            <!-- ... -->
            <security-identity>
                <run-as>
                    <description>A private internal role</description>
                    <role-name>InternalRole</role-name>
                </run-as>
            </security-identity>
        </session>
    </enterprise-beans>
    <!-- ... -->
</ejb-jar>
```

When you use <run-as> to assign a specific role to outgoing calls, a principal named **anonymous** is assigned to all outgoing calls. If you want another principal to be associated with the call, you must associate a <run-as-principal> with the bean in the **jboss.xml** file. The following fragment associates a principal named **internal** with **RunAsBean** from the prior example.

```

<session>
    <ejb-name>RunAsBean</ejb-name>
    <security-identity>
        <run-as-principal>internal</run-as-principal>
    </security-identity>
</session>
```

The <run-as> element is also available in servlet definitions in a **web.xml** file. The following example shows how to assign the role **InternalRole** to a servlet:

```

  <servlet>
    <servlet-name>AServlet</servlet-name>
    <!-- ... -->
    <run-as>
        <role-name>InternalRole</role-name>
    </run-as>
  </servlet>
```

Calls from this servlet are associated with the anonymous **principal**. The <run-as-principal> element is available in the **jboss-web.xml** file to assign a specific principal to go along with the **run-as** role. The following fragment shows how to associate a principal named **internal** to the servlet above.

```

  <servlet>
    <servlet-name>AServlet</servlet-name>
    <run-as-principal>internal</run-as-principal>
  </servlet>
```

Report a bug

2.1.4. Security Roles

The security role name referenced by either the **security-role-ref** or **security-identity** element needs to map to one of the application's declared roles. An application assembler defines logical security roles by declaring **security-role** elements. The **role-name** value is a logical application role name like Administrator, Architect, SalesManager, etc.

The J2EE specifications note that it is important to keep in mind that the security roles in the deployment descriptor are used to define the logical security view of an application. Roles defined in the J2EE deployment descriptors should not be confused with the user groups, users, principals, and other concepts that exist in the target enterprise's operational environment. The deployment descriptor roles are application constructs with application domain-specific names. For example, a banking application might use role names such as BankManager, Teller, or Customer.

In JBoss EAP, a **`security-role`** element is only used to map **`security-role-ref/role-name`** values to the logical role that the component role references. The user's assigned roles are a dynamic function of the application's security manager. JBoss does not require the definition of **`security-role`** elements in order to declare method permissions. However, the specification of **`security-role`** elements is still a recommended practice to ensure portability across application servers and for deployment descriptor maintenance.

Example 2.3. An ejb-jar.xml descriptor fragment that illustrates the security-role element usage.

```
<!-- A sample ejb-jar.xml fragment -->
<ejb-jar>
    <assembly-descriptor>
        <security-role>
            <description>The single application role</description>
            <role-name>TheApplicationRole</role-name>
        </security-role>
    </assembly-descriptor>
</ejb-jar>
```

Example 2.4. An example web.xml descriptor fragment that illustrates the security-role element usage.

```
<!-- A sample web.xml fragment -->
<web-app>
  <security-role>
    <description>The single application role</description>
    <role-name>TheApplicationRole</role-name>
  </security-role>
</web-app>
```

Report a bug

2.1.5. EJB Method Permissions

An application assembler can set the roles that are allowed to invoke an EJB's home and remote interface methods through method-permission element declarations.

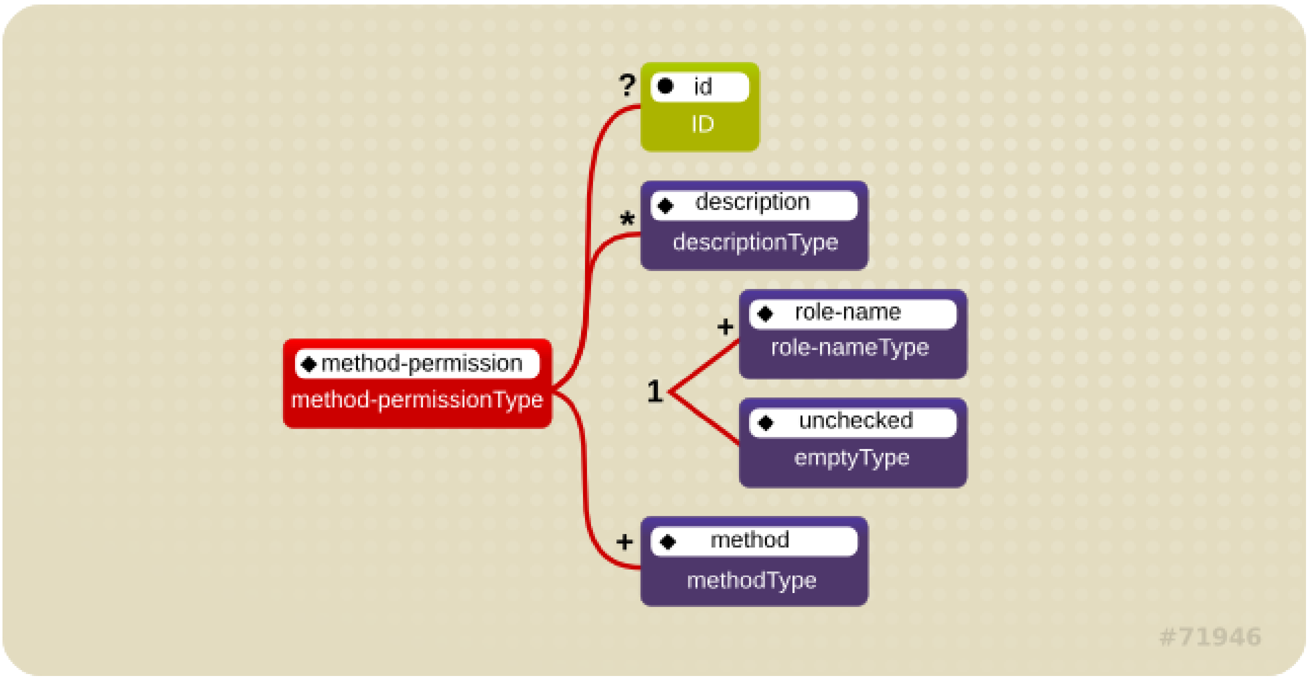


Figure 2.3. J2EE Method Permissions Element

Each **`method-permission`** element contains one or more role-name child elements that define the logical roles that are allowed to access the EJB methods as identified by method child elements. You can also specify an **`unchecked`** element instead of the **`role-name`** element to declare that any authenticated user can access the methods identified by method child elements. In addition, you can declare that no one should have access to a method that has the **`exclude-list`** element. If an EJB has methods that have not been declared as accessible by a role using a **`method-permission`** element, the EJB methods default to being excluded from use. This is equivalent to defaulting the methods into the **`exclude-list`**.

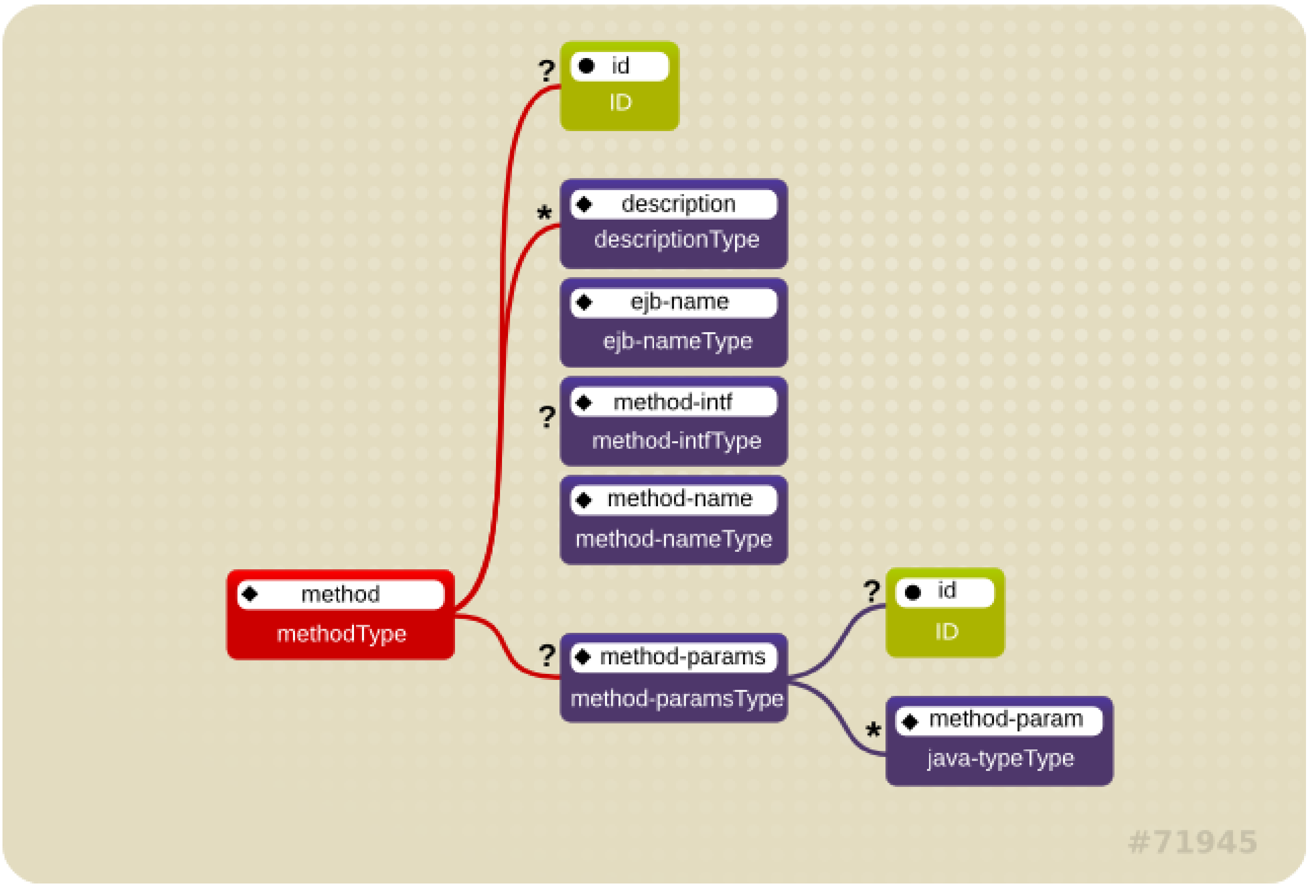


Figure 2.4. J2EE Method Element

There are three supported styles of method element declarations.

The first is used for referring to all the home and component interface methods of the named enterprise bean:

```
<method>
  <ejb-name>EJBNAME</ejb-name>
  <method-name>*</method-name>
</method>
```

The second style is used for referring to a specified method of the home or component interface of the named enterprise bean:

```
  <method>
    <ejb-name>EJBNAME</ejb-name>
    <method-name>METHOD</method-name>
  </method>
```

If there are multiple methods with the same overloaded name, this style refers to all of the overloaded methods.

The third style is used to refer to a specified method within a set of methods with an overloaded name:

```
<method>
    <ejb-name>EJBNAME</ejb-name>
    <method-name>METHOD</method-name>
    <method-params>
        <method-param>PARAMETER_1</method-param>
        <!-- ... -->
        <method-param>PARAMETER_N</method-param>
    </method-params>
</method>
```

The method must be defined in the specified enterprise bean's home or remote interface. The method-param element values are the fully qualified name of the corresponding method parameter type. If there are multiple methods with the same overloaded signature, the permission applies to all of the matching overloaded methods.

The optional **`method-intf`** element can be used to differentiate methods with the same name and signature that are defined in both the home and remote interfaces of an enterprise bean.

Example 2.5, “An ejb-jar.xml descriptor fragment that illustrates the method-permission element usage.” provides complete examples of the **`method-permission`** element usage.

Example 2.5. An ejb-jar.xml descriptor fragment that illustrates the method-permission element usage.

```
<ejb-jar>
    <assembly-descriptor>
        <method-permission>
            <description>The employee and temp-employee roles may
access any
                method of the EmployeeService bean </description>
            <role-name>employee</role-name>
            <role-name>temp-employee</role-name>
            <method>
                <ejb-name>EmployeeService</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <method-permission>
            <description>The employee role may access the
findByPrimaryKey,
                getEmployeeInfo, and the updateEmployeeInfo(String)
method of
                the AardvarkPayroll bean </description>
            <role-name>employee</role-name>
            <method>
                <ejb-name>AardvarkPayroll</ejb-name>
                <method-name>findByPrimaryKey</method-name>
            </method>
            <method>
                <ejb-name>AardvarkPayroll</ejb-name>
                <method-name>getEmployeeInfo</method-name>
            </method>
            <method>
                <ejb-name>AardvarkPayroll</ejb-name>
                <method-name>updateEmployeeInfo</method-name>
                <method-params>
                    <method-param>java.lang.String</method-param>
                </method-params>
            </method>
        </method-permission>
        <method-permission>
```

```
            <description>The admin role may access any method of the
                EmployeeServiceAdmin bean </description>
            <role-name>admin</role-name>
            <method>
                <ejb-name>EmployeeServiceAdmin</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <method-permission>
            <description>Any authenticated user may access any method
of the
                EmployeeServiceHelp bean</description>
            <unchecked/>
            <method>
                <ejb-name>EmployeeServiceHelp</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <exclude-list>
            <description>No fireTheCTO methods of the EmployeeFiring
bean may be
                used in this deployment</description>
            <method>
                <ejb-name>EmployeeFiring</ejb-name>
                <method-name>fireTheCTO</method-name>
            </method>
        </exclude-list>
    </assembly-descriptor>
</ejb-jar>
```

Report a bug

2.1.6. Enterprise Beans Security Annotations

Enterprise beans use Annotations to pass information to the deployer about security and other aspects of the application. The deployer can set up the appropriate enterprise bean security policy for the application if specified in annotations, or the deployment descriptor.

Any method values explicitly specified in the deployment descriptor override annotation values. If a method value is not specified in the deployment descriptor, those values set using annotations are used. The overriding granularity is on a per-method basis

Those annotations that address security and can be used in an enterprise beans include the following:

`@DeclareRoles`

Declares each security role declared in the code. For information about configuring roles, refer to the *Java EE 5 Tutorial* Declaring Security Roles Using Annotations.

`@RolesAllowed`*, *`@PermitAll`*, and *`@DenyAll`

Specifies method permissions for annotations. For information about configuring annotation method permissions, refer to the *Java EE 5 Tutorial* Specifying Method Permissions Using Annotations.

`@RunAs`

Configures the propagated security identity of a component. For information about configuring propagated security identities using annotations, refer to the *Java EE 5 Tutorial* Configuring a Component's Propagated Security Identity.

Report a bug

2.1.7. Web Content Security Constraints

In a web application, security is defined by the roles that are allowed access to content by a URL pattern that identifies the protected content. This set of information is declared by using the **`web.xml`**security-constraint element.

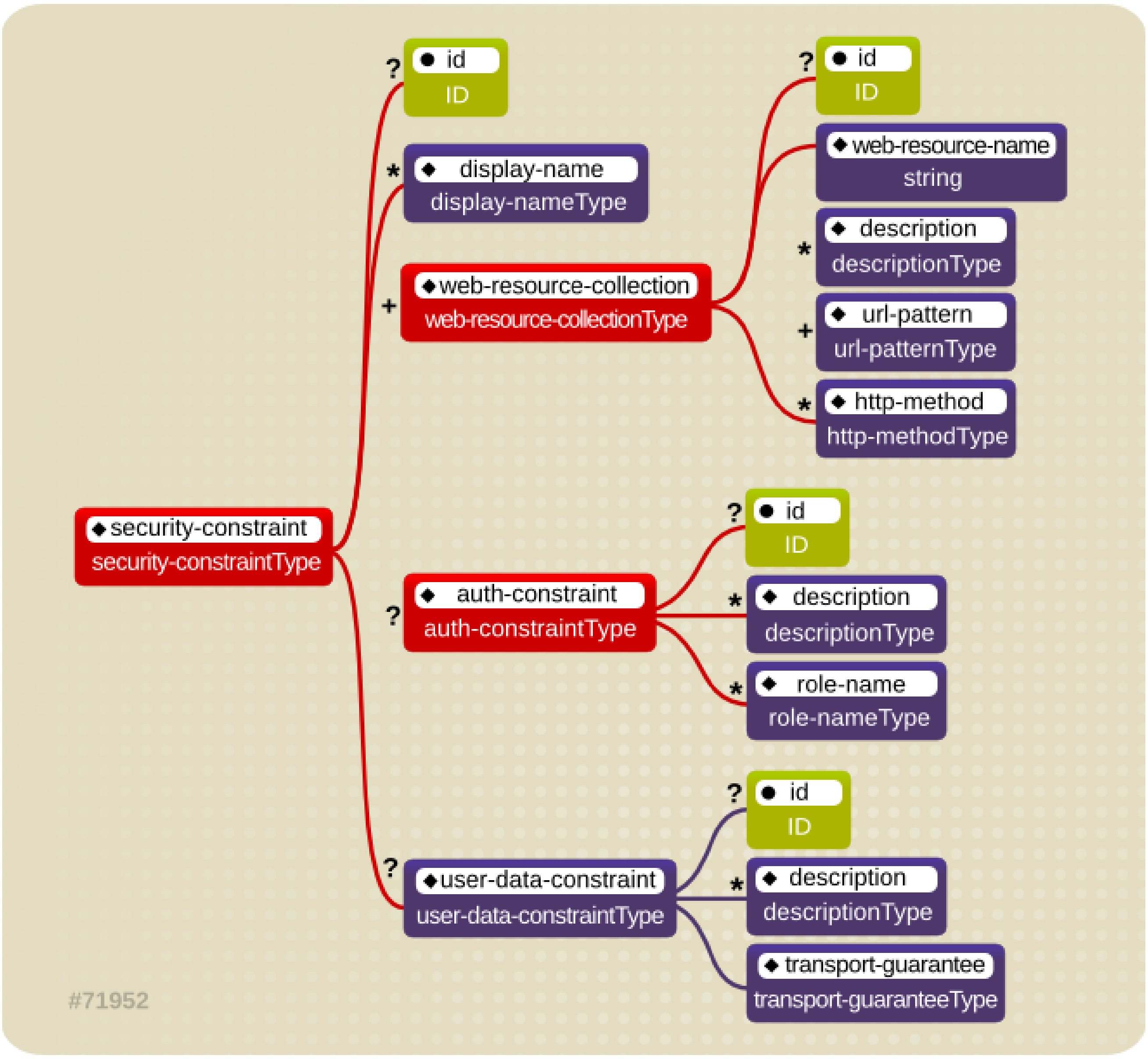


Figure 2.5. Web Content Security Constraints

The content to be secured is declared using one or more <web-resource-collection> elements. Each <web-resource-collection> element contains an optional series of <url-pattern> elements followed by an optional series of <http-method> elements. The <url-pattern> element value specifies a URL pattern against which a request URL must match for the request to correspond to an attempt to access secured content. The <http-method> element value specifies a type of HTTP request to allow.

The optional <user-data-constraint> element specifies the requirements for the transport layer of the client to server connection. The requirement may be for content integrity (preventing data tampering in the communication process) or for confidentiality (preventing reading while in transit). The <transport-guarantee> element value specifies the degree to which communication between the client and server should be protected. Its values are **`NONE`**, **`INTEGRAL`**, and **`CONFIDENTIAL`**. A value of **`NONE`** means that the application does not require any transport guarantees. A value of **`INTEGRAL`** means that the application requires the data sent between the client and server to be sent in such a way that it can not be changed in transit. A value of **`CONFIDENTIAL`** means that the application requires the data to be transmitted in a fashion that prevents other entities from observing the contents of the transmission. In most cases, the presence of the **`INTEGRAL`** or **`CONFIDENTIAL`** flag indicates that the use of SSL is required.

The optional <login-config> element is used to configure the authentication method that should be used, the realm name that should be used for the application, and the attributes that are needed by the form login mechanism.

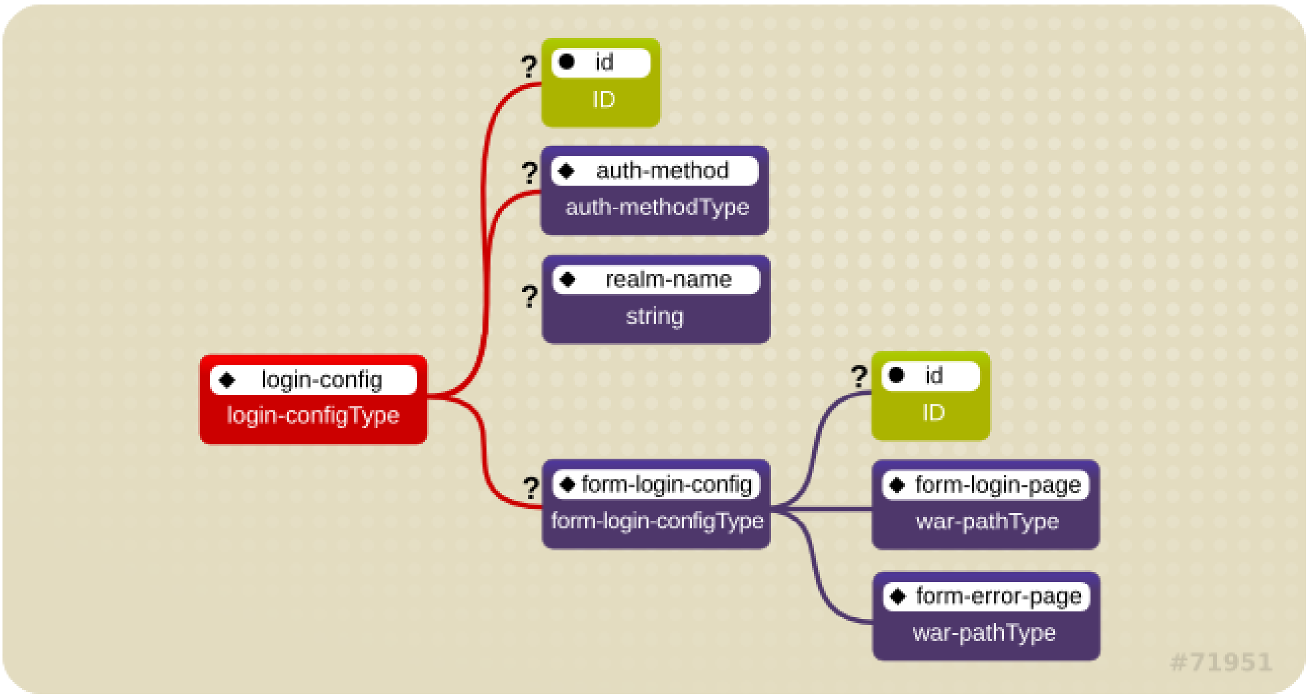


Figure 2.6. Web Login Configuration

The <auth-method> child element specifies the authentication mechanism for the web application. As a prerequisite to gaining access to any web resources that are protected by an authorization constraint, a user must have authenticated using the configured mechanism. Legal <auth-method> values are **`BASIC`**, **`DIGEST`**, **`FORM`**, and **`CLIENT-CERT`**. The <realm-name> child element specifies the realm name to use in HTTP basic and digest authorization. The <form-login-config> child element specifies the log in as well as error pages that should be used in form-based log in. If the <auth-method> value is not **`FORM`**, then **`form-login-config`** and its child elements are ignored.

The following configuration example indicates that any URL lying under the web application's **`/restricted`** path requires an **`AuthorizedUser`** role. There is no required transport guarantee and the authentication method used for obtaining the user identity is BASIC HTTP authentication.

Example 2.6. web.xml Descriptor Fragment

```
<web-app>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Secure Content</web-resource-name>
```

```
                <url-pattern>/restricted/*</url-pattern>
            </web-resource-collection>
            <auth-constraint>
                <role-name>AuthorizedUser</role-name>
            </auth-constraint>
            <user-data-constraint>
                <transport-guarantee>NONE</transport-guarantee>
            </user-data-constraint>
        </security-constraint>
        <!-- ... -->
        <login-config>
            <auth-method>BASIC</auth-method>
            <realm-name>The Restricted Zone</realm-name>
        </login-config>
        <!-- ... -->
        <security-role>
            <description>The role required to access restricted content
    </description>
            <role-name>AuthorizedUser</role-name>
        </security-role>
    </web-app>
```

Report a bug

2.1.8. Enable Form-based Authentication

Form-based authentication provides flexibility in defining a custom JSP/HTML page for log in, and a separate page to which users are directed if an error occurs during login.

Form-based authentication is defined by including **`<auth-method>FORM</auth-method>`** in the <login-config> element of the deployment descriptor, **`web.xml`**. The login and error pages are also defined in <login-config>, as follows:

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

When a web application with form-based authentication is deployed, the web container uses **`FormAuthenticator`** to direct users to the appropriate page. JBoss EAP maintains a session pool so that authentication information does not need to be present for each request. When **`FormAuthenticator`** receives a request, it queries **`org.apache.catalina.session.Manager`** for an existing session. If no session exists, a new session is created. **`FormAuthenticator`** then verifies the credentials of the session.

NOTE

Each session is identified by a session ID, a 16 byte string generated from random values. These values are retrieved from **`/dev/urandom`** (Linux) by default, and hashed with MD5. Checks are performed at session ID creation to ensure that the ID created is unique.

Once verified, the session ID is assigned as part of a cookie, and then returned to the client. This cookie is expected in subsequent client requests and is used to identify the user session.

The cookie passed to the client is a name value pair with several optional attributes. The identifier attribute is called **`JSESSIONID`** . Its value is a hex-string of the session ID. This cookie is configured to be non-persistent. This means that on the client side it will be deleted when the browser exits. On the server side, sessions expire after 60 seconds of inactivity, at which time session objects and their credential information are deleted.

Say a user attempts to access a web application that is protected with form-based authentication. **`FormAuthenticator`** caches the request, creates a new session if necessary, and redirects the user to the login page defined in **`login-config`**. (In the previous example code, the login page is **`login.html`**.) The user then enters their user name and password in the HTML form provided. User name and password are passed to **`FormAuthenticator`** via the **`j_security_check`** form action.

The **`FormAuthenticator`** then authenticates the user name and password against the realm attached to the web application context. In JBoss Enterprise Application Platform, the realm is **`JBossWebRealm`**. When authentication is successful, **`FormAuthenticator`** retrieves the saved request from the cache and redirects the user to their original request.

NOTE

The server recognizes form authentication requests only when the URI ends with **`/j_security_check`** and at least the **`j_username`** and **`j_password`** parameters exist.

Report a bug

2.1.9. Enable Declarative Security

The Java EE security elements that have been covered so far describe the security requirements only from the application's perspective. Because Java EE security elements declare logical roles, the application deployer maps the roles from the application domain onto the deployment environment. The Java EE specifications omit these application server-specific details.

To map application roles onto the deployment environment, you must specify a security manager that implements the Java EE security model using JBoss EAP-specific deployment descriptors. Refer to the custom login module example for details of this security configuration.

Report a bug

CHAPTER 3. INTRODUCTION TO JAAS

3.1. ABOUT JAAS

The JBossSX framework is based on the JAAS API. You must understand the basic elements of the JAAS API before you can understand the implementation details of JBossSX. The following sections provide an introduction to JAAS to prepare you for the JBossSX architecture discussion later in this guide.

The JAAS 1.0 API consists of a set of Java packages designed for user authentication and authorization. The API implements a Java version of the standard Pluggable Authentication Modules (PAM) framework and extends the Java 2 Platform access control architecture to support user-based authorization.

JAAS was first released as an extension package for JDK 1.3 and is bundled with JDK 1.5. Because the JBossSX framework only uses the authentication capabilities of JAAS to implement the declarative role-based J2EE security model, this introduction focuses on only that topic.

JAAS authentication is performed in a pluggable fashion. This permits Java applications to remain independent from underlying authentication technologies, and allows the JBossSX security manager to work in different security infrastructures. Integration with a security infrastructure is achievable without changing the JBossSX security manager implementation. You need only change the configuration of the authentication stack JAAS uses.

Report a bug

3.2. JAAS CORE CLASSES

The JAAS core classes can be broken down into three categories: common, authentication, and authorization. The following list presents only the common and authentication classes because these are the specific classes used to implement the functionality of JBossSX covered in this chapter.

These are the common classes:

- **`Subject`** (**`javax.security.auth.Subject`**)

These are the authentication classes:

- **`Configuration`** (**`javax.security.auth.login.Configuration`**)
- **`LoginContext`** (**`javax.security.auth.login.LoginContext`**)

These are the associated interfaces:

- **`Principal`** (**`java.security.Principal`**)
- **`Callback`** (**`javax.security.auth.callback.Callback`**)
- **`CallbackHandler`** (**`javax.security.auth.callback.CallbackHandler`**)
- **`LoginModule`** (**`javax.security.auth.spi.LoginModule`**)

Report a bug

3.3. SUBJECT AND PRINCIPAL CLASSES

To authorize access to resources, applications must first authenticate the request's source. The JAAS framework defines the term subject to represent a request's source. The **`Subject`** class is the central class in JAAS. A **`Subject`** represents information for a single entity, such as a person or service. It encompasses the entity's principals, public credentials, and private credentials. The JAAS APIs use the existing Java 2 **`java.security.Principal`** interface to represent a principal, which is essentially a typed name.

During the authentication process, a subject is populated with associated identities, or principals. A subject may have many principals. For example, a person may have a name principal (John Doe), a social security number principal (123-45-6789), and a user name principal (johnd), all of which help distinguish the subject from other subjects. To retrieve the principals associated with a subject, two methods are available:

```
public Set getPrincipals() {...}
public Set getPrincipals(Class c) {...}
```

`getPrincipals()` returns all principals contained in the subject. **`getPrincipals(Class c)`** returns only those principals that are instances of class **`c`** or one of its subclasses. An empty set is returned if the subject has no matching principals.

Note that the **`java.security.acl.Group`** interface is a sub-interface of **`java.security.Principal`**, so an instance in the principals set may represent a logical grouping of other principals or groups of principals.

Report a bug

3.4. SUBJECT AUTHENTICATION

Subject Authentication requires a JAAS login. The login process consists of the following points:

1. An application instantiates a **`LoginContext`** and passes in the name of the login configuration and a **`CallbackHandler`** to populate the **`Callback`** objects, as required by the configuration **`LoginModule`**s.

2. The **`LoginContext`** consults a **`Configuration`** to load all the **`LoginModules`** included in the named login configuration. If no such named configuration exists the **`other`** configuration is used as a default.

3. The application invokes the **`LoginContext.login`** method.

4. The login method invokes all the loaded **`LoginModule`**s. As each **`LoginModule`** attempts to authenticate the subject, it invokes the handle method on the associated **`CallbackHandler`** to obtain the information required for the authentication process. The required information is passed to the handle method in the form of an array of **`Callback`** objects. Upon success, the **`LoginModule`**s associate relevant principals and credentials with the subject.

5. The **`LoginContext`** returns the authentication status to the application. Success is represented by a return from the login method. Failure is represented through a LoginException being thrown by the login method.

6. If authentication succeeds, the application retrieves the authenticated subject using the **`LoginContext.getSubject`** method.

7. After the scope of the subject authentication is complete, all principals and related information associated with the subject by the **`login`** method can be removed by invoking the

LoginContext.logout method.

The **LoginContext** class provides the basic methods for authenticating subjects and offers a way to develop an application that is independent of the underlying authentication technology. The **LoginContext** consults a **Configuration** to determine the authentication services configured for a particular application. **LoginModule** classes represent the authentication services. Therefore, you can plug different login modules into an application without changing the application itself. The following code shows the steps required by an application to authenticate a subject.

```
CallbackHandler handler = new MyHandler();
LoginContext lc = new LoginContext("some-config", handler);

try {
    lc.login();
    Subject subject = lc.getSubject();
} catch(LoginException e) {
    System.out.println("authentication failed");
    e.printStackTrace();
}

// Perform work as authenticated Subject
// ...

// Scope of work complete, logout to remove authentication info
try {
    lc.logout();
} catch(LoginException e) {
    System.out.println("logout failed");
    e.printStackTrace();
}

// A sample MyHandler class
class MyHandler
    implements CallbackHandler
{
    public void handle(Callback[] callbacks) throws
        IOException, UnsupportedCallbackException
    {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof NameCallback) {
                NameCallback nc = (NameCallback)callbacks[i];
                nc.setName(username);
            } else if (callbacks[i] instanceof PasswordCallback) {
                PasswordCallback pc = (PasswordCallback)callbacks[i];
                pc.setPassword(password);
            } else {
                throw new UnsupportedCallbackException(callbacks[i],
                                                       "Unrecognized
Callback");
            }
        }
    }
}
```

Developers integrate with an authentication technology by creating an implementation of the **LoginModule** interface. This allows an administrator to plug different authentication technologies into an application. You can chain together multiple **`LoginModule`**s to allow for more than one authentication technology to participate in the authentication process. For example, one **`LoginModule`** may perform user name/password-based authentication, while another may interface to hardware devices such as smart card readers or biometric authenticators.

The life cycle of a **`LoginModule`** is driven by the **`LoginContext`** object against which the client creates and issues the login method. The process consists of two phases. The steps of the process are as follows:

- The **`LoginContext`** creates each configured **`LoginModule`** using its public no-arg constructor.

- Each **`LoginModule`** is initialized with a call to its initialize method. The **`Subject`** argument is guaranteed to be non-null. The signature of the initialize method is: **`public void initialize(Subject subject, CallbackHandler callbackHandler, Map sharedState, Map options)`**

- The **`login`** method is called to start the authentication process. For example, a method implementation might prompt the user for a user name and password and then verify the information against data stored in a naming service such as NIS or LDAP. Alternative implementations might interface to smart cards and biometric devices, or simply extract user information from the underlying operating system. The validation of user identity by each **`LoginModule`** is considered phase 1 of JAAS authentication. The signature of the **`login`** method is **`boolean login() throws LoginException`** . A **`LoginException`** indicates failure. A return value of true indicates that the method succeeded, whereas a return value of false indicates that the login module should be ignored.

- If the **`LoginContext`**'s overall authentication succeeds, **`commit`** is invoked on each **`LoginModule`**. If phase 1 succeeds for a **`LoginModule`**, then the commit method continues with phase 2 and associates the relevant principals, public credentials, and/or private credentials with the subject. If phase 1 fails for a **`LoginModule`**, then **`commit`** removes any previously stored authentication state, such as user names or passwords. The signature of the **`commit`** method is: **`boolean commit() throws LoginException`** . Failure to complete the commit phase is indicated by throwing a **`LoginException`**. A return of true indicates that the method succeeded, whereas a return of false indicates that the login module should be ignored.

- If the **`LoginContext`**'s overall authentication fails, then the **`abort`** method is invoked on each **`LoginModule`**. The **`abort`** method removes or destroys any authentication state created by the login or initialize methods. The signature of the **`abort`** method is **`boolean abort() throws LoginException`** . Failure to complete the **`abort`** phase is indicated by throwing a **`LoginException`**. A return of true indicates that the method succeeded, whereas a return of false indicates that the login module should be ignored.

- To remove the authentication state after a successful login, the application invokes **`logout`** on the **`LoginContext`**. This in turn results in a **`logout`** method invocation on each **`LoginModule`**. The **`logout`** method removes the principals and credentials originally associated with the subject during the **`commit`** operation. Credentials should be destroyed upon removal. The signature of the **`logout`** method is: **`boolean logout() throws LoginException`** . Failure to complete the logout process is indicated by throwing a **`LoginException`**. A return of true indicates that the method succeeded, whereas a return of false indicates that the login module should be ignored.

When a **`LoginModule`** must communicate with the user to obtain authentication information, it uses a **`CallbackHandler`** object. Applications implement the CallbackHandler interface and pass it to the **`LoginContext`**, which send the authentication information directly to the underlying login modules.

Login modules use the **CallbackHandler** both to gather input from users, such as a password or smart card PIN, and to supply information to users, such as status information. By allowing the application to specify the **CallbackHandler**, underlying **LoginModule**s remain independent from the different ways applications interact with users. For example, a **CallbackHandler**'s implementation for a GUI application might display a window to solicit user input. On the other hand, a **CallbackHandler** implementation for a non-GUI environment, such as an application server, might simply obtain credential information by using an application server API. The CallbackHandler interface has one method to implement:

```
void handle(Callback[] callbacks)
    throws java.io.IOException,
           UnsupportedCallbackException;
```

The **Callback** interface is the last authentication class we will look at. This is a tagging interface for which several default implementations are provided, including the **NameCallback** and **PasswordCallback** used in an earlier example. A **LoginModule** uses a **Callback** to request information required by the authentication mechanism. **LoginModule**s pass an array of **Callback**s directly to the **CallbackHandler.handle** method during the authentication's login phase. If a **callbackhandler** does not understand how to use a **Callback** object passed into the handle method, it throws an **UnsupportedCallbackException** to abort the login call.

Report a bug

PART II. SECURING THE PLATFORM

CHAPTER 4. THE SECURITY SUBSYSTEM

4.1. ABOUT THE SECURITY SUBSYSTEM

The security subsystem provides the infrastructure for all security functionality in JBoss EAP 6. Most configuration elements rarely need to be changed. The only configuration element which may need to be changed is whether to use *deep-copy-subject-mode*. In addition, you can configure system-wide security properties. Most of the configuration relates to *security domains*.

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.

Deep copy subject mode causes a complete copy of the data structure and all its associated data to be made, as long as they are marked cloneable. This is more thread-safe, but less efficient.

System-Wide Security Properties

You can set system-wide security properties, which are applied to **`java.security.Security`** class.

Security Domain

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`**. You can create as many security domains as you need to accommodate the needs of your applications.

Report a bug

4.2. ABOUT THE STRUCTURE OF THE SECURITY SUBSYSTEM

The security subsystem is configured in the managed domain or standalone configuration file. Most of the configuration elements can be configured using the web-based management console or the console-based management CLI. The following is the XML representing an example security subsystem.

Example 4.1. Example Security Subsystem Configuration

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
 <security-management>
  ...
 </security-management>
 <security-domains>
        <security-domain name="other" cache-type="default">
            <authentication>
                <login-module code="Remoting" flag="optional">
                    <module-option name="password-stacking"
value="useFirstPass"/>
                </login-module>
                <login-module code="RealmUsersRoles" flag="required">
                    <module-option name="usersProperties"
value="${jboss.domain.config.dir}/application-users.properties"/>
                    <module-option name="rolesProperties"
```

```
value="${jboss.domain.config.dir}/application-roles.properties"/>
                    <module-option name="realm"
value="ApplicationRealm"/>
                    <module-option name="password-stacking"
value="useFirstPass"/>
                </login-module>
            </authentication>
        </security-domain>
        <security-domain name="jboss-web-policy" cache-type="default">
            <authorization>
                <policy-module code="Delegating" flag="required"/>
            </authorization>
        </security-domain>
        <security-domain name="jboss-ejb-policy" cache-type="default">
            <authorization>
                <policy-module code="Delegating" flag="required"/>
            </authorization>
        </security-domain>
    </security-domains>
    <vault>
     ...
    </vault>
</subsystem>
```

The **`<security-management>`**, **`<subject-factory>`** and **`<security-properties>`** elements are not present in the default configuration. The **`<subject-factory>`** and **`<security-properties>`** elements have been deprecated in JBoss EAP 6.1 onwards.

Report a bug

4.3. CONFIGURING THE SECURITY SUBSYSTEM

4.3.1. Configure the Security Subsystem

You can configure the security subsystem using the Management CLI or web-based Management Console.

Each top-level element within the security subsystem contains information about a different aspect of the security configuration. Refer to Section 4.2, “About the Structure of the Security Subsystem” for an example of security subsystem configuration.

<security-management>

This section overrides 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.

Option	Description
deep-copy-subject-mode	Specifies whether to copy or link to security tokens, for additional thread safety.

Option	Description
authentication-manager-class-name	Specifies an alternate AuthenticationManager implementation class name to use.
authorization-manager-class-name	Specifies an alternate AuthorizationManager implementation class name to use.
audit-manager-class-name	Specifies an alternate AuditManager implementation class name to use.
identity-trust-manager-class-name	Specifies an alternate IdentityTrustManager implementation class name to use.
mapping-manager-class-name	Specifies the MappingManager implementation class name to use.

<subject-factory>

The subject factory controls creation of subject instances. It may use the authentication manager to verify the caller. The main use of the subject factory is for JCA components to establish a subject.It is unusual to need to modify the subject factory.

<security-domains>

A container element which holds multiple security domains. A security domain may contain information about authentication, authorization, mapping, and auditing modules, as well as JASPI authentication and JSSE configuration. Your application would specify a security domain to manage its security information.

<security-properties>

Contains names and values of properties which are set on the java.security.Security class.

Report a bug

4.3.2. Security Management

4.3.2.1. About Deep Copy Subject 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.

Deep copy subject mode causes a complete copy of the data structure and all its associated data to be made, as long as they are marked cloneable. This is more thread-safe, but less efficient.

Deep copy subject mode is configured as part of the security subsystem.

Report a bug

4.3.2.2. Enable Deep Copy Subject Mode

You can enable deep copy security mode from the web-based management console or the management CLI.

Procedure 4.1. Enable Deep Copy Security Mode from the Management Console

1. **Log into the Management Console.**
   The management console is usually available at a URL such as http://127.0.0.1:9990/. Adjust this value to suit your needs.

2. **Managed Domain: Select the appropriate profile.**
   In a managed domain, the security subsystem is configured per profile, and you can enable or disable the deep copy security mode in each, independently.

   To select a profile, click the **`Profiles`** label at the top right of the console display, and then select the profile you wish to change from the **`Profile`** selection box at the top left.

3. **Open the `Security Subsystem` configuration menu.**
   Expand the **`Security`** menu item at the right of the management console, then click the **`Security Subsystem`** link.

4. **Modify the `deep-copy-subject-mode` value.**
   Click the **`Edit`** button. Check the box beside **`Deep Copy Subjects:`** to enable deep copy subject mode.

Enable Deep Copy Subject Mode Using the Management CLI

If you prefer to use the management CLI to enable this option, use one of the following commands.

Example 4.2. Managed Domain

```
/profile=full/subsystem=security:write-attribute(name=deep-copy-subject-
mode,value=TRUE)
```

Example 4.3. Standalone Server

```
/subsystem=security:write-attribute(name=deep-copy-subject-
mode,value=TRUE)
```

Report a bug

4.3.3. Security Domains

4.3.3.1. About Security Domains

Security domains are part of the JBoss EAP 6 security subsystem. All security configuration is now managed centrally, by the domain controller of a managed domain, or by the standalone server.

A security domain consists of configurations for authentication, authorization, security mapping, and auditing. It implements *Java Authentication and Authorization Service (JAAS)* declarative security.

Authentication refers to verifying the identity of a user. In security terminology, this user is referred to as a *principal*. Although authentication and authorization are different, many of the included authentication modules also handle authorization.

An *authorization* is a security policy, by which the server determines whether an authenticated user has permission to access specific privileges or resources in the system or operation. In security terminology, this is often referred to as a role.

Security mapping refers to the ability to add, modify, or delete information from a principal, role, or attribute before passing the information to your application.

The auditing manager allows you to configure *provider modules* to control the way that security events are reported.

If you use security domains, you can remove all specific security configuration from your application itself. This allows you to change security parameters centrally. One common scenario that benefits from this type of configuration structure is the process of moving applications between testing and production environments.

Report a bug

4.3.3.2. About Picketbox

Picketbox is the foundational security framework that provides the authentication, authorization, audit and mapping capabilities to Java applications running in JBoss EAP 6. It provides the following capabilities, in a single framework with a single configuration:

- Section 5.9.1, “About Authentication”
- Section 5.11.1, “About Authorization” and access control
- Section 5.13.1, “About Security Auditing”
- Section 5.14.1, “About Security Mapping” of principals, roles, and attributes

Report a bug

CHAPTER 5. PICKETLINK IDENTITY MANAGEMENT

5.1. ABOUT SECURITY TOKEN SERVICE (STS)

The Security Token Service generates and manages the security tokens. It 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.

A security token request message specifies the following:

- Type of the request, such as Issue, Renew, and so on.
- Type of the token.
- Lifetime of the issued token.
- Information about the service provider that requested the token.
- Information used to encrypt the generated token.

The token request message is sent in the body of the SOAP message. All information related to the token request is enclosed in the **`RequestSecurityToken`** element. The sample request contains two other WS-Trust elements: **`RequestType`**, which specifies that this request is an Issue request, and **`TokenType`**, which specifies the type of the token to be issued.

The following is an example of the WS-Trust request message.

Example 5.1. WS-Trust security token request message

```
<S11:Envelope xmlns:S11=".." xmlns:wsu=".." xmlns:wst="..">
   <S11:Header>
      ...
   </S11:Header>
   <S11:Body wsu:Id="body">
      <wst:RequestSecurityToken Context="context">

<wst:TokenType>http://www.tokens.org/SpecialToken</wst:TokenType>
         <wst:RequestType>
            http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
         </wst:RequestType>
      </wst:RequestSecurityToken>
   </S11:Body>
</S11:Envelope>
```

The following is an example of a security token response.

Example 5.2. Security token response message

```
    <wst:RequestSecurityTokenResponse Context="context" xmlns:wst=".."
xmlns:wsu="..">
       <wst:TokenType>http://www.tokens.org/SpecialToken</wst:TokenType>
       <wst:RequestedSecurityToken>
```

```
            <token:SpecialToken xmlns:token="...">
               ARhjefhE2FEjneovi&@FHfeoveq3
            </token:SpecialToken>
         </wst:RequestedSecurityToken>
         <wst:Lifetime>
            <wsu:Created>...</wsu:Created>
            <wsu:Expires>...</wsu:Expires>
         </wst:Lifetime>
      </wst:RequestSecurityTokenResponse>
```

In the example for the security token response, the **`TokenType`** element specifies the type of the issued token, while the **`RequestedSecurityToken`** element contains the token itself. The format of the token depends on the type of the token. The **`Lifetime`** element specifies when the token was created and when it expires.

Security Token Request Processing

The following are the steps in which the security token requests are processed:

- A client sends a security token request to **`PicketLinkSTS`**.
- **`PicketLinkSTS`** parses the request message, generating a JAXB object model.
- **`PicketLinkSTS`** reads the configuration file and creates the **`STSConfiguration`** object, if needed. Then it obtains a reference to the **`WSTrustRequestHandler`** from the configuration and delegates the request processing to the handler instance.
- The request handler uses the **`STSConfiguration`** to set default values when needed (for example, when the request doesn't specify a token lifetime value).
- The **`WSTrustRequestHandler`** creates the **`WSTrustRequestContext`**, setting the **`JAXB`** request object and the caller principal it received from **`PicketLinkSTS`**.
- The **`WSTrustRequestHandler`** uses the **`STSConfiguration`** 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.
- The **`SecurityTokenProvider`** instance process the token request and stores the issued token in the request context.
- The **`WSTrustRequestHandler`** obtains the token from the context, encrypts it if needed, and constructs the WS-Trust response object containing the security token.
- **`PicketLinkSTS`** dictates the response generated by the request handler and returns it to the client.

Report a bug

5.2. CONFIGURE PICKETLINK STS

PicketLink STS defines several interfaces that provide extension points where implementations can be plugged via configuration and the default values for some properties can be specified via configuration. All PicketLink STS configurations must be specified in the **`picketlink-sts.xml`** file. The following are the elements that can be configured in the **picketlink-sts.xml** file.

NOTE

In the following text, a service provider refers to the Web service that requires a security token to be presented by its clients.

- **PicketLinkSTS**: This is the root element. It defines some properties that allows the STS administrator to set a the following default values:
  - **STSName**: A string representing the name of the security token service. If not specified, the default **PicketLinkSTS** value is used.
  - **TokenTimeout**: The token lifetime value in seconds. If not specified, the default value of 3600 (one hour) is used.
  - **EncryptToken**: A boolean specifying whether issued tokens are to be encrypted or not. The default value is false.
- **KeyProvider**: This element and all its sub elements are used to configure the keystore that are used by PicketLink STS to sign and encrypt tokens. Properties like the keystore location, its password, and the signing (private key) alias and password are all configured in this section.
- **RequestHandler**: This element specifies the fully qualified name of the **WSTrustRequestHandler** implementation to be used. If not specified, the default **org.picketlink.identity.federation.core.wstrust.StandardRequestHandler** is used.
- **SecurityTokenProvider**: This section specifies the **SecurityTokenProvider** implementations that must be used to handle each type of security token. In the example we have two providers - one that handles tokens of type **SpecialToken** and one that handles tokens of type **StandardToken**. The **WSTrustRequestHandler** calls the **getProviderForTokenType**(String type)method of **STSConfiguration** to obtain a reference to the appropriate **SecurityTokenProvider**.
- **TokenTimeout**: This is used by the **WSTrustRequestHandler** when no Lifetime has been specified in the WS-Trust request. It creates a Lifetime instance that has the current time as the creation time and expires after the specified number of seconds.
- **ServiceProviders**: This section specifies the token types that must be used for each service provider (the Web service that requires a security token). When a WS-Trust request does not contain the token type, the **WSTrustRequestHandler** must use the service provider endpoint to find out the type of the token that must be issued.
- **EncryptToken**: This is used by the **WSTrustRequestHandler** to decide if the issued token must be encrypted or not. If true, the public key certificate (PKC) of the service provider is used to encrypt the token.

The following is an example of PicketLink STS configuration.

Example 5.3. PicketLink STS Configuration

```
<PicketLinkSTS xmlns="urn:picketlink:identity-federation:config:1.0"
    STSName="Test STS" TokenTimeout="7200" EncryptToken="true">
```

```
         <KeyProvider
ClassName="org.picketlink.identity.federation.bindings.tomcat.KeyStoreKe
yManager">
              <Auth Key="KeyStoreURL"
Value="keystore/sts_keystore.jks"/>
                <Auth Key="KeyStorePass" Value="testpass"/>
                <Auth Key="SigningKeyAlias" Value="sts"/>
                <Auth Key="SigningKeyPass" Value="keypass"/>
                <ValidatingAlias
Key="http://services.testcorp.org/provider1" Value="service1"/>
                <ValidatingAlias
Key="http://services.testcorp.org/provider2" Value="service2"/>
         </KeyProvider>

<RequestHandler>org.picketlink.identity.federation.core.wstrust.Standard
RequestHandler</RequestHandler>
         <TokenProviders>
              <TokenProvider
ProviderClass="org.picketlink.test.identity.federation.bindings.wstrust.
SpecialTokenProvider"
                   TokenType="http://www.tokens.org/SpecialToken"/>
              <TokenProvider
ProviderClass="org.picketlink.identity.federation.api.wstrust.plugins.sa
ml.SAML20TokenProvider"
                   TokenType="http://docs.oasis-open.org/wss/oasis-wss-
saml-token-profile-1.1#SAMLV2.0"/>
         </TokenProviders>
         <ServiceProviders>
              <ServiceProvider
Endpoint="http://services.testcorp.org/provider1"
TokenType="http://www.tokens.org/SpecialToken"
                   TruststoreAlias="service1"/>
              <ServiceProvider
Endpoint="http://services.testcorp.org/provider2"
TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0"
                   TruststoreAlias="service2"/>
         </ServiceProviders>
    </PicketLinkSTS>
```

Report a bug

5.3. ABOUT PICKETLINK STS LOGIN MODULES

A PicketLink Login Module 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. PicketLink Login Modules support non-PicketLink STS implementations through standard WS-Trust calls.

Types of STS Login Modules

The following are the different types of STS Login Modules.

STSIssuingLoginModule

- Calls the configured STS and requests for a security token. Upon successfully receiving the **`RequestedSecurityToken`**, it marks the authentication as successful.
- A call to STS typically requires authentication. This Login Module uses credentials from one of the following sources:
  - Its properties file, if the **`useOptionsCredentials`** module option is set to **`true`**.
  - Previous login module credentials if the **`password-stacking`**module option is set to **`useFirstPass`**.
  - From the configured **`CallbackHandler`** by supplying a Name and Password Callback.
- Upon successful authentication, the **`SamlCredential`** is inserted in the Subject's public credentials if one with the same Assertion is not found to be already present there.

STSValidatingLoginModule

- Calls the configured STS and validates an available security token.
- A call to STS typically requires authentication. This Login Module uses credentials from one of the following sources:
  - Its properties file, if the **`useOptionsCredentials`** module option is set to **`true`**.
  - Previous login module credentials if the **`password-stacking`** module option is set to **`useFirstPass`**.
  - From the configured **`CallbackHandler`** by supplying a Name and Password Callback.
- Upon successful authentication, the SamlCredential is inserted in the Subject's public credentials if one with the same Assertion is not found to be already present there.

SAML2STSLoginModule

- This Login Module supplies a **`ObjectCallback`** to the configured **`CallbackHandler`** and expects a **`SamlCredential`** object back. The Assertion is validated against the configured STS.
- If a user ID and SAML token are shared, this Login Module bypasses validation When stacked on top of another Login Module that is successfully authenticated.
- Upon successful authentication, the **`SamlCredential`** is inspected for a **`NameID`** and a multi-valued role attribute that is respectively set as the ID and roles of the user.

SAML2LoginModule

- This login module is used in conjunction with other components for SAML authentication and performs no authentication itself.
- The **`SPRedirectFormAuthenticator`** uses this login module in PicketLink's implementation of the SAML v2 HTTP Redirect Profile.
- The Tomcat authenticator valve performs authentication through redirecting to the identity provider and getting a SAML assertion.

- This login module is used to pass the user ID and roles to the JBoss security framework to be populated in the JAAS subject.

Report a bug

5.4. CONFIGURE STSISSUINGLOGINMODULE

The **`STSIssuingLoginModule`** uses a user name and password to authenticate the user against an STS by retrieving a token.

Example 5.4. Configure STSIssuingLoginModule

```
<application-policy name="saml-issue-token">
    <authentication>
        <login-module

code="org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLog
inModule" flag="required">          <module-option
name="configFile">./picketlink-sts-client.properties</module-option>
          <module-option
name="endpointURI">http://security_saml/endpoint</module-option>
        </login-module>
    </authentication>
    <mapping>
        <mapping-module

code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STS
PrincipalMappingProvider"
            type="principal" />
        <mapping-module

code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STS
GroupMappingProvider"
            type="role" />
    </mapping>
</application-policy>
```

Most configurations can switch to the configuration sited in the above example by:

- changing their declared security-domain
- specifying a Principal mapping provider
- specifying a RoleGroup mapping provider

The specified Principal mapping provider and the RoleGroup mapping provider results in an authenticated Subject being populated that enables coarse-grained and role-based authorization. After authentication, the Security Token is available and may be used to invoke other services by Single Sign-On.

Report a bug

5.5. CONFIGURE STSVALIDATINGLOGINMODULE

The STSValidatingLoginModule uses a TokenCallback to ask the configured CallbackHandler an STS by retrieving a token.

Example 5.5. Configure STSValidatingLoginModule

```
<application-policy name="saml-validate-token">
    <authentication>
        <login-module

code="org.picketlink.identity.federation.core.wstrust.auth.STSValidating
LoginModule" flag="required">
            <module-option name="configFile">./picketlink-sts-
client.properties</module-option>
            <module-option
name="endpointURI">http://security_saml/endpoint</module-option>
        </login-module>
    </authentication>
    <mapping>
        <mapping-module

code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STS
PrincipalMappingProvider"
            type="principal" />
        <mapping-module

code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STS
GroupMappingProvider"
            type="role" />
    </mapping>
</application-policy>
```

The configuration cited in the example enables Single Sign-On for your applications and services. A token once issued, either by directly contacting the STS or through a token-issuing login module, can be used to authenticate against multiple applications and services by employing the setup provided in the example. Providing a Principal mapping provider and a RoleGroup mapping provider result in an authenticated Subject being populated that enables coarse-grained and role-based authorization. After authentication, the Security Token is available and can be used to invoke other services by Single Sign-On.

Report a bug

5.6. SAML WEB BROWSER BASED SSO

5.6.1. About SAML Web Browser Based SSO

PicketLink in JBoss EAP provides a platform to implement federated identity based services. This includes centralized identity services and Single Sign-On (SSO) for applications.

The SAML profile has support for both the HTTP/POST and the HTTP/Redirect bindings with centralized identity services to enable web SSO for your applications. The architecture for the SAML v2 based Web SSO follows the hub and spoke architecture of identity management. In this architecture an identity provider (IDP) acts as the central source (hub) for identity and role information to all the applications (Service Providers). The spokes are the service providers (SP).

Report a bug

5.6.2. Setup SAML v2 based Web SSO using HTTP/Redirect Binding

To setup SAML v2 based SSO using HTTP/Redirect Binding you have to configure the following:

- Identity Provider: The Identity Provider is the authoritative entity responsible for authenticating an end user and asserting the identity for that user in a trusted fashion to trusted partners.
- Service Provider: The Service Provider relies on the 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.

Report a bug

5.6.3. Configure Identity Provider

The Identity Provider (IDP) is a JBoss EAP server instance.

Procedure 5.1. Configure Identity Provider (IDP)

1. **Configure the web application security for the IDP**
   Configure a web application as the Identity provider.

   NOTE

   The use of FORM based web application security is recommended as it gives you the ability to customize the login page.

   The following is an example of the **`web.xml`** configuration

   Example 5.6. web.xml Configuration for IDP

```
<display-name>IDP</display-name>
<description>IDP</description>
<!-- Define a security constraint that gives unlimited access to
images -->
<security-constraint>
  <web-resource-collection>
  <web-resource-name>Images</web-resource-name>
  <url-pattern>/images/*</url-pattern>
</web-resource-collection>
</security-constraint>
<!-- Define a Security Constraint on this Application -->
<security-constraint>
  <web-resource-collection>
  <web-resource-name>IDP</web-resource-name>
  <url-pattern>/*</url-pattern>
</web-resource-collection>
  <auth-constraint>
  <role-name>manager</role-name>
</auth-constraint>
</security-constraint>
<!-- Define the Login Configuration for this Application -->
```

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>IDP Application</realm-name>
  <form-login-config>
    <form-login-page>/jsp/login.jsp</form-login-page>
    <form-error-page>/jsp/loginerror.jsp</form-error-page>
  </form-login-config>
</login-config>
<!-- Security roles referenced by this web application -->
<security-role>
 <description>
  The role that is required to log in to the IDP Application
 </description>
 <role-name>manager</role-name>
</security-role>
</web-app>
```

2. **Configure the IDP Valves**
   Create a **`context.xml`** file in the WEB-INF directory of your IDP web application to configure the valves for the IDP. The following is an example of **`context.xml`** file.

Example 5.7. context.xml File Configuration for IDP Valves

```
<context>
  <Valve
className="org.picketlink.identity.federation.bindings.tomcat.idp.
IDPWebBrowserSSOValve"/>
</context>
```

3. **Configure the PicketLink Configuration File (picketlink.xml)**
   Configure **`picketlink.xml`** in the WEB-INF directory of your IDP web application. In this configuration file you provide the URL that gets added as the issuer in the outgoing SAML2 assertions to the service providers and the IDP. The following is an example of **`picketlink.xml`** configuration.

Example 5.8. picketlink-idfed.xml Configuration

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
        <PicketLinkIDP xmlns="urn:picketlink:identity-
federation:config:2.1">

<IdentityURL>http://localhost:8080/idp/</IdentityURL>
        </PicketLinkIDP>
        <Handlers xmlns="urn:picketlink:identity-
federation:handler:config:2.1">
                <Handler

class="org.picketlink.identity.federation.web.handlers.saml2.SAML2
IssuerTrustHandler" />
                <Handler

class="org.picketlink.identity.federation.web.handlers.saml2.SAML2
```

```
LogOutHandler" />
                <Handler

class="org.picketlink.identity.federation.web.handlers.saml2.SAML2
AuthenticationHandler" />
                <Handler

class="org.picketlink.identity.federation.web.handlers.saml2.Roles
GenerationHandler" />
        </Handlers>
</PicketLink>
```

Report a bug

5.6.4. Configure Service Provider

The Service Provider (SP) can be a JBoss EAP server instance.

Procedure 5.2. Configure Service Provider (SP)

1. **Configure the Web Application Security For the SP**
   The web application to be configured as a SP should have FORM based security enabled in its web.xml file.

Example 5.9. web.xml Configuration for SP

```
<display-name>IDP</display-name>
<description>IDP</description>
<!-- Define a security constraint that gives unlimited access to
images -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Images</web-resource-name>
    <url-pattern>/images/*</url-pattern>
  </web-resource-collection>
</security-constraint>
<!-- Define a Security Constraint on this Application -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>IDP</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>
<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>IDP Application</realm-name>
  <form-login-config>
    <form-login-page>/jsp/login.jsp</form-login-page>
    <form-error-page>/jsp/loginerror.jsp</form-error-page>
  </form-login-config>
```

```
</login-config>
<!-- Security roles referenced by this web application -->
<security-role>
    <description>
   The role that is required to log in to the IDP Application
  </description>
  <role-name>manager</role-name>
</security-role>
</web-app>
```

2. **Configure the SP Valve**
   To configure the valve for the SP, create a **`context.xml`** in the WEB-INF directory of your SP web application.

   Example 5.10. context.xml File Configuration for IDP Valves

   ```
   <Context>
     <Valve
   className="org.jboss.identity.federation.bindings.tomcat.sp.SPRedi
   rectSignatureFormAuthenticator" />
   </Context>
   ```

3. **Configure the PicketLink Federation configuration file (picketlink-idfed.xml)**
   Configure **`picketlink-idfed.xml`** in WEB-INF of your IDP web application. In this configuration file you provide the URL that gets added as the issuer in the outgoing SAML2 assertions to the Service Providers and the IDP. The following is an example of **`picketlink-idfed.xml`** configuration.

   Example 5.11. picketlink-idfed.xml Configuration

   ```
   <PicketLinkIDP xmlns="urn:picketlink:identity-
   federation:config:1.0" >
   <IdentityURL>http://localhost:8080/idp/</IdentityURL>
   </PicketLinkIDP
   ```

4. **Configure the PicketLink Federation Handlers file (`picketlink-handlers.xml`)**
   Configure **`picketlink-handlers.xml`** in WEB-INF of your SP web application.

   Example 5.12. Configure picketlink-handlers.xml

   ```
   <Handlers xmlns="urn:picketlink:identity-
   federation:handler:config:1.0">
   <Handler
   class="org.picketlink.identity.federation.web.handlers.saml2.SAML2
   LogOutHandler"/>
   <Handler
   class="org.picketlink.identity.federation.web.handlers.saml2.SAML2
   AuthenticationHandler"/>
   </Handlers>
   ```

NOTE

Retain the order in which the handlers are listed.

Report a bug

5.6.5. Setup SAML v2 based Web SSO using HTTP/POST Binding

HTTP/POST binding is the recommended binding for obtaining the web browser based SSO.

Procedure 5.3. Setup SAML v2 based Web SSO using HTTP/POST Binding

1. **Configure the Identity Provider (IDP).**
   The steps to configure IDP for HTTP/POST Binding are same as that of the HTTP/Redirect Binding. For more information on configuring the IDP, see Section 5.6.2, "Setup SAML v2 based Web SSO using HTTP/Redirect Binding"

2. **Configure the Service Provider (SP)**

   NOTE

   The steps to configure SP for HTTP/POST Binding are the same as that of the HTTP/Redirect Binding, except for a variation in the **`context.xml`** file.

   The following is an example of the **`context.xml`** file for IDP valves.

   Example 5.13. context.xml File Configuration for IDP Valves

```
<Context>
  <Valve
className="org.picketlink.identity.federation.bindings.tomcat.sp.S
PPostFormAuthenticator"
/>
</Context>
```

   For more information on configuring the SP, see Section 5.6.4, "Configure Service Provider"

Report a bug

5.7. CONFIGURE SAML GLOBAL LOGOUT PROFILE

A Global Logout initiated at one service provider logs out the user from the Identity Provider (IDP) and all the service providers.

NOTE

For a Global Logout to function appropriately ensure that you have only up to five Service Providers per Identity Provider.

Procedure 5.4. Configure Global Logout

1. **Configure picketlink-handlers.xml**
   Add the **`SAML2LogOutHandler`** in the picketlink-handlers.xml.

2. **Configure Service Provider web page**
   Append **`GLO=true`** to the link at the end of your web page of the service provider.

   Example 5.14. Link to Global Logout

   ```
   <a href="?GLO=true">Click to Globally LogOut</a>
   ```

Report a bug

5.8. KERBEROS AND SPNEGO INTEGRATION

5.8.1. About Kerberos and SPNEGO Integration

Kerberos is an authentication method that is designed for open network computing environments. It works on the basis of a ticket and authenticator to establish the identity of both the user and the server. It helps the two nodes communicating over a non secure environment to establish their identity to each other in a secured manner.

SPNEGO is an authentication method used by a client application to authenticate itself to the server. This technology is used when the client application and the server trying to communicate with each other are not sure of the authentication protocol the other supports. SPNEGO determines the common GSSAPI mechanisms between the client application and the server and then dispatches all further security operations to it.

Kerberos and SPNEGO Integration

In a typical setup, the user logs into a desktop which is governed by the Active Directory domain. The user then uses the web browser, either Firebox or Internet Explorer, to access a web application that uses JBoss Negotiation hosted on the JBoss EAP. The web browser transfers the desktop sign on information 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.

Report a bug

5.8.2. Desktop SSO using SPNEGO

To configure the desktop SSO using SPNEGO configure the following:

- Security Domain
- System Properties
- Web Application

Procedure 5.5. Configure Desktop SSO using SPNEGO

1. **Configure Security Domain**

Configure the security domains to represent the identity of the server and to secure the web application.

Example 5.15. Security Domain Configuration

```
<security-domains>

    <security-domain name="host" cache-type="default">

      <authentication>

        <login-module code="Kerberos" flag="required">

          <module-option name="storeKey" value="true"/>

          <module-option name="useKeyTab" value="true"/>

          <module-option name="principal"
value="host/testserver@MY_REALM"/>

          <module-option name="keyTab"
value="/home/username/service.keytab"/>

          <module-option name="doNotPrompt" value="true"/>

          <module-option name="debug" value="false"/>

        </login-module>

       </authentication>

     </security-domain>



     <security-domain name="SPNEGO" cache-type="default">

       <authentication>

         <login-module code="SPNEGO"  flag="requisite">

           <module-option name="password-stacking"
value="useFirstPass"/>

           <module-option name="serverSecurityDomain"
value="host"/>

         </login-module>


         <!-- Login Module For Roles Search -->

       </security-domain>
```

2. **Setup the System Properties**
   If required, the system properties can be set in the domain model.

Example 5.16. Configure System Properties

```
<system-properties>

      <property name="java.security.krb5.kdc"
value="mykdc.mydomain"/>

      <property name="java.security.krb5.realm" value="MY_REALM"/>

    </system-properties>
```

3. **Configure Web Application**
   It is not possible to override the authenticators, but it is possible to add the **`NegotiationAuthenticator`** as a valve to your jboss-web.xml descriptor to configure the web application.

NOTE

The valve requires the **`security-constraint`** and **`login-config`** to be defined in the web.xml file as this is used to decide which resources are secured. However, the chosen **`auth-method`** is overridden by this authenticator.

Example 5.17. Configure Web Application

```
 <!DOCTYPE jboss-web PUBLIC
  "-//JBoss//DTD Web Application 2.4//EN"
  "http://www.jboss.org/j2ee/dtd/jboss-web_4_0.dtd">

  <jboss-web>

    <security-domain>java:/jaas/SPNEGO</security-domain>

    <valve>

      <class-
name>org.jboss.security.negotiation.NegotiationAuthenticator</clas
s-name>

    </valve>

  </jboss-web>
```

The web application also requires a dependency defining in **`META-INF/MANIFEST.MF`** so that the JBoss Negotiation classes can be located.

Example 5.18. Define Dependency in `META-INF/MANIFEST.MF`

```
Manifest-Version: 1.0

Build-Jdk: 1.6.0_24

Dependencies: org.jboss.security.negotiation
```

Report a bug

5.8.3. Configure JBoss Negotiation for Microsoft Windows Domain

This section describes how to configure the accounts required for JBoss Negotiation to be used when JBoss EAP is running on a Microsoft Windows server, which is a part of the Active Directory domain.

In this section, the hostname that is used to access the server as is referred to as **`{hostname}`**, realm is referred to as **`{realm}`**, domain is referred to as **`{domain}`**, and the server hosting the JBoss EAP instance is referred to as **`{machine_name}`**.

Procedure 5.6. Configure JBoss Negotiation for Microsoft Windows Domain

1. **Clear Existing Service Principal Mappings**
   On a Microsoft Windows network some mappings are created automatically. Delete the automatically created mappings to map the identity of the server to the service principal for negotiation to take place correctly. The mapping enables the web browser on the client computer to trust the server and attempt SPNEGO. The client computer verifies with the domain controller for a mapping in the form of **`HTTP{hostname}`**.

   The following are the steps to delete the existing mappings:

   - List the mapping registered with the domain for the computer using the command, **`setspn -L {machine_name}`**.

   - Delete the existing mappings using the commands, **`setspn -D HTTP/{hostname} {machine_name}`** and **`setspn -D host/{hostname} {machine_name}`**.

2. Create a host user account.

   NOTE

   Ensure the host user name is different from the **`{machine_name}`**.

   In the rest of the section the host user name is referred to as **`{user_name}`**.

3. **Define the mapping between the `{user_name}` and `{hostname}`.**

   - Run the following command to configure the Service Principal Mapping, **`ktpass -princ HTTP/{hostname}@{realm} -pass * -mapuser {domain}\{user_name}`**.

   - Enter the password for the user name when prompted.

> **NOTE**
>
> Reset the password for the user name as it is a prerequisite for exporting the keytab.

- Verify the mapping by running the following command, **`setspn -L {user_name}`**

4. **Export the keytab of the user to the server on which EAP JBoss is installed.**
   Run the following command to export the keytab, **`ktab -k service.keytab -a HTTP/{hostname}@{realm}`**.

   > **NOTE**
   >
   > This command exports the ticket for the HTTP/{hostname} principal to the keytab **`service.keytab`**, which is used to configure the host security domain on JBoss.

5. Define the principal within the security domain as follows:

```
<module-option name="principal">HTTP/{hostname}@{realm}</module-
option>
```

Report a bug

5.9. AUTHENTICATION

5.9.1. About 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. Other common authentication mechanisms use shared keys, smart cards, or fingerprints. The outcome of a successful authentication is referred to as a principal, in terms of Java Enterprise Edition declarative security.

JBoss EAP 6 uses a pluggable system of authentication modules to provide flexibility and integration with the authentication systems you already use in your organization. Each security domain contains one or more configured authentication modules. Each module includes additional configuration parameters to customize its behavior. The easiest way to configure the authentication subsystem is within the web-based management console.

Authentication is not the same as authorization, although they are often linked. Many of the included authentication modules can also handle authorization.

Report a bug

5.9.2. Configure Authentication in a Security Domain

To configure authentication settings for a security domain, log into the management console and follow this procedure.

Procedure 5.7. Setup Authentication Settings for a Security Domain

1. **Open the security domain's detailed view.**
   Click the **`Profiles`** label at the top right of the management console. In a managed domain,

select the profile to modify from the **Profile** selection box at the top left of the Profile view. Click the **Security** menu item at the left, and click **Security Domains** from the expanded menu. Click the **View** link for the security domain you want to edit.

2. **Navigate to the Authentication subsystem configuration.**
   Click the **Authentication** label at the top of the view if it is not already selected.

   The configuration area is divided into two areas: **Login Modules** and **Details**. The login module is the basic unit of configuration. A security domain can include several login modules, each of which can include several attributes and options.

3. **Add an authentication module.**
   Click the **Add** button to add a JAAS authentication module. Fill in the details for your module. The **Code** is the class name of the module. The **Flags** controls how the module relates to other authentication modules within the same security domain.

   Explanation of the Flags

   The Java Enterprise Edition 6 specification provides the following explanation of the flags for security modules. The following list is taken from http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html#Appendix Refer to that document for more detailed information.

Flag	Details
required	The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.
requisite	LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).
sufficient	The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the LoginModule list). If it fails, authentication continues down the LoginModule list.
optional	The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

   After you have added your module, you can modify its **Code** or **Flags** by clicking the **Edit** button in the **Details** section of the screen. Be sure the **Attributes** tab is selected.

4. **Optional: Add or remove module options.**
   If you need to add options to your module, click its entry in the **Login Modules** list, and select the **Module Options** tab in the **Details** section of the page. Click the **Add** button, and provide the key and value for the option. Use the **Remove** button to remove an option.

Result

Your authentication module is added to the security domain, and is immediately available to applications which use the security domain.

The `jboss.security.security_domain` Module Option

By default, each login module defined in a security domain has the **`jboss.security.security_domain`** module option added to it automatically. This option causes problems with login modules which check to make sure that only known options are defined. The IBM Kerberos login module, **`com.ibm.security.auth.module.Krb5LoginModule`** is one of these.

You can disable the behavior of adding this module option by setting the system property to **`true`** when starting JBoss EAP 6. Add the following to your start-up parameters.

```
-Djboss.security.disable.secdomain.option=true
```

You can also set this property using the web-based Management Console. In a standalone server, you can set system properties in the **`Profile`** section of the configuration. In a managed domain, you can set system properties for each server group.

Report a bug

5.10. JAVA AUTHENTICATION SPI FOR CONTAINERS (JASPI)

5.10.1. About Java Authentication SPI for Containers (JASPI) Security

Java Application SPI for Containers (JASPI or JASPIC) is a pluggable interface for Java applications. It is defined in JSR-196 of the Java Community Process. Refer to http://www.jcp.org/en/jsr/detail?id=196 for details about the specification.

Report a bug

5.10.2. Configure Java Authentication SPI for Containers (JASPI) Security

To authenticate against a JASPI provider, add a **`<authentication-jaspi>`** element to your security domain. The configuration is similar to a standard authentication module, but login module elements are enclosed in a **`<login-module-stack>`** element. The structure of the configuration is:

Example 5.19. Structure of the `authentication-jaspi` element

```
<authentication-jaspi>
 <login-module-stack name="...">
   <login-module code="..." flag="...">
     <module-option name="..." value="..."/>
   </login-module>
 </login-module-stack>
 <auth-module code="..." login-module-stack-ref="...">
   <module-option name="..." value="..."/>
 </auth-module>
</authentication-jaspi>
```

The login module itself is configured in exactly the same way as a standard authentication module.

Because the web-based management console does not expose the configuration of JASPI authentication modules, you need to stop JBoss EAP 6 completely before adding the configuration directly to ***EAP_HOME*/domain/configuration/domain.xml** or ***EAP_HOME*/standalone/configuration/standalone.xml**.

Report a bug

5.11. AUTHORIZATION

5.11.1. About Authorization

Authorization is a mechanism for granting or denying access to a resource based on identity. It is implemented as a set of declarative security roles which can be granted to principals.

JBoss EAP 6 uses a modular system to configure authorization. Each security domain can contain one or more authorization policies. Each policy has a basic module which defines its behavior. It is configured through specific flags and attributes. The easiest way to configure the authorization subsystem is by using the web-based management console.

Authorization is different from authentication, and usually happens after authentication. Many of the authentication modules also handle authorization.

NOTE

XACML is not permitted in the Common Criteria Certified configuration.

Report a bug

5.11.2. Configure Authorization in a Security Domain

To configure authorization settings for a security domain, log into the management console and follow this procedure.

Procedure 5.8. Setup Authorization in a Security Domain

1. **Open the security domain's detailed view.**
   Click the **Profiles** label at the top right of the management console. In a managed domain, select the profile to modify from the **Profile** selection box at the top left of the Profile view. Click the **Security** menu item at the left, and click **Security Domains** from the expanded menu. Click the **View** link for the security domain you want to edit.

2. **Navigate to the Authorization subsystem configuration.**
   Click the **Authorization** label at the top of the view if it is not already selected.

   The configuration area is divided into two areas: **Policies** and **Details**. The login module is the basic unit of configuration. A security domain can include several authorization policies, each of which can include several attributes and options.

3. **Add a policy.**
   Click the **Add** button to add a JAAS authorization policy module. Fill in the details for your module. The **Code** is the class name of the module. The **Flags** controls how the module relates to other authorization policy modules within the same security domain.

Explanation of the Flags

The Java Enterprise Edition 6 specification provides the following explanation of the flags for security modules. The following list is taken from http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html#Appendix Refer to that document for more detailed information.

Flag	Details
required	The LoginModule is required to succeed. If it succeeds or fails, authorization still continues to proceed down the LoginModule list.
requisite	LoginModule is required to succeed. If it succeeds, authorization continues down the LoginModule list. If it fails, control immediately returns to the application (authorization does not proceed down the LoginModule list).
sufficient	The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authorization does not proceed down the LoginModule list). If it fails, authorization continues down the LoginModule list.
optional	The LoginModule is not required to succeed. If it succeeds or fails, authorization still continues to proceed down the LoginModule list.

After you have added your module, you can modify its **`Code`** or **`Flags`** by clicking the **`Edit`** button in the **`Details`** section of the screen. Be sure the **`Attributes`** tab is selected.

4. **Optional: Add, edit, or remove module options.**
If you need to add options to your module, click its entry in the **`Login Modules`** list, and select the **`Module Options`** tab in the **`Details`** section of the page. Click the **`Add`** button, and provide the key and value for the option. To edit an option that already exists, click the key or to change it. Use the **`Remove`** button to remove an option.

Result

Your authorization policy module is added to the security domain, and is immediately available to applications which use the security domain.

Report a bug

5.12. JAVA AUTHORIZATION CONTRACT FOR CONTAINERS (JACC)

5.12.1. About Java Authorization Contract for Containers (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, which can be found on the Java Community Process website at http://jcp.org/en/jsr/detail?id=115. It has been part of the core Java Enterprise Edition (Java EE) specification since Java EE version 1.3.

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

Report a bug

5.12.2. Configure Java Authorization Contract for Containers (JACC) Security

To configure Java Authorization Contract for Containers (JACC), you need to configure your security domain with the correct module, and then modify your **jboss-web.xml** to include the correct parameters.

Add JACC Support to the Security Domain

To add JACC support to the security domain, add the **JACC** authorization policy to the authorization stack of the security domain, with the **required** flag set. The following is an example of a security domain with JACC support. However, the security domain is configured in the Management Console or Management CLI, rather than directly in the XML.

```
<security-domain name="jacc" cache-type="default">
    <authentication>
        <login-module code="UsersRoles" flag="required">
        </login-module>
    </authentication>
    <authorization>
        <policy-module code="JACC" flag="required"/>
    </authorization>
</security-domain>
```

Configure a Web Application to use JACC

The **jboss-web.xml** is located in the **META-INF/** or **WEB-INF/** directory of your deployment, and contains overrides and additional JBoss-specific configuration for the web container. To use your JACC-enabled security domain, you need to include the **<security-domain>** element, and also set the **<use-jboss-authorization>** element to **true**. The following application is properly configured to use the JACC security domain above.

```
<jboss-web>
    <security-domain>jacc</security-domain>
    <use-jboss-authorization>true</use-jboss-authorization>
</jboss-web>
```

Configure an EJB Application to Use JACC

Configuring EJBs to use a security domain and to use JACC differs from Web Applications. For an EJB, you can declare *method permissions* on a method or group of methods, in the **ejb-jar.xml** descriptor. Within the **<ejb-jar>** element, any child **<method-permission>** elements contain information about JACC roles. Refer to the example configuration for more details. The **EJBMethodPermission** class is part of the Java Enterprise Edition 6 API, and is documented at http://docs.oracle.com/javaee/6/api/javax/security/jacc/EJBMethodPermission.html.

Example 5.20. Example JACC Method Permissions in an EJB

```
<ejb-jar>
  <method-permission>
    <description>The employee and temp-employee roles may access any
method of the EmployeeService bean </description>
    <role-name>employee</role-name>
    <role-name>temp-employee</role-name>
    <method>
      <ejb-name>EmployeeService</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
</ejb-jar>
```

You can also constrain the authentication and authorization mechanisms for an EJB by using a security domain, just as you can do for a web application. Security domains are declared in the **`jboss-ejb3.xml`** descriptor, in the **`<security>`** child element. In addition to the security domain, you can also specify the *run-as principal*, which changes the principal the EJB runs as.

Example 5.21. Example Security Domain Declaration in an EJB

```
<security>
  <ejb-name>*</ejb-name>
  <security-domain>myDomain</security-domain>
  <run-as-principal>myPrincipal</run-as-principal>
</security>
```

Report a bug

5.13. SECURITY AUDITING

5.13.1. About 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*. You can use one of the included ones, or implement your own.

Report a bug

5.13.2. Configure Security Auditing

To configure security auditing settings for a security domain, log into the management console and follow this procedure.

Procedure 5.9. Setup Security Auditing for a Security Domain

1. **Open the security domain's detailed view.**
   Click the **`Profiles`** label at the top right of the management console. In a standalone server, the tab is labeled **`Profile`**. In a managed domain, select the profile to modify from the **`Profile`** selection box at the top left of the Profile view. Click the **`Security`** menu item at the left, and

click **Security Domains** from the expanded menu. Click the **View** link for the security domain you want to edit.

2. **Navigate to the Auditing subsystem configuration.**
   Click the **Audit** label at the top of the view if it is not already selected.

   The configuration area is divided into two areas: **Provider Modules** and **Details**. The provider module is the basic unit of configuration. A security domain can include several provider modules each of which can include attributes and options.

3. **Add a provider module.**
   Click the **Add** button to add a provider module. Fill in the **Code** section with the classname of the provider module.

   After you have added your module, you can modify its **Code** by clicking the **Edit** button in the **Details** section of the screen. Be sure the **Attributes** tab is selected.

4. **Verify if your module is working**
   The goal of an audit module is to provide a way to monitor the events in the security subsystem. This monitoring can be done by means of writing to a log file, email notifications or any other measurable auditing mechanism.

   For example, JBoss EAP 6 includes the `LogAuditProvider` module by default. If enabled following the steps above, this audit module writes security notifications to a **audit.log** file in the **log** subfolder within the ***EAP_HOME*** directory.

   To verify if the steps above have worked in the context of the `LogAuditProvider`, perform an action that is likely to trigger a notification and then check the audit log file.

   For a full list of included security auditing provider modules, see here: Section A.4, “Included Security Auditing Provider Modules”

5. **Optional: Add, edit, or remove module options.**
   If you need to add options to your module, click its entry in the **Modules** list, and select the **Module Options** tab in the **Details** section of the page. Click the **Add** button, and provide the key and value for the option. To edit an option that already exists, remove it by clicking the **Remove** label, and add it again with the correct options by clicking the **Add** button.

Result

Your security auditing module is added to the security domain, and is immediately available to applications which use the security domain.

Report a bug

5.13.3. New Security Properties

New system properties have been added to the security audit functionality for JBoss EAP versions 6.2.2 and later. These new properties mitigate security concerns surrounding plain text logging of web request components, particularly in scenarios involving BASIC or FORM based authentication.

The new properties allow greater control over which components of a web request are captured in audit logs (parameters, cookies, headers or attributes). These components can also be masked using the new properties.

The new properties are:

Table 5.1. New Security Properties

Name	Description	Possible values	Behavior	Default
`org.jboss.security.web.audit`	This property controls the granularity of the security auditing of web requests.	`off`, `headers`, `cookies`, `parameters`, `attributes`	Any component (or comma-separated group of components) specified will be audited out of web requests.	`headers,parameters`
`org.jboss.security.web.audit.mask`	This property can be used to specify a list of strings to be matched against headers, parameters, cookies, and attributes of web requests. Any element matching the specified masks will be excluded from security audit logging.	Any comma separated string indicating keys of headers, parameters, cookies, and attributes.	Currently, the matching of the masks is fuzzy rather than strict. For example, a mask of `authorization` will mask both the header called *authorization* and the parameter called *custom_authorization*. A future release may introduce strict masks.	j_password,authorization

Report a bug

5.14. SECURITY MAPPING

5.14.1. About Security Mapping

Security mapping allows you to combine authentication and authorization information after the authentication or authorization happens, but before the information is passed to your application. One example of this is using an X509 certificate for authentication, and then converting the principal from the certificate to a logical name which your application can display.

You can map principals (authentication), roles (authorization), or credentials (attributes which are not principals or roles).

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.

Attribute mapping is used to convert attributes from an external system to be used by your application, and vice versa.

Report a bug

5.14.2. Configure Security Mapping in a Security Domain

To configure security mapping settings for a security domain, log into the management console and follow this procedure.

Procedure 5.10. Setup Security Mapping Settings in a Security Domain

1. **Open the security domain's detailed view.**
   Click the **`Profiles`** label at the top right of the management console. This tab is labeled **`Profile`** in a standalone server. In a managed domain, select the profile to modify from the **`Profile`** selection box at the top left of the Profile view. Click the **`Security`** menu item at the left, and click **`Security Domains`** from the expanded menu. Click the **`View`** link for the security domain you want to edit.

2. **Navigate to the Mapping subsystem configuration.**
   Click the **`Mapping`** label at the top of the view if it is not already selected.

   The configuration area is divided into two areas: **`Modules`** and **`Details`**. The mapping module is the basic unit of configuration. A security domain can include several mapping modules, each of which can include several attributes and options.

3. **Add a module.**
   Click the **`Add`** button to add a security mapping module. Fill in the details for your module. The **`Code`** is the class name of the module. The **`Type`** field refers to the type of mapping this module performs. Allowed values are principal, role, attribute or credential.

   After you have added your module, you can modify its **`Code`** or **`Type`** by clicking the **`Edit`** button in the **`Details`** section of the screen. Be sure the **`Attributes`** tab is selected.

4. **Optional: Add, edit, or remove module options.**
   If you need to add options to your module, click its entry in the **`Modules`** list, and select the **`Module Options`** tab in the **`Details`** section of the page. Click the **`Add`** button, and provide the key and value for the option. To edit an option that already exists, click the **`Remove`** label key to remove it, and add it again with the new value. Use the **`Remove`** button to remove an option.

Result

Your security mapping module is added to the security domain, and is immediately available to applications which use the security domain.

Report a bug

CHAPTER 6. JAVA SECURITY MANAGER

6.1. ABOUT THE JAVA SECURITY MANAGER

Java Security Manager

> The Java Security Manager is a class that manages the external boundary of the Java Virtual Machine (JVM) sandbox, controlling how code executing within the JVM can interact with resources outside the JVM. When the Java Security Manager is activated, the Java API checks with the security manager for approval before executing a wide range of potentially unsafe operations.

The Java Security Manager uses a security policy to determine whether a given action will be permitted or denied.

Report a bug

6.2. ABOUT JAVA SECURITY MANAGER POLICIES

Security Policy

> A set of defined permissions for different classes of code. The Java Security Manager compares actions requested by applications against the security policy. If an action is allowed by the policy, the Security Manager will permit that action to take place. If the action is not allowed by the policy, the Security Manager will deny that action. The security policy can define permissions based on the location of code, on the code's signature, or based on the subject's principals.

The Java Security Manager and the security policy used are configured using the Java Virtual Machine options **`java.security.manager`** and **`java.security.policy`**.

Basic Information

A security policy's entry consists of the following configuration elements, which are connected to the **`policytool`**:

CodeBase

The URL location (excluding the host and domain information) where the code originates from. This parameter is optional.

SignedBy

The alias used in the keystore to reference the signer whose private key was used to sign the code. This can be a single value or a comma-separated list of values. This parameter is optional. If omitted, presence or lack of a signature has no impact on the Java Security Manager.

Principals

A list of ***`principal_type`***/***`principal_name`*** pairs, which must be present within the executing thread's principal set. The Principals entry is optional. If it is omitted, it signifies that the principals of the executing thread will have no impact on the Java Security Manager.

Permissions

A permission is the access which is granted to the code. Many permissions are provided as part of the Java Enterprise Edition 6 (Java EE 6) specification. This document only covers additional permissions which are provided by JBoss EAP 6.

Report a bug

6.3. WRITE A JAVA SECURITY MANAGER POLICY

Introduction

An application called **`policytool`** is included with most JDK and JRE distributions, for the purpose of creating and editing Java Security Manager security policies. Detailed information about **`policytool`** is linked from http://docs.oracle.com/javase/6/docs/technotes/tools/.

Procedure 6.1. Setup a new Java Security Manager Policy

1. **Start `policytool`.**
   Start the **`policytool`** tool in one of the following ways.

   - **Red Hat Enterprise Linux**
     From your GUI or a command prompt, run **`/usr/bin/policytool`**.

   - **Microsoft Windows Server**
     Run **`policytool.exe`** from your Start menu or from the **`bin\`** of your Java installation. The location can vary.

2. **Create a policy.**
   To create a policy, select **`Add Policy Entry`**. Add the parameters you need, then click **`Done`**.

3. **Edit an existing policy**
   Select the policy from the list of existing policies, and select the **`Edit Policy Entry`** button. Edit the parameters as needed.

4. **Delete an existing policy.**
   Select the policy from the list of existing policies, and select the **`Remove Policy Entry`** button.

Report a bug

6.4. IBM JRE AND THE JAVA SECURITY MANAGER

IBM JRE uses a default policy provider which does not work correctly with the JBoss Enterprise Application Platform security policy. You must change the JRE configuration to use the standard policy provider, if you want to use the IBM JRE to host JBoss Enterprise Application Platform with the Java Security Manager enabled.

To configure the JRE configuration for the IBM JRE, edit the ***`JAVA_HOME/jre/lib/security/java.security`*** file, and set the **`policy.provider`** value to **`sun.security.provider.PolicyFile`**.

```
policy.provider=sun.security.provider.PolicyFile
```

Report a bug

6.5. JAVA SECURITY POLICY STATEMENTS

A policy file specifies permissions to modules and deployed applications. Permissions are applied to deployed applications via the **VFS** protocol. See the following Oracle Java SE documentation page *Default Policy Implementation and Policy File Syntax* for further information at http://docs.oracle.com/javase/7/docs/technotes/guides/security/PolicyFiles.html

The following is an example of policy statements.

```
// Grant all to the jboss-modules.jar
grant codeBase "file:${jboss.home.dir}/jboss-modules.jar" {
  permission java.security.AllPermission;
};

// Standard extensions get all permissions by default
grant codeBase "file:${{java.ext.dirs}}/*" {
 permission java.security.AllPermission;
};

// Grant read PropertyPermission for all properties to a deployed EJB
application
grant codeBase "vfs:/content/ejb-app.jar" {
  permission java.util.PropertyPermission "*", "read";
};

// Grant read FilePermission for all files to a web application
grant codeBase "vfs:/content/myapp.war/-" {
  permission java.io.FilePermission "/-", "read";
};
```

NOTE

On Microsoft Windows Server, when specifying a **`FilePermission`** statement including a file path in a string, not a codeBase URL, you must replace single backslash characters with two backslash characters. This is because when file paths are parsed, a single backslash is interpreted as an escape character.

NOTE

Two VFS issues currently exist in JBoss EAP that require a workaround:

- If you define a grant for a deployment on Microsoft Windows, instead of:

  ```
  grant codeBase "vfs:/content/..." {
  ```

  you must use:

  ```
  grant codeBase "vfs:/${user.dir}/content/..." {
  ```

- If your application contains JSPs, then the permissions granted to the deployment using a VFS URL is not sufficient, and you will have to duplicate it with file-based URL. For example, if you have permission:

  ```
  grant codeBase "vfs:/content/application.war/-" {
    permission java.util.PropertyPermission "*", "read";
  };
  ```

  then you also need to add the following:

  ```
  grant codeBase
  "file:${jboss.home.dir}/standalone/tmp/work/jboss.web/def
  ault-host/application/-" {
    permission java.util.PropertyPermission "*", "read";
  };
  ```

Module permissions are defined in **`module.xml`** (version 1.2 or higher). The following example demonstrates specifying module permissions.

```
<module xmlns="urn:jboss:module:1.2" name="org.jboss.custom.module">
 <permissions>
  <grant permission="java.io.FilePermission" name="/-" actions="read"/>
  <grant permission="java.lang.RuntimePermission" name="org.jboss.*"/>
 </permissions>

 <resources>
  <resource-root path="custom-module.jar" />
 </resources>
</module>
```

If there is no **`<permissions/>`** element, then **`AllPermission`** permission is granted to the module. If there is an empty **`<permissions/>`** element, then no permission is granted.

Report a bug

6.6. RUN JBOSS EAP 6 WITHIN THE JAVA SECURITY MANAGER

To specify a Java Security Manager policy, you need to edit the Java options passed to the domain or server instance during the bootstrap process. For this reason, you cannot pass the parameters as options to the **`domain.sh`** or **`standalone.sh`** scripts. The following procedure guides you through the steps of configuring your instance to run within a Java Security Manager policy.

Prerequisites

- Before you following this procedure, you need to write a security policy, using the **`policytool`** command which is included with your Java Development Kit (JDK). This procedure assumes that your policy is located at ***`EAP_HOME`*`/bin/server.policy`**. As an alternative, write the security policy using any text editor and manually save it as ***`EAP_HOME`*`/bin/server.policy`**
- The domain or standalone server must be completely stopped before you edit any configuration files.

Perform the following procedure for each physical host or instance in your domain, if you have domain members spread across multiple systems.

Procedure 6.2. Configure the Security Manager for JBoss EAP 6

1. **Open the configuration file.**
   Open the configuration file for editing. This file is located in one of two places, depending on whether you use a managed domain or standalone server. This is not the executable file used to start the server or domain.
   - **Managed Domain**
     - For Linux: ***`EAP_HOME`*`/bin/domain.conf`**
     - For Windows: ***`EAP_HOME`*`\bin\domain.conf.bat`**
   - **Standalone Server**
     - For Linux: ***`EAP_HOME`*`/bin/standalone.conf`**
     - For Windows: ***`EAP_HOME`*`\bin\standalone.conf.bat`**
2. **Add the Java options to the file.**
   To ensure the Java options are used, add them to the code block that begins with:

   ```
   if [ "x$JAVA_OPTS" = "x" ]; then
   ```

   You can modify the **`-Djava.security.policy`** value to specify the exact location of your security policy. It should go onto one line only, with no line break. Using **==** when setting the **`-Djava.security.policy`** property specifies that the security manager will use *only* the specified policy file. Using **=** specifies that the security manager will use the specified policy *combined with* the policy set in the **`policy.url`** section of ***`JAVA_HOME`*`/lib/security/java.security`**.

   IMPORTANT

   JBoss Enterprise Application Platform releases from 6.2.2 onwards require that the system property **`jboss.modules.policy-permissions`** is set to *true*.

   Example 6.1. domain.conf

   ```
   JAVA_OPTS="$JAVA_OPTS -Djava.security.manager -
   Djava.security.policy==$PWD/server.policy -
   Djboss.home.dir=/path/to/EAP_HOME -Djboss.modules.policy-
   ```

```
permissions=true"
```

Example 6.2. domain.conf.bat

```
set "JAVA_OPTS=%JAVA_OPTS% -Djava.security.manager -
Djava.security.policy==\path\to\server.policy -
Djboss.home.dir=\path\to\EAP_HOME -Djboss.modules.policy-
permissions=true"
```

Example 6.3. standalone.conf

```
JAVA_OPTS="$JAVA_OPTS -Djava.security.manager -
Djava.security.policy==$PWD/server.policy -
Djboss.home.dir=$JBOSS_HOME -Djboss.modules.policy-
permissions=true"
```

Example 6.4. standalone.conf.bat

```
set "JAVA_OPTS=%JAVA_OPTS% -Djava.security.manager -
Djava.security.policy==\path\to\server.policy -
Djboss.home.dir=%JBOSS_HOME% -Djboss.modules.policy-
permissions=true"
```

3. **Start the domain or server.**
   Start the domain or server as normal.

Report a bug

6.7. DEBUG SECURITY MANAGER POLICIES

You can enable debugging information to help you troubleshoot security policy-related issues. The **`java.security.debug`** option configures the level of security-related information reported. The command **`java -Djava.security.debug=help`** will produce help output with the full range of debugging options. Setting the debug level to **`all`** is useful when troubleshooting a security-related failure whose cause is completely unknown, but for general use it will produce too much information. A sensible general default is **`access:failure`**.

Procedure 6.3. Enable general debugging

- **This procedure will enable a sensible general level of security-related debug information.**
  Add the following line to the server configuration file.
  - If the JBoss EAP 6 instance is running in a managed domain, the line is added to the **`bin/domain.conf`** file for Linux or the **`bin/domain.conf.bat`** file for Windows.

- If the JBoss EAP 6 instance is running as a standalone server, the line is added to the **`bin/standalone.conf`** file for Linux, or the **`bin\standalone.conf.bat`** file for Windows.

`Linux`

```
JAVA_OPTS="$JAVA_OPTS -Djava.security.debug=access:failure"
```

`Windows`

```
JAVA_OPTS="%JAVA_OPTS% -Djava.security.debug=access:failure"
```

Result

A general level of security-related debug information has been enabled.

Report a bug

CHAPTER 7. SECURITY REALMS

7.1. ABOUT SECURITY REALMS

A *security realm* is a series of mappings between users and passwords, and users and roles. Security realms are a mechanism for adding authentication and authorization to your EJB and Web applications. JBoss EAP 6 provides two security realms by default:

- **`ManagementRealm`** stores authentication information for the Management API, which provides the functionality for the Management CLI and web-based Management Console. It provides an authentication system for managing JBoss EAP 6 itself. You could also use the **`ManagementRealm`** if your application needed to authenticate with the same business rules you use for the Management API.
- **`ApplicationRealm`** stores user, password, and role information for Web Applications and EJBs.

Each realm is stored in two files on the filesystem:

- ***`REALM`*****`-users.properties`** stores usernames and hashed passwords.
- ***`REALM`*****`-users.properties`** stores user-to-role mappings.

The properties files are stored in the **`domain/configuration/`** and **`standalone/configuration/`** directories. The files are written simultaneously by the **`add-user.sh`** or **`add-user.bat`** command. When you run the command, the first decision you make is which realm to add your new user to.

Report a bug

7.2. ADD A NEW SECURITY REALM

1. **Run the Management CLI.**
   Start the **`jboss-cli.sh`** or **`jboss-cli.bat`** command and connect to the server.

2. **Create the new security realm itself.**
   Run the following command to create a new security realm named **`MyDomainRealm`** on a domain controller or a standalone server.

   ```
   /host=master/core-service=management/security-
   realm=MyDomainRealm:add()
   ```

3. **Create the references to the properties file which will store information about the new role.**
   Run the following command to create a pointer a file named **`myfile.properties`**, which will contain the properties pertaining to the new role.

   NOTE

   The newly-created properties file is not managed by the included **`add-user.sh`** and **`add-user.bat`** scripts. It must be managed externally.

```
/host=master/core-service=management/security-
realm=MyDomainRealm/authentication=properties:add(path=myfile.proper
ties)
```

Result

Your new security realm is created. When you add users and roles to this new realm, the information will be stored in a separate file from the default security realms. You can manage this new file using your own applications or procedures.

Report a bug

7.3. ADD A USER TO A SECURITY REALM

1. **Run the `add-user.sh` or `add-user.bat` command.**
   Open a terminal and change directories to the ***`EAP_HOME`*****`/bin/`** directory. If you run Red Hat Enterprise Linux or another UNIX-like operating system, run **`add-user.sh`**. If you run Microsoft Windows Server, run **`add-user.bat`**.

2. **Choose whether to add a Management User or Application User.**
   For this procedure, type **`b`** to add an Application User.

3. **Choose the realm the user will be added to.**
   By default, the only available realm is **`ApplicationRealm`**. If you have added a custom realm, you can type its name instead.

4. **Type the username, password, and roles, when prompted.**
   Type the desired username, password, and optional roles when prompted. Verify your choice by typing **`yes`**, or type **`no`** to cancel the changes. The changes are written to each of the properties files for the security realm.

Report a bug

CHAPTER 8. ENCRYPTION

8.1. ABOUT ENCRYPTION

Encryption refers to obfuscating sensitive information by applying mathematical algorithms to it. Encryption is one of the foundations of securing your infrastructure from data breaches, system outages, and other risks.

Encryption can be applied to simple string data, such as passwords. It can also be applied to data communication streams. The HTTPS protocol, for instance, encrypts all data before transferring it from one party to another. If you connect from one server to another using the Secure Shell (SSH) protocol, all of your communication is sent in an encrypted *tunnel* .

Report a bug

8.2. ABOUT SSL ENCRYPTION

Secure Sockets Layer (SSL) encrypts network traffic between two systems. Traffic between the two systems is encrypted using a two-way key, generated during the *handshake* phase of the connection and known only by those two systems.

For secure exchange of the two-way encryption key, SSL 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 others 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, a handshake phase takes place before secure communication can begin. During the SSL 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 two-way encryption key for the SSL connection, encrypts it using the public key of the server, and sends it back to the server. The server decrypts the two-way encryption key, using its private key, and further communication between the two machines over this connection is encrypted using the two-way encryption key.

Report a bug

8.3. IMPLEMENT SSL ENCRYPTION FOR THE JBOSS EAP 6 WEB SERVER

Introduction

Many web applications require a SSL-encrypted connection between clients and server, also known as a **`HTTPS`** connection. You can use this procedure to enable **`HTTPS`** on your server or server group.

Prerequisites

- You need a set of SSL encryption keys and a SSL encryption certificate. You may purchase these from a certificate-signing authority, or you can generate them yourself using command-line utilities. To generate encryption keys using Red Hat Enterprise Linux utilities, refer to Section 8.4, “Generate a SSL Encryption Key and Certificate”.

- You need to know the following details about your specific environment and set-up:
  - The full directory name and path to your certificate files
  - The encryption password for your encryption keys.
- You need to run the Management CLI and connect it to your domain controller or standalone server.

NOTE

This procedure uses commands appropriate for a JBoss EAP 6 configuration that uses a managed domain. If you use a standalone server, modify Management CLI commands by removing the **`/profile=default`** from the beginning of any Management CLI commands.

Procedure 8.1. Configure the JBoss Web Server to use HTTPS

1. **Add a new HTTPS connector.**
   Execute the following Management CLI command, changing the profile as appropriate. This creates a new secure connector, called **`HTTPS`**, which uses the **`https`** scheme, the **`https`** socket binding (which defaults to **`8443`**), and is set to be secure.

   Example 8.1. Management CLI Command

   ```
   /profile=default/subsystem=web/connector=HTTPS/:add(socket-
   binding=https,scheme=https,protocol=HTTP/1.1,secure=true)
   ```

2. **Configure the SSL encryption certificate and keys.**
   Execute the following CLI commands to configure your SSL certificate, substituting your own values for the example ones. This example assumes that the keystore is copied to the server configuration directory, which is ***`EAP_HOME`*`/domain/configuration/`** for a managed domain.

   Example 8.2. Management CLI Command

   ```
   /profile=default/subsystem=web/connector=HTTPS/ssl=configuration:a
   dd(name=https,certificate-key-
   file="${jboss.server.config.dir}/keystore.jks",password=SECRET,
   key-alias=KEY_ALIAS)
   ```

   For a full listing of parameters you can set for the SSL properties of the connector, refer to Section 8.5, “SSL Connector Reference”.

3. **Deploy an application.**
   Deploy an application to a server group which uses the profile you have configured. If you use a standalone server, deploy an application to your server. HTTP requests to it use the new SSL-encrypted connection.

Report a bug

8.4. GENERATE A SSL ENCRYPTION KEY AND CERTIFICATE

To use a SSL-encrypted HTTP connection (HTTPS), as well as other types of SSL-encrypted communication, you need a signed encryption certificate. You can purchase a certificate from a Certificate Authority (CA), or you can use a self-signed certificate. Self-signed certificates are not considered trustworthy by many third parties, but are appropriate for internal testing purposes.

This procedure enables you to create a self-signed certificate using utilities which are available on Red Hat Enterprise Linux.

Prerequisites

- You need the **`keytool`** utility, which is provided by any Java Development Kit implementation. OpenJDK on Red Hat Enterprise Linux installs this command to **`/usr/bin/keytool`**.
- Understand the syntax and parameters of the **`keytool`** command. This procedure uses extremely generic instructions, because further discussion of the specifics of SSL certificates or the **`keytool`** command are out of scope for this documentation.

Procedure 8.2. Generate a SSL Encryption Key and Certificate

1. **Generate a keystore with public and private keys.**
   Run the following command to generate a keystore named **`server.keystore`** with the alias **`jboss`** in your current directory.

   ```
   keytool -genkeypair -alias jboss -keyalg RSA -keystore
   server.keystore -storepass mykeystorepass --dname
   "CN=jsmith,OU=Engineering,O=mycompany.com,L=Raleigh,S=NC,C=US"
   ```

   The following table describes the parameters used in the keytool command:

Parameter	Description
-genkeypair	The **`keytool`** command to generate a key pair containing a public and private key.
-alias	The alias for the keystore. This value is arbitrary, but the alias **`jboss`** is the default used by the JBoss Web server.
-keyalg	The key pair generation algorithm. In this case it is **RSA**.
-keystore	The name and location of the keystore file. The default location is the current directory. The name you choose is arbitrary. In this case, the file will be named **`server.keystore`**.

Parameter	Description
-storepass	This password is used to authenticate to the keystore so that the key can be read. The password must be at least 6 characters long and must be provided when the keystore is accessed. In this case, we used **`mykeystorepass`**. If you omit this parameter, you will be prompted to enter it when you execute the command.
-keypass	This is the password for the actual key. **NOTE** Due to an implementation limitation this must be the same as the store password.
--dname	A quoted string describing the distinguished name for the key, for example: "CN=jsmith,OU=Engineering,O=mycompany.co m,L=Raleigh,C=US". This string is a concatenation of the following components: ◦ **`CN`** - The common name or host name. If the hostname is "jsmith.mycompany.com", the **`CN`** is "jsmith". ◦ **`OU`** - The organizational unit, for example "Engineering" ◦ **`O`** - The organization name, for example "mycompany.com". ◦ **`L`** - The locality, for example "Raleigh" or "London" ◦ **`S`** - The state or province, for example "NC". This parameter is optional. ◦ **`C`** - The 2 letter country code, for example "US" or "UK",

When you execute the above command, you are prompted for the following information:

- If you did not use the ***-storepass*** parameter on the command line, you are asked to enter the keystore password. Re-enter the new password at the next prompt.

- If you did not use the **`-keypass`** parameter on the command line, you are asked to enter the key password. Press **`Enter`** to set this to the same value as the keystore password.

When the command completes, the file **`server.keystore`** now contains the single key with the alias **`jboss`**.

2. **Verify the key.**
   Verify that the key works properly by using the following command.

   ```
   keytool -list -keystore server.keystore
   ```

   You are prompted for the keystore password. The contents of the keystore are displayed (in this case, a single key called **`jboss`**). Notice the type of the **`jboss`** key, which is **`keyEntry`**. This indicates that the keystore contains both a public and private entry for this key.

3. **Generate a certificate signing request.**
   Run the following command to generate a certificate signing request using the public key from the keystore you created in step 1.

   ```
   keytool -certreq -keyalg RSA -alias jboss -keystore server.keystore
   -file certreq.csr
   ```

   You are prompted for the password in order to authenticate to the keystore. The **`keytool`** command then creates a new certificate signing request called **`certreq.csr`** in the current working directory.

4. **Test the newly generated certificate signing request.**
   Test the contents of the certificate by using the following command.

   ```
   openssl req -in certreq.csr -noout -text
   ```

   The certificate details are shown.

5. **Optional: Submit your certificate signing request to a Certificate Authority (CA).**
   A Certificate Authority (CA) can authenticate your certificate so that it is considered trustworthy by third-party clients. The CA supplies you with a signed certificate, and optionally with one or more intermediate certificates.

6. **Optional: Export a self-signed certificate from the keystore.**
   If you only need it for testing or internal purposes, you can use a self-signed certificate. You can export one from the keystore you created in step 1 as follows:

   ```
   keytool -export -alias jboss -keystore server.keystore -file
   server.crt
   ```

   You are prompted for the password in order to authenticate to the keystore. A self-signed certificate, named **`server.crt`**, is created in the current working directory.

7. **Import the signed certificate, along with any intermediate certificates.**
   Import each certificate, in the order that you are instructed by the CA. For each certificate to import, replace **`intermediate.ca`** or **`server.crt`** with the actual file name. If your certificates are not provided as separate files, create a separate file for each certificate, and paste its contents into the file.

   NOTE

   Your signed certificate and certificate keys are valuable assets. Be cautious with how you transport them between servers.

```
keytool -import -keystore server.keystore -alias intermediateCA -
file intermediate.ca
```

```
keytool -import -alias jboss -keystore server.keystore -file
server.crt
```

8. **Test that your certificates imported successfully.**
   Run the following command, and enter the keystore password when prompted. The contents of your keystore are displayed, and the certificates are part of the list.

   ```
   keytool -list -keystore server.keystore
   ```

Result

Your signed certificate is now included in your keystore and is ready to be used to encrypt SSL connections, including HTTPS web server communications.

Report a bug

8.5. SSL CONNECTOR REFERENCE

JBoss Web connectors may include the following SSL configuration attributes. The CLI commands provided are designed for a managed domain using profile **`default`**. Change the profile name to the one you wish to configure, for a managed domain, or omit the **`/profile=default`** portion of the command, for a standalone server.

Table 8.1. SSL Connector Attributes

Attribute	Description	CLI Command
name	The display name of the SSL connector.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=name,value=https)`
verify-client	Set to **`true`** to require a valid certificate chain from the client before accepting a connection. Set to **`want`** if you want the SSL stack to request a client Certificate, but not fail if one is not presented. Set to **`false`** (the default) to not require a certificate chain unless the client requests a resource protected by a security constraint that uses **`CLIENT-CERT`** authentication.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=verify-client,value=want)`

Attribute	Description	CLI Command
verify-depth	The maximum number of intermediate certificate issuers checked before deciding that the clients do not have a valid certificate. The default value is **`10`**.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=verify-depth,value=10)`
certificate-key-file	The full file path and file name of the keystore file where the signed server certificate is stored. With JSSE encryption, this certificate file will be the only one, while OpenSSL uses several files. The default value is the **`.keystore`** file in the home directory of the user running JBoss EAP 6. If your **`keystoreType`** does not use a file, set the parameter to an empty string.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=certificate-key-file,value=../domain/configuration/server.keystore)`
certificate-file	If you use OpenSSL encryption, set the value of this parameter to the path to the file containing the server certificate.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=certificate-file,value=server.crt)`
password	The password for both the trustore and keystore. In the following example, replace *PASSWORD* with your own password.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=password,value=`*`PASSWORD`*`)`
protocol	The version of the SSL protocol to use. Supported values include **`SSLv2`**, **`SSLv3`**, **`TLSv1`**, **`SSLv2+SSLv3`**, and **`ALL`**. The default is **`ALL`**.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=protocol,value=ALL)`

Attribute	Description	CLI Command
cipher-suite	A comma-separated list of the encryption ciphers which are allowed. The JVM default for JSSE contains weak ciphers which should not be used. The example only lists two possible ciphers, but real-world examples will likely use more.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=cipher-suite, value="TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA")`
key-alias	The alias used to for the server certificate in the keystore. In the following example, replace *KEY_ALIAS* with your certificate's alias.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=key-alias,value=`*`KEY_ALIAS`*`)`
truststore-type	The type of the truststore. Various types of keystores are available, including **`PKCS12`** and Java's standard **`JKS`**.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=truststore-type,value=jks)`
keystore-type	The type of the keystore, Various types of keystores are available, including **`PKCS12`** and Java's standard **`JKS`**.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=keystore-type,value=jks)`
ca-certificate-file	The file containing the CA certificates. This is the **`truststoreFile`**, in the case of JSSE, and uses the same password as the keystore. The **`ca-certificate-file`** file is used to validate client certificates.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=certificate-file,value=ca.crt)`

Attribute	Description	CLI Command
ca-certificate-password	The Certificate password for the **`ca-certificate-file`**. In the following example, replace the *MASKED_PASSWORD* with your own masked password.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=ca-certificate-password,value=`*`MASKED_PASSWORD`*`)`
ca-revocation-url	A file or URL which contains the revocation list. It refers to the **`crlFile`** for JSSE or the **`SSLCARevocationFile`** for SSL.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=ca-revocation-url,value=ca.crl)`
session-cache-size	The size of the SSLSession cache. This attribute applies only to JSSE connectors. The default is **`0`**, which specifies an unlimited cache size.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=session-cache-size,value=100)`
session-timeout	The number of seconds before a cached SSLSession expires. This attribute applies only to JSSE connectors. The default is **`86400`** seconds, which is 24 hours.	`/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=session-timeout,value=43200)`

Report a bug

8.6. FIPS 140-2 COMPLIANT ENCRYPTION

8.6.1. About FIPS 140-2 Compliance

The Federal Information Processing Standard 140-2 (FIPS 140-2) is a US government computer security standard for the accreditation of cryptographic software modules. FIPS 140-2 compliance is often a requirement of software systems used by government agencies and private sector business.

JBoss EAP 6 uses external modules encryption and can be configured to use a FIPS 140-2 compliant cryptography module.

Report a bug

8.6.2. FIPS 140-2 Compliant Passwords

A FIPS compliant password must have the following characteristics:

1. Must be at least seven (7) characters in length.
2. Must include characters from at least three (3) of the following character classes:
   - ASCII digits,
   - lowercase ASCII,
   - uppercase ASCII,
   - non-alphanumeric ASCII, and
   - non-ASCII.

If the first character of the password is an uppercase ASCII letter, then it is not counted as an uppercase ASCII letter for restriction 2.

If the last character of the password is an ASCII digit, then it does not count as an ASCII digit for restriction 2.

Report a bug

8.6.3. Enable FIPS 140-2 Cryptography for SSL on Red Hat Enterprise Linux 6

This task describes how to configure the web container (JBoss Web) of JBoss EAP 6 to FIPS 140-2 compliant cryptography for SSL. This task only covers the steps to do this on Red Hat Enterprise Linux 6.

This task uses the Mozilla NSS library in FIPS mode for this feature.

Prerequisites

- Red Hat Enterprise Linux 6 must already be configured to be FIPS 140-2 compliant. Refer to https://access.redhat.com/knowledge/solutions/137833.

Procedure 8.3. Enable FIPS 140-2 Compliant Cryptography for SSL

1. **Create the database**
   Create the NSS database in a directory own by the **`jboss`** user.

   ```
   $ mkdir -p  /usr/share/jboss-as/nssdb
   $ chown jboss /usr/share/jboss-as/nssdb
   $ modutil -create -dbdir /usr/share/jboss-as/nssdb
   ```

2. **Create NSS configuration file**
   Create a new text file with the name **`nss_pkcsll_fips.cfg`** in the **`/usr/share/jboss-as`** directory with the following contents:

```
name = nss-fips
nssLibraryDirectory=/usr/lib64
nssSecmodDirectory=/usr/share/jboss-as/nssdb
nssModule = fips
```

The NSS configuration file must specify:

- a name,
- the directory where the NSS library is located, and
- the directory where the NSS database was created as per step 1.

If you are not running a 64bit version of Red Hat Enterprise Linux 6 then set **`nssLibraryDirectory`** to **`/usr/lib`** instead of **`/usr/lib64`**.

3. **Enable SunPKCS11 provider**
   Edit the **`java.security`** configuration file for your JRE (**`$JAVA_HOME/jre/lib/security/java.security`**) and add the following line:

   ```
   security.provider.1=sun.security.pkcs11.SunPKCS11  /usr/share/jboss-
   as/nss_pkcsll_fips.cfg
   ```

   Note that the configuration file specified in this line is the file created in step 2.

   Any other **`security.provider.`*`X`*** lines in this file must have the value of their *X* increased by one to ensure that this provider is given priority.

4. **Enable FIPS mode for the NSS library**
   Run the **`modutil`** command as shown to enable FIPS mode:

   ```
   modutil -fips true -dbdir /usr/share/jboss-as/nssdb
   ```

   Note that the directory specified here is the one created in step 1.

   You may get a security library error at this point requiring you to regenerate the library signatures for some of the NSS shared objects.

5. **Change the password on the FIPS token**
   Set the password on the FIPS token using the following command. Note that the name of the token must be **`NSS FIPS 140-2 Certificate DB`**.

   ```
   modutil -changepw "NSS FIPS 140-2 Certificate DB" -dbdir
   /usr/share/jboss-as/nssdb
   ```

   The password used for the FIPS token must be a FIPS compliant password.

6. **Create certificate using NSS tools**
   Enter the following command to create a certificate using the NSS tools.

   ```
   certutil -S -k rsa -n jbossweb  -t "u,u,u" -x -s "CN=localhost,
   OU=MYOU, O=MYORG, L=MYCITY, ST=MYSTATE, C=MY" -d /usr/share/jboss-
   as/nssdb
   ```

7. **Configure the HTTPS connector to use the PKCS11 keystore**
   Add a HTTPS connector using the following command in the JBoss CLI Tool:

   ```
   /subsystem=web/connector=https/:add(socket-
   binding=https,scheme=https,protocol=HTTP/1.1,secure=true)
   ```

   Then add the SSL configuration with the following command, replacing PASSWORD with the FIPS compliant password from step 5.

   ```
   /subsystem=web/connector=https/ssl=configuration:add(name=https,pass
   word=PASSWORD,keystore-type=PCKS11,
   cipher-
   suite="SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_S
   HA,
   TLS_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
   TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,
   TLS_DHE_DSS_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
   TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_128_CBC
   _SHA,
   TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_3DES_EDE_CB
   C_SHA,
   TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_CB
   C_SHA,
   TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
   ,
   TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SH
   A,
   TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SH
   A,
   TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_anon_WITH_AES_128_CBC_S
   HA,
   TLS_ECDH_anon_WITH_AES_256_CBC_SHA")
   ```

8. **Verify**
   Verify that the JVM can read the private key from the PKCS11 keystore by running the following command:

   ```
   keytool -list -storetype pkcs11
   ```

Example 8.3. XML configuration for HTTPS connector using FIPS 140-2 compliance

```
<connector name="https" protocol="HTTP/1.1" scheme="https" socket-
binding="https" secure="true">
  <ssl name="https" password="****"
      cipher-
suite="SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
         TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
         TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,

TLS_DHE_DSS_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,

TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
,
```

```
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SH
A,

TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SH
A,

TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,

TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,

TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,

TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_anon_WITH_AES_128_CBC_SHA,
         TLS_ECDH_anon_WITH_AES_256_CBC_SHA"
      keystore-type="PKCS11"/>
</connector>
```

Note that the **`cipher-suite`** attribute has linebreaks inserted to make it easier to read.

Report a bug

CHAPTER 9. NETWORK SECURITY

9.1. SECURE THE MANAGEMENT INTERFACES

Summary

In a test environment, it is typical to run JBoss EAP 6 with no security layer on the management interfaces, comprised of the Management Console, Management CLI, and any other API implementation. This allows for rapid development and configuration changes.

In addition, a silent authentication mode is present by default, allowing a local client on the host machine to connect to the Management CLI without requiring a username or password. This behavior is a convenience for local users and Management CLI scripts, but it can be disabled if required. The procedure is described in the topic Section 10.5, “Remove Silent Authentication from the Default Security Realm”.

When you begin testing and preparing your environment to move to production, it is vitally important to secure the management interfaces by at least the following methods:

- Section 9.2, “Specify Which Network Interface JBoss EAP 6 Uses”
- Section 9.3, “Configure Network Firewalls to Work with JBoss EAP 6”

Report a bug

9.2. SPECIFY WHICH NETWORK INTERFACE JBOSS EAP 6 USES

Overview

Isolating services so that they are accessible only to the clients who need them increases the security of your network. JBoss EAP 6 includes two interfaces in its default configuration, both of which bind to the IP address **`127.0.0.1`**, or **`localhost`**, by default. One of the interfaces is called **`management`**, and is used by the Management Console, CLI, and API. The other is called **`public`**, and is used to deploy applications. These interfaces are not special or significant, but are provided as a starting point.

The **`management`** interface uses ports 9990 and 9999 by default, and the**`public`** interface uses port 8080, or port 8443 if you use HTTPS.

You can change the IP address of the management interface, public interface, or both.

WARNING

If you expose the management interfaces to other network interfaces which are accessible from remote hosts, be aware of the security implications. Most of the time, it is not advisable to provide remote access to the management interfaces.

1. **Stop JBoss EAP 6.**
   Stop JBoss EAP 6 by sending an interrupt in the appropriate way for your operating system. If you are running JBoss EAP 6 as a foreground application, the typical way to do this is to press **`Ctrl+C`**.

2. **Restart JBoss EAP 6, specifying the bind address.**
   Use the `-b` command-line switch to start JBoss EAP 6 on a specific interface.

   Example 9.1. Specify the public interface.

   ```
   EAP_HOME/bin/domain.sh -b 10.1.1.1
   ```

   Example 9.2. Specify the management interface.

   ```
   EAP_HOME/bin/domain.sh -bmanagement=10.1.1.1
   ```

   Example 9.3. Specify different addresses for each interface.

   ```
   EAP_HOME/bin/domain.sh -bmanagement=127.0.0.1 -b 10.1.1.1
   ```

   Example 9.4. Bind the public interface to all network interfaces.

   ```
   EAP_HOME/bin/domain.sh -b 0.0.0.0
   ```

It is possible to edit your XML configuration file directly, to change the default bind addresses. However, if you do this, you will no longer be able to use the `-b`command-line switch to specify an IP address at run-time, so this is not recommended. If you do decide to do this, be sure to stop JBoss EAP 6 completely before editing the XML file.

Report a bug

9.3. CONFIGURE NETWORK FIREWALLS TO WORK WITH JBOSS EAP 6

Summary

Most production environments use firewalls as part of an overall network security strategy. If you need multiple server instances to communicate with each other or with external services such as web servers or databases, your firewall must take this into account. A well-managed firewall only opens the ports which are necessary for operation, and limits access to the ports to specific IP addresses, subnets, and network protocols.

A full discussion of firewalls is out of the scope of this documentation.

Prerequisites

- Determine the ports you need to open.
- An understanding of your firewall software is required. This procedure uses the **`system-config-firewall`** command in Red Hat Enterprise Linux 6. Microsoft Windows Server includes a built-in firewall, and several third-party firewall solutions are available for each platform.

Assumptions

This procedure configures a firewall in an environment with the following assumptions:

- The operating system is Red Hat Enterprise Linux 6.
- JBoss EAP 6 runs on host **`10.1.1.2`**. Optionally, the server has its own firewall.
- The network firewall server runs on host **`10.1.1.1`** on interface **`eth0`**, and has an external interface **`eth1`**.
- You want traffic on port 5445 (a port used by JMS) forwarded to JBoss EAP 6. No other traffic should be allowed through the network firewall.

Procedure 9.1. Manage Network Firewalls and JBoss EAP 6 to work together

1. **Log into the Management Console.**
   Log into the Management Console. By default, it runs on http://localhost:9990/console/.

2. **Determine the socket bindings used by the socket binding group.**
   Click the **`Profiles`** label at the top right of the Management Console. At the left side of the screen, a series of menus is shown. The bottom menu heading is **`General Configuration`**. Click the **`Socket Binding`** item below this heading. The **`Socket Binding Declarations`** screen appears. Initially, the **`standard-sockets`** group is shown. You can choose a different group by selecting it from the combo box on the right-hand side.

   NOTE

   If you use a standalone server, it has only one socket binding group.

   The list of socket names and ports is shown, eight values per page. You can go through the pages by using the arrow navigation below the table.

3. **Determine the ports you need to open.**
   Depending on the function of the particular port and the requirements of your environment, some ports may need to be opened on your firewall.

4. **Configure your firewall to forward traffic to JBoss EAP 6.**
   Perform these steps to configure your network firewall to allow traffic on the desired port.

   a. Log into your firewall machine and access a command prompt, as the root user.

   b. Issue the command **`system-config-firewall`** to launch the firewall configuration utility. A GUI or command-line utility launches, depending on the way you are logged into the firewall system. This task makes the assumption that you are logged in via SSH and using the command-line interface.

   c. Use the **TAB** key on your keyboard to navigate to the **`Customize`** button, and press the **ENTER** key. The **`Trusted Services`** screen appears.

   d. Do not change any values, but use the **TAB** key to navigate to the **`Forward`** button, and press **ENTER** to advanced to the next screen. The **`Other Ports`** screen appears.

   e. Use the **TAB** key to navigate to the **<Add>** button, and press **ENTER**. The **`Port and Protocol`** screen appears.

f. Enter **`5445`** in the **`Port / Port Range`** field, then use the **`TAB`** key to move to the **`Protocol`** field, and enter **`tcp`**. Use the **`TAB`** key to navigate to the **`OK`** button, and press **`ENTER`**.

g. Use the **`TAB`** key to navigate to the **`Forward`** button until you reach the **`Port Forwarding`** screen.

h. Use the **`TAB`** key to navigate to the **`<Add>`** button, and press the **`ENTER`** key.

i. Fill in the following values to set up port forwarding for port 5445.

   - Source interface: eth1
   - Protocol: tcp
   - Port / Port Range: 5445
   - Destination IP address: 10.1.1.2
   - Port / Port Range: 5445

   Use the **`TAB`** key to navigate to the **`OK`** button, and press **`ENTER`**.

j. Use the **`TAB`** key to navigate to the **`Close`** button, and press **`ENTER`**.

k. Use the **`TAB`** key to navigate to the **`OK`** button, and press **`ENTER`**. To apply the changes, read the warning and click **`Yes`**.

5. **Configure a firewall on your JBoss EAP 6 host.**
   Some organizations choose to configure a firewall on the JBoss EAP 6 server itself, and close all ports that are not necessary for its operation. See Section 9.4, "Network Ports Used By JBoss EAP 6" and determine which ports to open, then close the rest. The default configuration of Red Hat Enterprise Linux 6 closes all ports except 22 (used for Secure Shell (SSH) and 5353 (used for multicast DNS). While you are configuring ports, ensure you have physical access to your server so that you do not inadvertently lock yourself out.

Result

Your firewall is configured to forward traffic to your internal JBoss EAP 6 server in the way you specified in your firewall configuration. If you chose to enable a firewall on your server, all ports are closed except the ones needed to run your applications.

Report a bug

9.4. NETWORK PORTS USED BY JBOSS EAP 6

The ports used by the JBoss EAP 6 default configuration depend on several factors:

- Whether your server groups use one of the default socket binding groups, or a custom group.
- The requirements of your individual deployments.

NOTE

A numerical port offset can be configured, to alleviate port conflicts when you run multiple servers on the same physical server. If your server uses a numerical port offset, add the offset to the default port number for its server group's socket binding group. For instance, if the HTTP port of the socket binding group is 8080, and your server uses a port offset of 100, its HTTP port is 8180.

Unless otherwise stated, the ports use the TCP protocol.

The default socket binding groups

- `full-ha-sockets`
- `full-sockets`
- `ha-sockets`
- `standard-sockets`

Table 9.1. Reference of the default socket bindings

Name	Port	Mulicast Port	Description	full-ha-sockets	full-sockets	ha-socket	standard-socket
`ajp`	8009		Apache JServ Protocol. Used for HTTP clustering and load balancing.	Yes	Yes	Yes	Yes
`http`	8080		The default port for deployed web applications.	Yes	Yes	Yes	Yes
`https`	8443		SSL-encrypted connection between deployed web applications and clients.	Yes	Yes	Yes	Yes
`jacorb`	3528		CORBA services for JTS transactions and other ORB-dependent services.	Yes	Yes	No	No
`jacorb-ssl`	3529		SSL-encrypted CORBA services.	Yes	Yes	No	No

Name	Port	Mulicast Port	Description	full-ha-sockets	full-sockets	ha-socket	standard-socket
`jgroups-diagnostics`		7500	Multicast. Used for peer discovery in HA clusters. Not configurable using the Management Interfaces.	Yes	No	Yes	No
`jgroups-mping`		45700	Multicast. Used to discover initial membership in a HA cluster.	Yes	No	Yes	No
`jgroups-tcp`	7600		Unicast peer discovery in HA clusters using TCP.	Yes	No	Yes	No
`jgroups-tcp-fd`	57600		Used for HA failure detection over TCP.	Yes	No	Yes	No
`jgroups-udp`	55200	45688	Unicast peer discovery in HA clusters using UDP.	Yes	No	Yes	No
`jgroups-udp-fd`	54200		Used for HA failure detection over UDP.	Yes	No	Yes	No
`messaging`	5445		JMS service.	Yes	Yes	No	No
`messaging-group`			Referenced by HornetQ JMS broadcast and discovery groups.	Yes	Yes	No	No
`messaging-throughput`	5455		Used by JMS Remoting.	Yes	Yes	No	No
`mod_cluster`		23364	Multicast port for communication between JBoss EAP 6 and the HTTP load balancer.	Yes	No	Yes	No

Name	Port	Mulicast Port	Description	full-ha-sockets	full-sockets	ha-socket	standard-socket
`osgi-http`	8090		Used by internal components which use the OSGi subsystem. Not configurable using the Management Interfaces.	Yes	Yes	Yes	Yes
`remoting`	4447		Used for remote EJB invocation.	Yes	Yes	Yes	Yes
`txn-recovery-environment`	4712		The JTA transaction recovery manager.	Yes	Yes	Yes	Yes
`txn-status-manager`	4713		The JTA / JTS transation manager.	Yes	Yes	Yes	Yes

Management Ports

In addition to the socket binding groups, each host controller opens two more ports for management purposes:

- 9990 - The Web Management Console port
- 9999 - The port used by the Management Console and Management API

Report a bug

CHAPTER 10. MANAGEMENT INTERFACE SECURITY

10.1. SECURE THE MANAGEMENT INTERFACES

Summary

In a test environment, it is typical to run JBoss EAP 6 with no security layer on the management interfaces, comprised of the Management Console, Management CLI, and any other API implementation. This allows for rapid development and configuration changes.

In addition, a silent authentication mode is present by default, allowing a local client on the host machine to connect to the Management CLI without requiring a username or password. This behavior is a convenience for local users and Management CLI scripts, but it can be disabled if required. The procedure is described in the topic Section 10.5, "Remove Silent Authentication from the Default Security Realm".

When you begin testing and preparing your environment to move to production, it is vitally important to secure the management interfaces by at least the following methods:

- Section 9.2, "Specify Which Network Interface JBoss EAP 6 Uses"
- Section 9.3, "Configure Network Firewalls to Work with JBoss EAP 6"

Report a bug

10.2. DEFAULT USER SECURITY CONFIGURATION

Introduction

All management interfaces in JBoss EAP 6 are secured by default. This security takes two different forms:

- Local interfaces are secured by a SASL contract between local clients and the server they connect to. This security mechanism is based on the client's ability to access the local filesystem. This is because access to the local filesystem would allow the client to add a user or otherwise change the configuration to thwart other security mechanisms. This adheres to the principle that if physical access to the filesystem is achieved, other security mechanisms are superfluous. The mechanism happens in four steps:

  NOTE

  HTTP access is considered to be remote, even if you connect to the localhost using HTTP.

  1. The client sends a message to the server which includes a request to authenticate with the local SASL mechanism.
  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.

- Remote clients, including local HTTP clients, use realm-based security. The default realm with the permissions to configure the JBoss EAP 6 remotely using the management interfaces is **`ManagementRealm`**. A script is provided which allows you to add users to this realm (or realms you create). For more information on adding users, see the *Getting Started* chapter of the *JBoss EAP 6 Installation Guide*. For each user, the username, a hashed password, and the realm are stored in a file.

  Managed domain

  ***`EAP_HOME`*`/domain/configuration/mgmt-users.properties`**

  Standalone server

  ***`EAP_HOME`*`/standalone/configuration/mgmt-users.properties`**

  Even though the contents of the **`mgmt-users.properties`** are masked, the file must still be treated as a sensitive file. It is recommended that it be set to the file mode of **`600`**, which gives no access other than read and write access by the file owner.

Report a bug

10.3. OVERVIEW OF ADVANCED MANAGEMENT INTERFACE CONFIGURATION

The Management interface configuration in the ***`EAP_HOME`*`/domain/configuration/host.xml`** or ***`EAP_HOME`*`/standalone/configuration/standalone.xml`** controls which network interfaces the host controller process binds to, which types of management interfaces are available at all, and which type of authentication system is used to authenticate users on each interface. This topic discusses how to configure the Management Interfaces to suit your environment.

The Management subsystem consists of a **`<management>`** element that includes several configurable attributes, and the following three configurable child elements. The security realms and outbound connections are each first defined, and then applied to the management interfaces as attributes.

- <security-realms>
- <outbound-connections>
- <management-interfaces>
- <audit-log>

NOTE

Refer to the *Management Interface Audit Logging* section of the *Administration and Configuration Guide* for more information on audit logging.

Security Realms

The security realm is responsible for the authentication and authorization of users allowed to administer JBoss EAP 6 via the Management API, Management CLI, or web-based Management Console.

Two different file-based security realms are included in a default installation: **`ManagementRealm`** and **`ApplicationRealm`**. Each of these security realms uses a **`-users.properties`** file to store users and hashed passwords, and a **`-roles.properties`** to store mappings between users and roles.

Support is also included for an LDAP-enabled security realm.

NOTE

Security realms can also be used for your own applications. The security realms discussed here are specific to the management interfaces.

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.

For more information on how to configure LDAP authentication see Section 10.12.2, "Use LDAP to Authenticate to the Management Interfaces".

Management Interfaces

A management interface includes properties about how connect to and configure JBoss EAP. Such information includes the named network interface, port, security realm, and other configurable information about the interface. Two interfaces are included in a default installation:

- **`http-interface`** is the configuration for the web-based Management Console.
- **`native-interface`** is the configuration for the command-line Management CLI and the REST-like Management API.

Each of the three main configurable elements of the host management subsystem are interrelated. A security realm refers to an outbound connection, and a management interface refers to a security realm.

Associated information can be found in Section 10.1, "Secure the Management Interfaces".

Report a bug

10.4. DISABLE THE HTTP MANAGEMENT INTERFACE

In a managed domain, you only need access to the HTTP interface on the domain controller, rather than on domain member servers. In addition, on a production server, you may decide to disable the web-based Management Console altogether.

NOTE

Other clients, such as JBoss Operations Network, also operate using the HTTP interface. If you want to use these services, and simply disable the Management Console itself, you can set the **`console-enabled`** attribute of the HTTP interface to **`false`**, instead of disabling the interface completely.

```
/host=master/core-service=management/management-interface=http-
interface/:write-attribute(name=console-enabled,value=false)
```

To disable access to the HTTP interface, which also disables access to the web-based Management Console, you can delete the HTTP interface altogether.

The following JBoss CLI command allows you to read the current contents of your HTTP interface, in case you decide to add it again.

Example 10.1. Read the Configuration of the HTTP Interface

```
/host=master/core-service=management/management-interface=http-
interface/:read-resource(recursive=true,proxies=false,include-
runtime=false,include-defaults=true)
{
    "outcome" => "success",
    "result" => {
        "console-enabled" => true,
        "interface" => "management",
        "port" => expression "${jboss.management.http.port:9990}",
        "secure-port" => undefined,
        "security-realm" => "ManagementRealm"
    }
}
```

To remove the HTTP interface, issue the following command:

Example 10.2. Remove the HTTP Interface

```
/host=master/core-service=management/management-interface=http-
interface/:remove
```

To re-enable access, issue the following commands to re-create the HTTP Interface with the default values.

Example 10.3. Re-Create the HTTP Interface

```
/host=master/core-service=management/management-interface=http-
interface:add(console-
enabled=true,interface=management,port="${jboss.management.http.port:999
0}",security-realm=ManagementRealm)
```

Report a bug

10.5. REMOVE SILENT AUTHENTICATION FROM THE DEFAULT SECURITY REALM

Summary

The default installation of JBoss EAP 6 contains a method of silent authentication for a local Management CLI user. This 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 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 convenience of silent authentication for local users can be disabled where greater security control is required. This can be achieved by removing the **`local`** element within the **`security-realm`** section of the configuration file. This applies to both the **`standalone.xml`** for a Standalone Server instance, or **`host.xml`** for a Managed Domain. You should only consider the removal of the **`local`** element if you understand the impact that it might have on your particular server configuration.

The preferred method of removing silent authentication is by use of the Management CLI, which directly removes the **`local`** element visible in the following example.

Example 10.4. Example of the `local` element in the `security-realm`

```
<security-realms>
    <security-realm name="ManagementRealm">
        <authentication>
            <local default-user="$local"/>
            <properties path="mgmt-users.properties" relative-
to="jboss.server.config.dir"/>
        </authentication>
    </security-realm>
    <security-realm name="ApplicationRealm">
        <authentication>
            <local default-user="$local" allowed-users="*"/>
            <properties path="application-users.properties" relative-
to="jboss.server.config.dir"/>
        </authentication>
        <authorization>
            <properties path="application-roles.properties" relative-
to="jboss.server.config.dir"/>
        </authorization>
    </security-realm>
</security-realms>
```

Prerequisites

- Start the JBoss EAP 6 instance.
- Launch the Management CLI.

Procedure 10.1. Remove Silent Authentication from the Default Security Realm

- **Remove silent authentication with the Management CLI**
  Remove the **`local`** element from the Management Realm and Application Realm as required.

  a. Remove the **`local`** element from the Management Realm.

     - **For Standalone Servers**

       ```
       /core-service=management/security-
       realm=ManagementRealm/authentication=local:remove
       ```

     - **For Managed Domains**

```
/host=HOST_NAME/core-service=management/security-
realm=ManagementRealm/authentication=local:remove
```

b. Remove the **`local`** element from the Application Realm.

- **For Standalone Servers**

```
/core-service=management/security-
realm=ApplicationRealm/authentication=local:remove
```

- **For Managed Domains**

```
/host=HOST_NAME/core-service=management/security-
realm=ApplicationRealm/authentication=local:remove
```

Result

The silent authentication mode is removed from the **`ManagementRealm`** and the **`ApplicationRealm`**.

Report a bug

10.6. DISABLE REMOTE ACCESS TO THE JMX SUBSYSTEM

Remote JMX connectivity allows you to trigger JDK and application management operations. In order to secure an installation, disable this function. You can do this either by removing the remote connection configuration, or removing the JMX subsystem entirely. The JBoss CLI commands reference the default profile in a managed domain configuration. To modify a different profile, modify the **`/profile=default`** part of the command. For a standalone server, remove that portion of the command completely.

NOTE

In a managed domain the remoting connector is removed from the JMX subsystem by default. This command is provided for your information, in case you add it during development.

Example 10.5. Remove the Remote Connector from the JMX Subsystem

```
/profile=default/subsystem=jmx/remoting-connector=jmx/:remove
```

Example 10.6. Remove the JMX Subsystem

Run this command for each profile you use, if you use a managed domain.

```
/profile=default/subsystem=jmx/:remove
```

Report a bug

10.7. CONFIGURE SECURITY REALMS FOR THE MANAGEMENT INTERFACES

The Management Interfaces use security realms to control authentication and access to the configuration mechanisms of JBoss EAP 6. This topic shows you how to read and configure security realms. These commands use the Management CLI.

Read a Security Realm's Configuration

This example shows the default configuration for the **`ManagementRealm`** security realm. It uses a file called **`mgmt-users.properties`** to store its configuration information.

Example 10.7. Default ManagementRealm

```
 /host=master/core-service=management/security-
realm=ManagementRealm/:read-
resource(recursive=true,proxies=false,include-runtime=false,include-
defaults=true)
{
    "outcome" => "success",
    "result" => {
        "authorization" => undefined,
        "server-identity" => undefined,
        "authentication" => {"properties" => {
            "path" => "mgmt-users.properties",
            "plain-text" => false,
            "relative-to" => "jboss.domain.config.dir"
        }}
    }
}
```

Write a Security Realm

The following commands create a new security realm called **`TestRealm`** and set the directory for the relevant properties file.

Example 10.8. Writing a Security Realm

```
/host=master/core-service=management/security-realm=TestRealm/:add
/host=master/core-service=management/security-
realm=TestRealm/authentication=properties/:add(path=TestUsers.properties
, relative-to=jboss.domain.config.dir)
```

Apply a Security Realm to the Management Interface

After adding a security realm, supply it as a reference to the Management Interface.

Example 10.9. Add a Security Realm to a Management Interface

```
/host=master/core-service=management/management-interface=http-
interface/:write-attribute(security-realm=TestRealm)
```

Report a bug

10.8. CONFIGURE THE MANAGEMENT CONSOLE FOR HTTPS IN STANDALONE MODE

Procedure 10.2.

1. Ensure the management console binds to **`HTTPS`** for its interface by adding the **`management-https`** configuration and removing the **`management-http`** configuration.

   This can be done by editing the **`standalone.xml`** file (which is not recommended) or by using the following CLI interface commands:

   ```
   /core-service=management/management-interface=http-interface:write-
   attribute(name=secure-socket-binding, value=management-https)
   ```

   ```
   /core-service=management/management-interface=http-
   interface:undefine-attribute(name=socket-binding)
   ```

2. **Optional:**
   If you are using a custom ***`socket-binding`*** group, ensure the ***`management-https`*** binding is defined (it is present by default, bound to port **`9443`**).

   ```
    <socket-binding-group name="standard-sockets" default-
   interface="public" port-offset="${jboss.socket.binding.port-
   offset:0}">
           <socket-binding name="management-native"
   interface="management" port="${jboss.management.native.port:9999}"/>
           <socket-binding name="management-http"
   interface="management" port="${jboss.management.http.port:9990}"/>
           <socket-binding name="management-https"
   interface="management" port="${jboss.management.https.port:9443}"/>
   ```

3. Generate a keypair as discussed in Section 8.4, “Generate a SSL Encryption Key and Certificate”.

4. Add a ***`server-identities`*** element to the **`security-realm`** section of the **`standalone.xml`** configuration file of your installation.

   Within this element you define the protocol, the keystore path, the keystore password and alias for the key pair.

   Execute the following CLI command, substituting your own values for the example ones. This example assumes that the keystore is copied to the server configuration directory, which is ***`EAP_HOME`*`/standalone/configuration/`** for a standalone server.

   ```
   /core-service=management/security-realm=ManagementRealm/server-
   identity=ssl:add(keystore-path=server.keystore,keystore-relative-
   to=jboss.server.config.dir, keystore-password=SECRET,
   alias=KEY_ALIAS)
   ```

5. Restart your standalone server.

Report a bug

10.9. CONFIGURE THE MANAGEMENT CONSOLE FOR HTTPS IN DOMAIN MODE

Procedure 10.3.

1. Generate a keypair as discussed in Section 8.4, “Generate a SSL Encryption Key and Certificate”.

2. Add a **`server-identities`** element to the **`security-realm`** block in your installations **`host.xml.`**.

   Within this element you define the protocol, the keystore path, the keystore password and alias for the key pair.

   Execute the following CLI command, substituting your own values for the example ones. This example assumes that the keystore is copied to the server configuration directory, which is EAP_HOME/domain/configuration/ for a managed domain.

   ```
   /host=master/core-service=management/security-
   realm=ManagementRealm/server-identity=ssl:add(protocol=TLSv1,
   keystore-path=server.keystore,keystore-relative-
   to=jboss.domain.config.dir, keystore-password=SECRET,
   alias=KEY_ALIAS)
   ```

3. Change the socket element within the ***`management-interface`*** section by adding ***`secure-port`*** and removing port configuration.

   Use the following commands:

   ```
   /host=master/core-service=management/management-interface=http-
   interface:write-attribute(name=secure-port,value=9443)
   ```

   ```
   /host=master/core-service=management/management-interface=http-
   interface:undefine-attribute(name=port)
   ```

4. Restart your domain.

Report a bug

10.10. USING 2-WAY SSL FOR THE MANAGEMENT INTERFACE AND THE CLI

In this topic the following conventions are used:

HOST1

The JBoss server hostname. For example; **`jboss.redhat.com`**

HOST2

A suitable name for the client. For example: **`myclient`**. Note this is not necessarily an actual hostname.

CA_HOST1

The DN (distinguished name) to use for the *HOST1* certificate. For example **`cn=jboss,dc=redhat,dc=com`**.

CA_HOST2

The DN (distinguished name) to use for the *HOST2* certificate. For example **`cn=myclient,dc=redhat,dc=com`**.

Procedure 10.4.

1. Generate the stores:

```
keytool -genkeypair -alias HOST1_alias -keyalg RSA -keysize 1024 -
validity 365 -keystore host1.keystore.jks -dname "CA_HOST1" -keypass
secret -storepass secret
```

```
keytool -genkeypair -alias HOST2_alias -keyalg RSA -keysize 1024 -
validity 365 -keystore host2.keystore.jks -dname "CA_HOST2" -keypass
secret -storepass secret
```

2. Export the certificates:

```
keytool -exportcert  -keystore HOST1.keystore.jks -alias HOST1_alias
-keypass secret -storepass secret -file HOST1.cer
```

```
keytool -exportcert  -keystore HOST2.keystore.jks -alias HOST2_alias
-keypass secret -storepass secret -file HOST2.cer
```

3. Import the certificates into the opposing trust stores:

```
keytool -importcert -keystore HOST1.truststore.jks -storepass secret
-alias HOST2_alias -trustcacerts -file HOST2.cer
```

```
keytool -importcert -keystore HOST2.truststore.jks -storepass secret
-alias HOST1_alias -trustcacerts -file HOST1.cer
```

4. Define a CertificateRealm in the configuration for your installation (**`host.xml`** or **`standalone.xml`**) and point the interface to it:

   This can be done by manually editing the configuration file (not recommended) or by using the following commands:

```
/core-service=management/security-realm=CertificateRealm:add()
```

```
/core-service=management/security-realm=CertificateRealm:add/server-
identity=ssl:add(keystore-path=/path/to/HOST1.keystore.jks,keystore-
password=secret, alias=HOST1_alias)
```

-

  ```
  /core-service=management/security-
  realm=CertificateRealm/authentication=truststore:add(keystore-
  path=/path/to/HOST1.truststore.jks,keystore-password=secret)
  ```

5. Edit the ***JBOSS_HOME*/bin/jboss-cli.xml** and add the SSL configuration (using the appropriate values for the variables):

   ```
   <ssl>
     <alias>$HOST2alias</alias>
     <key-store>/path/to/HOST2.keystore.jks</key-store>
     <key-store-password>secret</key-store-password>
     <trust-store>/path/to/HOST2.truststore.jks</trust-store>
     <trust-store-password>secret</trust-store-password>
     <modify-trust-store>true</modify-trust-store>
   </ssl>
   ```

Report a bug

10.11. PASSWORD VAULTS FOR SENSITIVE STRINGS

10.11.1. About Securing Sensitive Strings in Clear-Text Files

Web applications and other deployments often include clear-text files, such as XML deployment descriptors, which include sensitive information such as passwords and other sensitive strings. JBoss EAP 6 includes a password vault mechanism which enables you to encrypt sensitive strings and store them in an encrypted keystore. The vault mechanism manages decrypting the strings for use with security domains, security realms, or other verification systems. This provides an extra layer of security. The mechanism relies upon tools that are included in all supported Java Development Kit (JDK) implementations.

WARNING

Problems have been encountered when using the Vault security feature with JBoss EAP 6. It has been found that the vault.keystore generated the Sun/Oracle keytool is not a valid keystore when used with an IBM JDK. This is due to the fact that the JCEKS keystore implementations differ across Java vendors.

The issue presents when a keystore generated by Oracle Java is used in a JBoss EAP instance on an IBM Java installation. In these cases the server will not start and throws the following exception:

```
java.io.IOException:
com.sun.crypto.provider.SealedObjectForKeyProtector
```

At the moment, the only workaround is to avoid attempting to use a keystore generated with an Oracle keytool in an environment using an IBM Java implementation.

Report a bug

10.11.2. Create a Java Keystore to Store Sensitive Strings

Prerequisites

- The **`keytool`** command must be available to use. It is provided by the Java Runtime Environment (JRE). Locate the path for the file. In Red Hat Enterprise Linux, it is installed to **`/usr/bin/keytool`**.

Procedure 10.5. Setup a Java Keystore

1. **Create a directory to store your keystore and other encrypted information.**
   Create a directory to hold your keystore and other important information. The rest of this procedure assumes that the directory is **`/home/`*`USER`*`/vault/`**.

2. **Determine the parameters to use with `keytool`.**
   Determine the following parameters:

   alias

   The alias is a unique identifier for the vault or other data stored in the keystore. The alias in the example command at the end of this procedure is **`vault`**. Aliases are case-insensitive.

   keyalg

   The algorithm to use for encryption. The example in this procedure uses **`RSA`**. Use the documentation for your JRE and operating system to see which other choices may be available to you.

   keysize

   The size of an encryption key impacts how difficult it is to decrypt through brute force. The example in this procedure uses **`2048`**. For information on appropriate values, see the documentation distributed with the **`keytool`**.

   keystore

   The keystore is a database which holds encrypted information and the information about how to decrypt it. If you do not specify a keystore, the default keystore to use is a file called **`.keystore`** in your home directory. The first time you add data to a keystore, it is created. The example in this procedure uses the **`vault.keystore`** keystore.

   The **`keytool`** command has many other options. Refer to the documentation for your JRE or your operating system for more details.

3. **Determine the answers to questions the `keystore` command will ask.**
   The **`keystore`** needs the following information in order to populate the keystore entry:

   Keystore password

   When you create a keystore, you must set a password. In order to work with the keystore in the future, you need to provide the password. Create a strong password that you will remember. The keystore is only as secure as its password and the security of the file system and operating system where it resides.

Key password (optional)

In addition to the keystore password, you can specify a password for each key it holds. In order to use such a key, the password needs to be given each time it is used. Usually, this facility is not used.

First name (given name) and last name (surname)

This, and the rest of the information in the list, helps to uniquely identify the key and place it into a hierarchy of other keys. It does not necessarily need to be a name at all, but it should be two words, and must be unique to the key. The example in this procedure uses **`Accounting Administrator`**. In directory terms, this becomes the *common name* of the certificate.

Organizational unit

This is a single word that identifies who uses the certificate. It may be the application or the business unit. The example in this procedure uses **`AccountingServices`**. Typically, all keystores used by a group or application use the same organizational unit.

Organization

This is usually a single-word representation of your organization's name. This typically remains the same across all certificates used by an organization. This example uses **`MyOrganization`**.

City or municipality

Your city.

State or province

Your state or province, or the equivalent for your locality.

Country

The two-letter code for your country.

All of this information together will create a hierarchy for your keystores and certificates, ensuring that they use a consistent naming structure but are unique.

4. **Run the `keytool` command, supplying the information that you gathered.**

Example 10.10. Example input and output of `keystore` command

```
$ keytool -genseckey -alias vault -storetype jceks -keyalg AES -
keysize 128 -storepass vault22 -keypass vault22 -keystore
/home/USER/vault/vault.keystore
Enter keystore password: vault22
Re-enter new password:vault22
What is your first and last name?
  [Unknown]:  Accounting Administrator
What is the name of your organizational unit?
  [Unknown]:  AccountingServices
What is the name of your organization?
  [Unknown]:  MyOrganization
What is the name of your City or Locality?
  [Unknown]:  Raleigh
```

```
What is the name of your State or Province?
  [Unknown]:  NC
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Accounting Administrator, OU=AccountingServices,
O=MyOrganization, L=Raleigh, ST=NC, C=US correct?
  [no]:  yes

Enter key password for <vault>
        (RETURN if same as keystore password):
```

Result

A file named **`vault.keystore`** is created in the **`/home/`*`USER`*`/vault/`** directory. It stores a single key, called **`vault`**, which will be used to store encrypted strings, such as passwords, for JBoss EAP 6.

Report a bug

10.11.3. Mask the Keystore Password and Initialize the Password Vault

Prerequisites

- Section 10.11.2, “Create a Java Keystore to Store Sensitive Strings”
- The ***`EAP_HOME`*`/bin/vault.sh`** application needs to be accessible via a command-line interface.

1. **Run the `vault.sh` command.**
   Run ***`EAP_HOME`*`/bin/vault.sh`**. Start a new interactive session by typing **`0`**.

2. **Enter the directory where encrypted files will be stored.**
   This directory should be reasonably secure, but JBoss EAP 6 needs to be able to access it. If you followed Section 10.11.2, “Create a Java Keystore to Store Sensitive Strings”, your keystore is in a directory called **`vault/`** in your home directory. This example uses the directory **`/home/`*`USER`*`/vault/`**.

   NOTE

   Do not forget to include the trailing slash on the directory name. Either use **`/`** or **`\`**, depending on your operating system.

3. **Enter the path to the keystore.**
   Enter the full path to the keystore file. This example uses **`/home/`*`USER`*`/vault/vault.keystore`**.

4. **Encrypt the keystore password.**
   The following steps encrypt the keystore password, so that you can use it in configuration files and applications securely.

   a. **Enter the keystore password.**
      When prompted, enter the keystore password.

   b. **Enter a salt value.**

Enter an 8-character salt value. The salt value, together with the iteration count (below), are used to create the hash value.

c. **Enter the iteration count.**
Enter a number for the iteration count.

d. **Make a note of the masked password information.**
The masked password, the salt, and the iteration count are printed to standard output. Make a note of them in a secure location. An attacker could use them to decrypt the password.

e. **Enter the alias of the vault.**
When prompted, enter the alias of the vault. If you followed Section 10.11.2, "Create a Java Keystore to Store Sensitive Strings" to create your vault, the alias is **`vault`**.

5. **Exit the interactive console.**
Type **2** to exit the interactive console.

Result

Your keystore password has been masked for use in configuration files and deployments. In addition, your vault is fully configured and ready to use.

Report a bug

10.11.4. Configure JBoss EAP 6 to Use the Password Vault

Overview

Before you can mask passwords and other sensitive attributes in configuration files, you need to make JBoss EAP 6 aware of the password vault which stores and decrypts them. Follow this procedure to enable this functionality.

Prerequisites

- Section 10.11.2, "Create a Java Keystore to Store Sensitive Strings"
- Section 10.11.3, "Mask the Keystore Password and Initialize the Password Vault"

Procedure 10.6. Setup a Password Vault

1. **Determine the correct values for the command.**
Determine the values for the following parameters, which are determined by the commands used to create the keystore itself. For information on creating a keystore, refer to the following topics: Section 10.11.2, "Create a Java Keystore to Store Sensitive Strings" and Section 10.11.3, "Mask the Keystore Password and Initialize the Password Vault".

Parameter	Description
KEYSTORE_URL	The file system path or URI of the keystore file, usually called something like **`vault.keystore`**
KEYSTORE_PASSWORD	The password used to access the keystore. This value should be masked.

Parameter	Description
KEYSTORE_ALIAS	The name of the keystore.
SALT	The salt used to encrypt and decrypt keystore values.
ITERATION_COUNT	The number of times the encryption algorithm is run.
ENC_FILE_DIR	The path to the directory from which the keystore commands are run. Typically the directory containing the password vault.
host (managed domain only)	The name of the host you are configuring

2. **Use the Management CLI to enable the password vault.**
   Run one of the following commands, depending on whether you use a managed domain or standalone server configuration. Substitute the values in the command with the ones from the first step of this procedure.

   NOTE

   If you use Microsoft Windows Server, replace each `/` character in a filename or directory path with four `\` characters. This is because it should be two `\` characters, each escaped. This does not need to be done for other `/` characters.

   - **Managed Domain**

     ```
     /host=YOUR_HOST/core-service=vault:add(vault-options=
     [("KEYSTORE_URL" => "PATH_TO_KEYSTORE"), ("KEYSTORE_PASSWORD" =>
     "MASKED_PASSWORD"), ("KEYSTORE_ALIAS" => "ALIAS"), ("SALT" =>
     "SALT"),("ITERATION_COUNT" => "ITERATION_COUNT"), ("ENC_FILE_DIR"
     => "ENC_FILE_DIR")])
     ```

   - **Standalone Server**

     ```
     /core-service=vault:add(vault-options=[("KEYSTORE_URL" =>
     "PATH_TO_KEYSTORE"), ("KEYSTORE_PASSWORD" => "MASKED_PASSWORD"),
     ("KEYSTORE_ALIAS" => "ALIAS"), ("SALT" => "SALT"),
     ("ITERATION_COUNT" => "ITERATION_COUNT"), ("ENC_FILE_DIR" =>
     "ENC_FILE_DIR")])
     ```

The following is an example of the command with hypothetical values:

```
/core-service=vault:add(vault-options=[("KEYSTORE_URL" =>
"/home/user/vault/vault.keystore"), ("KEYSTORE_PASSWORD" => "MASK-
3y28rCZlcKR"), ("KEYSTORE_ALIAS" => "vault"), ("SALT" =>
"12438567"),("ITERATION_COUNT" => "50"), ("ENC_FILE_DIR" =>
"/home/user/vault/")])
```

Result

JBoss EAP 6 is configured to decrypt masked strings using the password vault. To add strings to the vault and use them in your configuration, refer to the following topic: Section 10.11.5, “Store and Retrieve Encrypted Sensitive Strings in the Java Keystore”.

Report a bug

10.11.5. Store and Retrieve Encrypted Sensitive Strings in the Java Keystore

Summary

Including passwords and other sensitive strings in plain-text configuration files is insecure. JBoss EAP 6 includes the ability to store and mask these sensitive strings in an encrypted keystore, and use masked values in configuration files.

Prerequisites

- Section 10.11.2, “Create a Java Keystore to Store Sensitive Strings”
- Section 10.11.3, “Mask the Keystore Password and Initialize the Password Vault”
- Section 10.11.4, “Configure JBoss EAP 6 to Use the Password Vault”
- The ***EAP_HOME*/bin/vault.sh** application needs to be accessible via a command-line interface.

Procedure 10.7. Setup the Java Keystore

1. **Run the `vault.sh` command.**
   Run ***EAP_HOME*/bin/vault.sh**. Start a new interactive session by typing **0**.

2. **Enter the directory where encrypted files will be stored.**
   If you followed Section 10.11.2, “Create a Java Keystore to Store Sensitive Strings”, your keystore is in a directory called **vault/** in your home directory. In most cases, it makes sense to store all of your encrypted information in the same place as the key store. This example uses the directory **/home/*USER*/vault/**.

   NOTE

   Do not forget to include the trailing slash on the directory name. Either use `/` or `\`, depending on your operating system.

3. **Enter the path to the keystore.**
   Enter the full path to the keystore file. This example uses **/home/*USER*/vault/vault.keystore**.

4. **Enter the keystore password, vault name, salt, and iteration count.**
   When prompted, enter the keystore password, vault name, salt, and iteration count. A handshake is performed.

5. **Select the option to store a password.**
   Select option **0** to store a password or other sensitive string.

6. **Enter the value.**

When prompted, enter the value twice. If the values do not match, you are prompted to try again.

7. **Enter the vault block.**
   Enter the vault block, which is a container for attributes which pertain to the same resource. An example of an attribute name would be **`ds_ExampleDS`**. This will form part of the reference to the encrypted string, in your datasource or other service definition.

8. **Enter the attribute name.**
   Enter the name of the attribute you are storing. An example attribute name would be **`password`**.

   Result

   A message such as the one below shows that the attribute has been saved.

   ```
   Attribute Value for (ds_ExampleDS, password) saved
   ```

9. **Make note of the information about the encrypted string.**
   A message prints to standard output, showing the vault block, attribute name, shared key, and advice about using the string in your configuration. Make note of this information in a secure location. Example output is shown below.

   ```
   ********************************************
   Vault Block:ds_ExampleDS
   Attribute Name:password
   Configuration should be done as follows:
   VAULT::ds_ExampleDS::password::1
   ********************************************
   ```

10. **Use the encrypted string in your configuration.**
    Use the string from the previous step in your configuration, in place of a plain-text string. A datasource using the encrypted password above is shown below.

    ```
    ...
      <subsystem xmlns="urn:jboss:domain:datasources:1.0">
        <datasources>
          <datasource jndi-name="java:jboss/datasources/ExampleDS"
    enabled="true" use-java-context="true" pool-name="H2DS">
            <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
    1</connection-url>
            <driver>h2</driver>
            <pool></pool>
            <security>
              <user-name>sa</user-name>
              <password>${VAULT::ds_ExampleDS::password::1}</password>
            </security>
          </datasource>
          <drivers>
             <driver name="h2" module="com.h2database.h2">
                <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
    datasource-class>
             </driver>
          </drivers>
        </datasources>
      </subsystem>
    ...
    ```

You can use an encrypted string anywhere in your domain or standalone configuration file where expressions are allowed.

NOTE

To check if expressions are allowed within a particular subsystem, run the following CLI command against that subsystem:

```
/host=master/core-service=management/security-
realm=TestRealm:read-resource-description(recursive=true)
```

From the output of running this command, look for the value for the **`expressions-allowed`** parameter. If this is true, then you can use expressions within the configuration of this particular subsystem.

After you store your string in the keystore, use the following syntax to replace any clear-text string with an encrypted one.

```
${VAULT::<replaceable>VAULT_BLOCK</replaceable>::
<replaceable>ATTRIBUTE_NAME</replaceable>::
<replaceable>ENCRYPTED_VALUE</replaceable>}
```

Here is a sample real-world value, where the vault block is **`ds_ExampleDS`** and the attribute is **`password`**.

```
<password>${VAULT::ds_ExampleDS::password::1}</password>
```

Report a bug

10.11.6. Store and Resolve Sensitive Strings In Your Applications

Overview

Configuration elements of JBoss EAP 6 support the ability to resolve encrypted strings against values stored in a Java Keystore, via the Security Vault mechanism. You can add support for this feature to your own applications.

First, add the password to the vault. Second, replace the clear-text password with the one stored in the vault. You can use this method to obscure any sensitive string in your application.

Prerequisites

Before performing this procedure, make sure that the directory for storing your vault files exists. It does not matter where you place them, as long as the user who executes JBoss EAP 6 has permission to read and write the files. This example places the **`vault/`** directory into the **`/home/`*****`USER`*****`/vault/`** directory. The vault itself is a file called **`vault.keystore`** inside the **`vault/`** directory.

Example 10.11. Adding the Password String to the Vault

Add the string to the vault using the ***`EAP_HOME`*****`/bin/vault.sh`** command. The full series of commands and responses is included in the following screen output. Values entered by the user are emphasized. Some output is removed for formatting. In Microsoft Windows, the name of the command is **`vault.bat`**. Note that in Microsoft Windows, file paths use the `\` character as a directory separator, rather than the `/` character.

```
[user@host bin]$ ./vault.sh
**********************************
****  JBoss Vault ********
**********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove
Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:/home/user/vault/
Enter Keystore URL:/home/user/vault/vault.keystore
Enter Keystore password: ...
Enter Keystore password again: ...
Values match
Enter 8 character salt:12345678
Enter iteration count as a number (Eg: 44):25

Enter Keystore Alias:vault
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::   0: Store a password  1: Check whether password
exists  2: Exit
0
Task:  Store a password
Please enter attribute value: sa
Please enter attribute value again: sa
Values match
Enter Vault Block:DS
Enter Attribute Name:thePass
Attribute Value for (DS, thePass) saved

Please make note of the following:
********************************************
Vault Block:DS
Attribute Name:thePass
Configuration should be done as follows:
VAULT::DS::thePass::1
********************************************

Please enter a Digit::   0: Store a password  1: Check whether password
exists  2: Exit
2
```

The string that will be added to the Java code is the last value of the output, the line beginning with **VAULT**.

The following servlet uses the vaulted string instead of a clear-text password. The clear-text version is commented out so that you can see the difference.

Example 10.12. Servlet Using a Vaulted Password

```
package vaulterror.web;

import java.io.IOException;
import java.io.Writer;
import javax.annotation.Resource;
import javax.annotation.sql.DataSourceDefinition;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;


/*@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "sa",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)*/
@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "VAULT::DS::thePass::1",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)
@WebServlet(name = "MyTestServlet", urlPatterns = { "/my/" },
loadOnStartup = 1)
public class MyTestServlet  extends HttpServlet {

    private static final long serialVersionUID = 1L;


    @Resource(lookup = "java:jboss/datasources/LoginDS")
    private DataSource ds;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
        Writer writer = resp.getWriter();
        writer.write((ds != null) + "");
    }
}
```

Your servlet is now able to resolve the vaulted string.

Report a bug

10.12. LDAP

10.12.1. About 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.

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

JBoss EAP 6 includes several authentication and authorization modules which allow you to use a LDAP server as the authentication and authorization authority for your Web and EJB applications.

Report a bug

10.12.2. Use LDAP to Authenticate to the Management Interfaces

To use an LDAP directory server as the authentication source for the Management Console, Management CLI, or Management API, you need to perform the following procedures:

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

Create an Outbound Connection to an LDAP Server

The LDAP outbound connection allows the following attributes:

Table 10.1. Attributes of an LDAP Outbound Connection

Attribute	Required	Description
url	yes	The URL address of the directory server.
search-dn	yes	The fully distinguished name (DN) of the user authorized to perform searches.
search-credentials	yes	The password of the user authorized to perform searches.
initial-context-factory	no	The initial context factory to use when establishing the connection. Defaults to **`com.sun.jndi.ldap.LdapCtxFactory`**.
security-realm	no	The security realm to reference to obtain a configured **`SSLContext`** to use when establishing the connection.

Example 10.13. Add an LDAP Outbound Connection

This example adds an outbound connection with the following properties set:

- Search DN: **`cn=search,dc=acme,dc=com`**
- Search Credential: **`myPass`**

- URL: **`ldap://127.0.0.1:389`**

The first command adds the security realm.

```
/host=master/core-service=management/security-
realm=ldap_security_realm:add
```

The second command adds the LDAP connection.

```
/host=master/core-service=management/ldap-
connection=ldap_connection/:add(search-
credential=myPass,url=ldap://127.0.0.1:389,search-
dn="cn=search,dc=acme,dc=com")
```

Create an LDAP-Enabled Security Realm

The Management Interfaces can authenticate against LDAP server instead of the property-file based security realms configured by default. 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 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.

The LDAP security realm needs the following configuration attributes and elements in order to perform its functions.

connection

The name of the connection defined in **`<outbound-connections>`** to use to connect to the LDAP directory.

base-dn

The distinguished name of the context to begin searching for the user.

recursive

Whether the search should be recursive throughout the LDAP directory tree, or only search the specified context. Defaults to **`false`**.

user-dn

The attribute of the user that holds the distinguished name. This is subsequently used to test authentication as the user can complete. Defaults to **`dn`**.

One of `username-filter` or `advanced-filter`, as a child element

The **`username-filter`** takes a single attribute called **`attribute`**, whose value is the name of the LDAP attribute which holds the username, such as **`userName`** or **`sambaAccountName`**.

The **`advanced-filter`** takes a single attribute called **`filter`**. This attribute contains a filter query in standard LDAP syntax. Be cautious to escape any **`&`** characters by changing them to **`&amp;`**. An example of a filter is:

```
(&(sAMAccountName={0})(memberOf=cn=admin,cn=users,dc=acme,dc=com))
```

After escaping the ampersand character, the filter appears as:

```
(&amp;(sAMAccountName={0})(memberOf=cn=admin,cn=users,dc=acme,dc=com))
```

Example 10.14. XML Representing an LDAP-enabled Security Realm

This example uses the following parameters:

- connection - **`ldap_connection`**
- base-dn - **`cn=users,dc=acme,dc=com`**.
- username-filter - **`attribute="sambaAccountName"`**

```
<security-realm name="ldap_security_realm">
   <authentication>
      <ldap connection="ldap_connection" base-
dn="cn=users,dc=acme,dc=com">
         <username-filter attribute="sambaAccountName" />
      </ldap>
  </authentication>
</security-realm>
```

WARNING

It is important to ensure that you do not allow empty LDAP passwords; unless you specifically desire this in your environment, it is a serious security concern.

EAP 6.1 includes a patch for CVE-2012-5629, which sets the allowEmptyPasswords option for the LDAP login modules to false if the option is not already configured. For older versions, this option should be configured manually

Example 10.15. Add an LDAP Security Realm

The command below adds a security realm and sets its attributes for a standalone server.

```
/host=master/core-service=management/security-
realm=ldap_security_realm/authentication=ldap:add(base-
dn="DC=mycompany,DC=org", recursive=true, username-
attribute="MyAccountName", connection="ldap_connection")
```

Apply the New Security Realm to the Management Interface

After you create a security realm, you need to reference it in the configuration of your management interface. The management interface will use the security realm for HTTP digest authentication.

Example 10.16. Apply the Security Realm to the HTTP Interface

After this configuration is in place, and you restart the host controller, the web-based Management Console will use LDAP to authenticate its users.

```
/host=master/core-service=management/management-interface=http-
interface/:write-attribute(name=security-realm,value=ldap-security-
realm)
```

Configure a Managed Domain Member to Authenticate using Microsoft Active Directory

To configure a host in a managed domain to authenticate to Microsoft Active Directory, follow this procedure, which creates a security domain and maps roles to Active Directory groups, using JAAS authentication. This procedure is required because Microsoft Active Directory allows binding with an empty password. This procedure prevents an empty password from being used within the application platform.

Before performing this procedure, you need to know the name of your host controller. This example assumes the host controller is named **`master`**.

1. **Add a new `<security-realm>` named `ldap_security_realm`, and configure it to use JAAS.**
   The following Management CLI commands add the new security realm and set its authentication mechanism. Change the name of the host as required.

   ```
   /host=master/core-service=management/security-
   realm=ldap_security_realm/:add
   ```

   ```
   /host=master/core-service=management/security-
   realm=ldap_security_realm/authentication=jaas/:add(name=managementLD
   APDomain)
   ```

2. **Configure the `<http-interface>` to use the new security realm.**
   The following Management CLI command configures the HTTP interface.

   ```
   /host=master/core-service=management/management-interface=http-
   interface/:write-attribute(name=security-
   realm,value=ldap_security_realm)
   ```

3. **Configure JBoss Enterprise Application Platform to add the custom JAAS configuration to its start-up parameters.**
   Edit the ***`EAP_HOME`*`/bin/domain.conf`** file. Search for the **`HOST_CONTROLLER_JAVA_OPTS`** variable. This is where you add directives for the JVM which are needed before JBoss Enterprise Application Platform starts. The following is an example of the default contents of this parameter:

   ```
   HOST_CONTROLLER_JAVA_OPTS="$JAVA_OPTS"
   ```

   Add the following directive to the line: **`-Djava.security.auth.login.config=/opt/jboss-eap-6.0/domain/configuration/jaas.conf"`**

The edited line is similar to the following:

```
-Djava.security.auth.login.config=/opt/jboss-eap-
6.0/domain/configuration/jaas.conf"
```

4. **Add the login module to the module options.**
   In the same file, find the line containing the following:

   ```
   JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman"
   ```

   Change that line to read as follows. Make sure not to insert any extra spaces.

   ```
   JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman,com.sun.security.auth.l
   ogin"
   ```

   Save and close the **`domain.conf`** file.

5. **Create the JAAS configuration which will be added to the classpath.**
   Create a new file at the following location: ***`EAP_HOME`*****`/domain/configuration/jaas.conf`**

   The file should contain the following contents. Edit the parameters to match your own environment.

   ```
   managementLDAPDomain {
       org.jboss.security.auth.spi.LdapExtLoginModule required

   java.naming.factory.initial="com.sun.jndi.ldap.LdapCtxFactory"

   java.naming.provider.url="ldap://your_active_directory_host:389"
           java.naming.security.authentication="simple"

   bindDN="cn=Administrator,cn=users,dc=domain,dc=your_company,dc=com"
           bindCredential="password"
           baseCtxDN="cn=users,dc=domain,dc=redhat,dc=com"
           baseFilter="(&(sAMAccountName={0})(|(memberOf=cn=Domain
   Guests,cn=Users,dc=domain,dc=acme,dc=com)(memberOf=cn=Domain
   Admins,cn=Users,dc=domain,dc=acme,dc=com)))"
           allowEmptyPasswords="false"
           rolesCtxDN="cn=users,dc=domain,dc=acme,dc=com"
           roleFilter="(cn=no such group)"
           searchScope="SUBTREE_SCOPE";
   };
   ```

6. Restart JBoss Enterprise Application Platform and your HTTP interface uses your LDAP server for authentication.

Report a bug

CHAPTER 11. SECURING THE MANAGEMENT INTERFACES WITH ROLE-BASED ACCESS CONTROL

11.1. ABOUT ROLE-BASED ACCESS CONTROL (RBAC)

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 6.2 servers without each of them requiring unrestricted access. By providing "separation of duties" for management users, JBoss EAP 6.2 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 6.2 works through a combination of role permissions and constraints.

Seven predefined roles are provided that each have different fixed permissions. The predefined roles are: Monitor, Operator, Maintainer, Deployer, Auditor, Administrator, and SuperUser. Each management user is assigned one or more roles, what the user is permitted to do when managing the server is specified by their assigned roles.

Report a bug

11.2. ROLE-BASED ACCESS CONTROL IN THE GUI AND CLI

When Role-Based Access Control(RBAC) is enabled, some users will not be able to run some operations, read from some resources or even be able to see parts of the management model at all, according to the roles assigned to them.

The Management Console

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

If you do 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 you do not have write permissions to a resource attribute, that attribute will be disabled (greyed-out) in the edit form for the resource. If you do not have write permissions to the resource at all, then the edit button for the resource will not appear.

If you do not have permissions to access a resource or attribute at all (it is "unaddressable" for your role) then it does not appear in the console at all for you. An example of that is the access control system itself which is only visible to a few roles by default.

The Management API

Users that use the **`jboss-cli.sh`** tool or use the API directly will encounter slightly different behaviour 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 at all.

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.

Report a bug

11.3. SUPPORTED AUTHENTICATION SCHEMES

Role-Based Access Control works with the standard authentication providers that are included with JBoss EAP 6.2. The standard authentication providers are: **`username/password`**, **`client certificate`**, and **`local user`**.

Username/Password

Users are authenticated using a username and password combination which is verified against either the **`mgmt-users.properties`** file, or an LDAP server.

Client Certificate

Using the Trust Store.

Local User

`jboss-cli.sh` 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 **`mgmt-users.properties`** file 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.

Report a bug

11.4. THE 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 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. Administrator is the only role (except SuperUser) that 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.

Report a bug

11.5. ABOUT ROLE 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 11.1, “Role Permissions Matrix” summarizes the permissions of each role.

Table 11.1. Role Permissions Matrix

	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
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.

Report a bug

11.6. ABOUT 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 two classifications: application, sensitivity.

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.

Application constraint configuration is in the Management API at **`/core-service=management/access=authorization/constraint=application-classification`**.

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.

Sensitivity constraint configuration is in the Management API at **`/core-service=management/access=authorization/constraint=sensitivity-classification`**.

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.

Vault Expression constraint configuration is in the Management API at **`/core-service=management/access=authorization/constraint=vault-expression`**.

Constraints can not be configured in the Management Console at this time.

Report a bug

11.7. ABOUT JMX AND ROLE-BASED ACCESS CONTROL

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

1. The Management API of JBoss EAP 6 is exposed as JMX Management Beans. These Management 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 Management 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, SuperUser roles can write to them.

Report a bug

11.8. CONFIGURING ROLE-BASED ACCESS CONTROL

11.8.1. Overview of RBAC Configuration Tasks

When RBAC is enabled only users of the Administration or SuperUser role can view and make changes to the Access Control system.

The Management Console 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

The CLI provides access to the complete access control system. This means that everything that can be done in the management console can be done there, but also a number of additional tasks can be performed with the CLI that cannot be done with the access control system.

The following additional tasks can be performed in the CLI:

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

Report a bug

11.8.2. Enabling Role-Based Access Control

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

Once enabled it can only be disabled by a user of the Administrator or SuperUser roles. By default the **`jboss-cli.sh`** runs as the SuperUser role if it is run on the same machine as the server.

Procedure 11.1. Enabling RBAC

- To enable RBAC with **`jboss-cli.sh`** use the **`write-attribute`** operation of the access authorization resource to set the provider attribute to **`rbac`**.

  ```
  /core-service=management/access=authorization:write-
  attribute(name=provider, value=rbac)
  ```

  ```
  [standalone@localhost:9999 /] /core-
  ```

```
service=management/access=authorization:write-
attribute(name=provider, value=rbac)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
[standalone@localhost:9999 /] /:reload
{
    "outcome" => "success",
    "result" => undefined
}
[standalone@localhost:9999 /]
```

Procedure 11.2. Disabling RBAC

- To disable RBAC with **`jboss-cli.sh`** use the **`write-attribute`** operation of the access authorization resource to set the provider attribute to **`simple`**.

```
/core-service=management/access=authorization:write-
attribute(name=provider, value=simple)
```

```
[standalone@localhost:9999 /] /core-
service=management/access=authorization:write-
attribute(name=provider, value=simple)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
[standalone@localhost:9999 /] /:reload
{
    "outcome" => "success",
    "result" => undefined
}
[standalone@localhost:9999 /]
```

If the server is off-line the XML configuration can be edited to enabled or disable RBAC. To do this, edit the **`provider`** attribute of the access-control element of the management element. Set the value to **`rbac`** to enable, and **`simple`** to disable.

```
<management>

        <access-control provider="rbac">
            <role-mapping>
                <role name="SuperUser">
                    <include>
                        <user name="$local"/>
                    </include>
                </role>
            </role-mapping>
        </access-control>

    </management>
```

Report a bug

11.8.3. Changing the Permission Combination Policy

Permission Combination Policy determines how permissions are determined if a user is assigned more than one role. This can be set to **`permissive`** or **`rejecting`**. The default is **`permissive`**.

When set to **`permissive`**, if any role is assigned to the user that permits an action, then the action is allowed.

When set to **`rejecting`**, if multiple roles are assigned to a user that permit an action, then the action is *not* allowed.

When the policy is set to rejecting each user should only be assigned one role. Users with multiple roles will not be able to use the management console or **`jboss-cli.sh`** tool when the policy is set to **`rejecting`**.

The Permission Combination Policy is configured by setting the **`permission-combination-policy`** attribute to either **`permissive`** or **`rejecting`**. This can be done using the **`jboss-cli.sh`** tool or by editing the server configuration XML file if the server is off-line.

Procedure 11.3. Set the Permission Combination Policy

- Use the **`write-attribute`** operation of the access authorization resource to set the **`permission-combination-policy`** attribute to the required policy name.

  ```
  /core-service=management/access=authorization:write-
  attribute(name=permission-combination-policy, value=POLICYNAME)
  ```

  The valid policy names are rejecting and permissive.

  ```
  [standalone@localhost:9999 /] /core-
  service=management/access=authorization:write-
  attribute(name=permission-combination-policy, value=rejecting)
  {"outcome" => "success"}
  [standalone@localhost:9999 access=authorization]
  ```

If the server is off-line the XML configuration can be edited to change the permission combination policy value. To do this, edit the **`permission-combination-policy`** attribute of the access-control element.

```
<access-control provider="rbac" permission-combination-policy="rejecting">
  <role-mapping>
    <role name="SuperUser">
      <include>
        <user name="$local"/>
      </include>
    </role>
  </role-mapping>
</access-control>
```

Report a bug

11.9. MANAGING ROLES

11.9.1. About Role Membership

When Role-Based Access Control (RBAC) is enabled, what a management user is permitted to do is determined by the roles that the user is assigned to. JBoss EAP 6.2 uses a system of includes and excludes based on both the user and group membership to determine what role a user belongs to.

A user is considered to be assigned to a role if:

1. The user is:
    - listed as a user to be included in the role, or
    - a member of a group that is listed to be included in the role.
2. The user is not:
    - listed as a user to exclude from the role, or
    - a member of a group that is listed to be excluded from the role.

Exclusions take priority over inclusions.

Role include and exclude settings for users and groups can be configured using both the Management Console and the **`jboss-cli.sh`** tool.

Only users of the SuperUser or Administrator roles can perform this configuration.

Report a bug

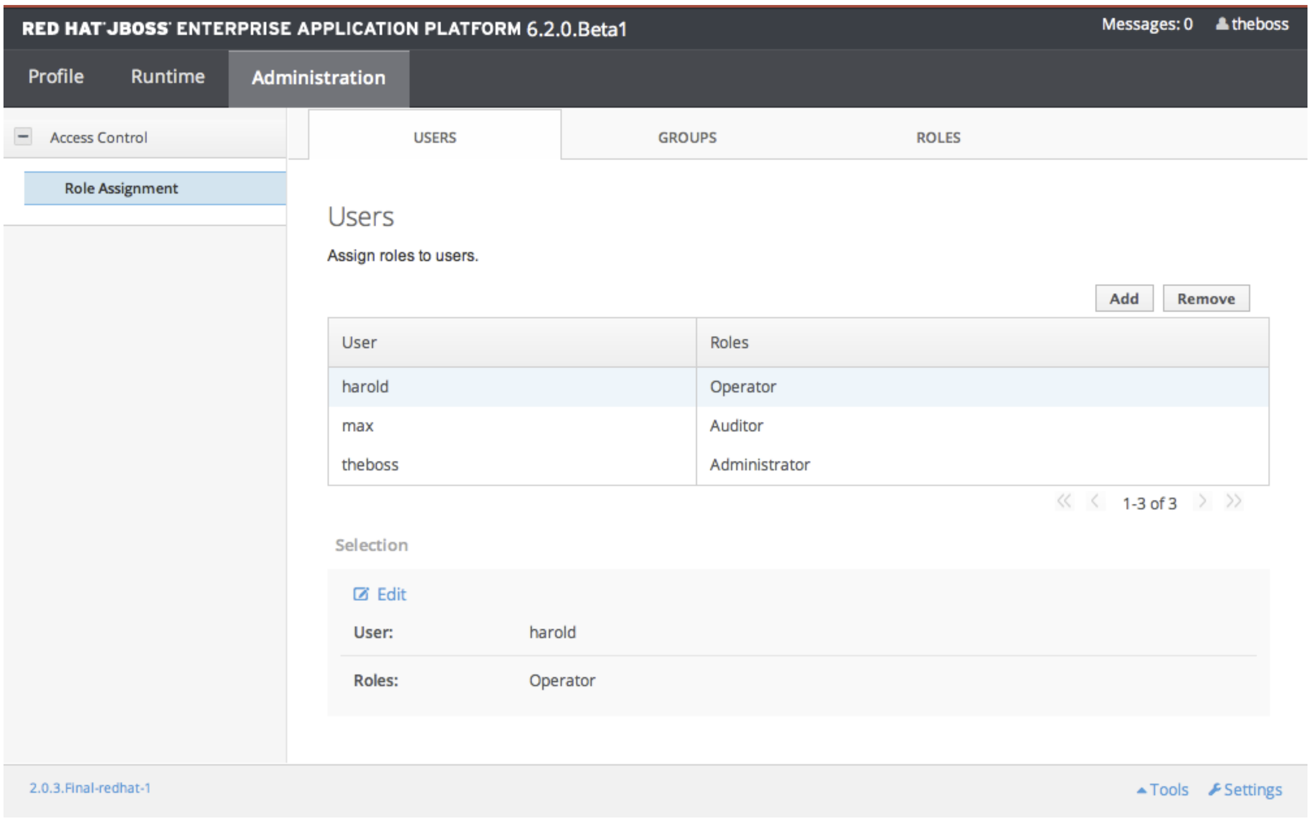
11.9.2. Configure User Role Assignment

Roles for a user to be included in and excluded from can be configured in the Management Console and the **`jboss-cli.sh`** tool. This topic only shows using the Management Console.

Only users in the SuperUser or Administrator roles can perform this configuration.

The User roles configuration in the management console can be found by following these steps:

1. Login to the Management Console.
2. Click on the Administration tab.
3. Expand the Access Control item on the left and select Role Assignment.
4. Select the USERS tab.



Users

Assign roles to users.

User	Roles
harold	Operator
max	Auditor
theboss	Administrator

Selection

User:	harold
Roles:	Operator

Figure 11.1. User Role Management in the Management Console

Procedure 11.4. Create a new role assignment for a user

1. Login to the Management console.
2. Navigate to the Users tab of the Role Assignment section.
3. Click the Add button at the top right of the user list. Add User dialog appears.

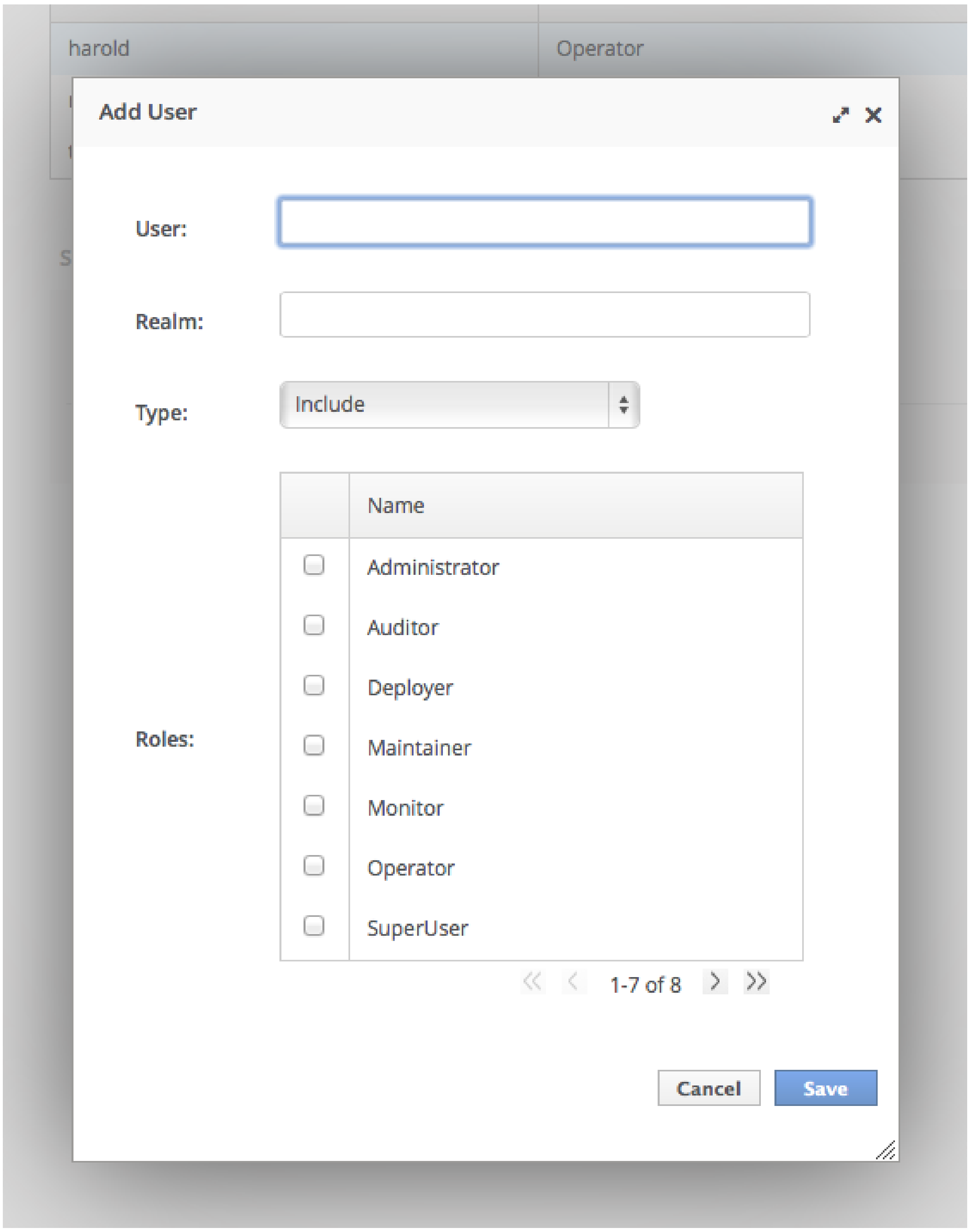


Figure 11.2. Add User Dialog

4. Specify user name, and optionally realm.

5. Set the type menu to include or exclude.

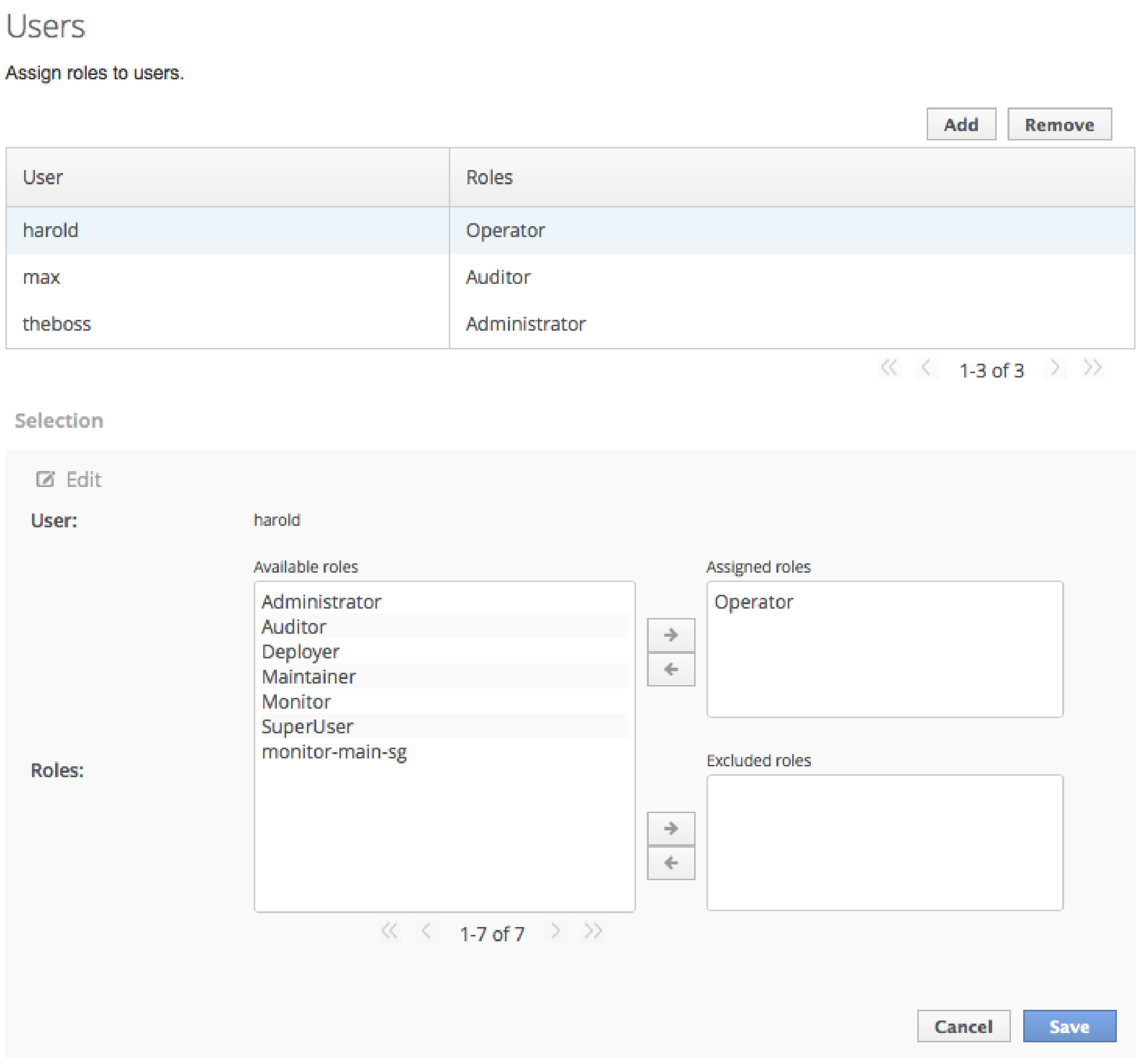
6. Click the checkbox of the roles to include or exclude. You can use the Control key (Command key on OSX) to check multiple items.

7. Click save.

   When successful, the Add User dialog closes, and the list of users is updated to reflect the changes made. If unsuccessful a "Failed to save role assignment" message is displayed.

Procedure 11.5. Update the role assignment for a user

1. Login to the Management console.
2. Navigate to the Users tab of the Role Assignment section.
3. Select user from the list.
4. Click Edit. The selection panel enters edit mode.


Users

Assign roles to users.

User	Roles
harold	Operator
max	Auditor
theboss	Administrator

1-3 of 3

Selection

User: harold

Available roles: Administrator, Auditor, Deployer, Maintainer, Monitor, SuperUser, monitor-main-sg (1-7 of 7)

Assigned roles: Operator

Excluded roles:

Figure 11.3. Selection Edit View

Here you can add and remove assigned and excluded roles for the user.

1. To add an assigned role, select the required role from the list of available roles on the left and click button with the right-facing arrow next to the assigned roles list. The role moves from the available list to the assigned list.
2. To remove an assigned role, selected the required role from the assigned roles list on the right and click the button with the left-facing arrow next to the assigned roles list. The role moves from the assigned list to the available list.

   3. To add an excluded role, select the required role from the list of available roles on the left and click button with the right-facing arrow next to the excluded roles list. The role moves from the available list to the excluded list.

   4. To remove an excluded role, selected the required role from the excluded roles list on the right and click the button with the left-facing arrow next to the excluded roles list. The role moves from the excluded list to the available list.

5. Click save.

   When successful, the edit view closes, and the list of users is updated to reflect the changes made. If unsuccessful a "Failed to save role assignment" message is displayed.

Procedure 11.6. Remove role assignment for a user

1. Login to the Management console.

2. Navigate to the Users tab of the Role Assignment section.

3. Select the user from list.

4. Click the Remove button. The Remove Role Assignment confirmation prompt appears.

5. Click the Confirm button.

   When successful, the user will no longer appear in the list of user role assignments.

IMPORTANT

Removing the user from the list of role assignments does not remove the user from the system, nor does it guarantee that no roles will be assigned to the user. Roles might still be assigned from group membership.

Report a bug

11.9.3. Configure User Role Assignment using jboss-cli.sh

Roles for a user to be included in and excluded from can be configured in the Management Console and the **`jboss-cli.sh`** tool. This topic only shows using the **`jboss-cli.sh`** tool.

The configuration of mapping users and groups to roles is located in the management API at: **`/core-service=management/access=authorization`** as **`role-mapping`** elements.

Only users of the SuperUser or Administrator roles can perform this configuration.

Procedure 11.7. Viewing Role Assignment Configuration

1. Use the :read-children-names operation to get a complete list of the configured roles:

```
/core-service=management/access=authorization:read-children-
names(child-type=role-mapping)
```

```
[standalone@localhost:9999 access=authorization] :read-children-
names(child-type=role-mapping)
```

```
{
    "outcome" => "success",
    "result" => [
        "ADMINISTRATOR",
        "DEPLOYER",
        "MAINTAINER",
        "MONITOR",
        "OPERATOR",
        "SuperUser"
    ]
}
```

2. Use the **`read-resource`** operation of a specified role-mapping to get the full details of a specific role:

```
/core-service=management/access=authorization/role-
mapping=ROLENAME:read-resource(recursive=true)
```

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=ADMINISTRATOR:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "include-all" => false,
        "exclude" => undefined,
        "include" => {
            "user-theboss" => {
                "name" => "theboss",
                "realm" => undefined,
                "type" => "USER"
            },
            "user-harold" => {
                "name" => "harold",
                "realm" => undefined,
                "type" => "USER"
            },
            "group-SysOps" => {
                "name" => "SysOps",
                "realm" => undefined,
                "type" => "GROUP"
            }
        }
    }
}
[standalone@localhost:9999 access=authorization]
```

Procedure 11.8. Add a new role

This procedure shows how to add a role-mapping entry for a role. This must be done before the role can be configured.

- Use the **add** operation to add a new role configuration.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME:add
```

ROLENAME is the name of the role that the new mapping is for.

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=AUDITOR:add
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

Procedure 11.9. Add a user as included in a role

This procedure shows how to add a user to the included list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be done first.

- Use the **add** operation to add a user entry to the includes list of the role.

  ```
  /core-service=management/access=authorization/role-
  mapping=ROLENAME/include=ALIAS:add(name=USERNAME, type=USER)
  ```

  ROLENAME is the name of the role being configured.

  `ALIAS` is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as **`user-USERNAME`**.

  USERNAME is the name of the user being added to the include list.

  ```
   [standalone@localhost:9999 access=authorization] ./role-
  mapping=AUDITOR/include=user-max:add(name=max, type=USER)
  {"outcome" => "success"}
  [standalone@localhost:9999 access=authorization]
  ```

Procedure 11.10. Add a user as excluded in a role

This procedure shows how to add a user to the excluded list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be done first.

- Use the **add** operation to add a user entry to the excludes list of the role.

  ```
  /core-service=management/access=authorization/role-
  mapping=ROLENAME/exclude=ALIAS:add(name=USERNAME, type=USER)
  ```

  ROLENAME is the name of the role being configured.

  USERNAME is the name of the user being added to the exclude list.

  `ALIAS` is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as **`user-USERNAME`**.

  ```
  [standalone@localhost:9999 access=authorization] ./role-
  mapping=AUDITOR/exclude=user-max:add(name=max, type=USER)
  ```

```
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

Procedure 11.11. Remove user role include configuration

This procedure shows how to remove a user include entry from a role mapping.

- Use the **`remove`** operation to remove the entry.

  ```
  /core-service=management/access=authorization/role-
  mapping=ROLENAME/include=ALIAS:remove
  ```

  ROLENAME is the name of the role being configured

  `ALIAS` is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as **`user-`*****`USERNAME`***.

  ```
  [standalone@localhost:9999 access=authorization] ./role-
  mapping=AUDITOR/include=user-max:remove
  {"outcome" => "success"}
  [standalone@localhost:9999 access=authorization]
  ```

  Removing the user from the list of includes does not remove the user from the system, nor does it guarantee that the role won't be assigned to the user. The role might still be assigned based on group membership.

Procedure 11.12. Remove user role exclude configuration

This procedure shows how to remove an user exclude entry from a role mapping.

- Use the **`remove`** operation to remove the entry.

  ```
  /core-service=management/access=authorization/role-
  mapping=ROLENAME/exclude=ALIAS:remove
  ```

  ROLENAME is the name of the role being configured.

  `ALIAS` is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as **`user-`*****`USERNAME`***.

  ```
  [standalone@localhost:9999 access=authorization] ./role-
  mapping=AUDITOR/exclude=user-max:remove
  {"outcome" => "success"}
  [standalone@localhost:9999 access=authorization]
  ```

  Removing the user from the list of excludes does not remove the user from the system, nor does it guarantee the role will be assigned to the user. Roles might still be excluded based on group membership.

Report a bug

11.9.4. About Roles and User Groups

Users authenticated using either the **`mgmt-users.properties`** file or an LDAP server, can be members of user groups. A user group is an arbitrary label that can be assigned to one or more users.

The RBAC system can be configured to automatically assign roles to users depending on what user groups they are members of. It can also exclude users from roles based on group membership.

When using the **`mgmt-users.properties`** file, group information is stored in the **`mgmt-groups.properties`** file. When using LDAP the group information is stored in the LDAP sever and maintained by those responsible for the LDAP server.

Report a bug

11.9.5. Configure Group Role Assignment

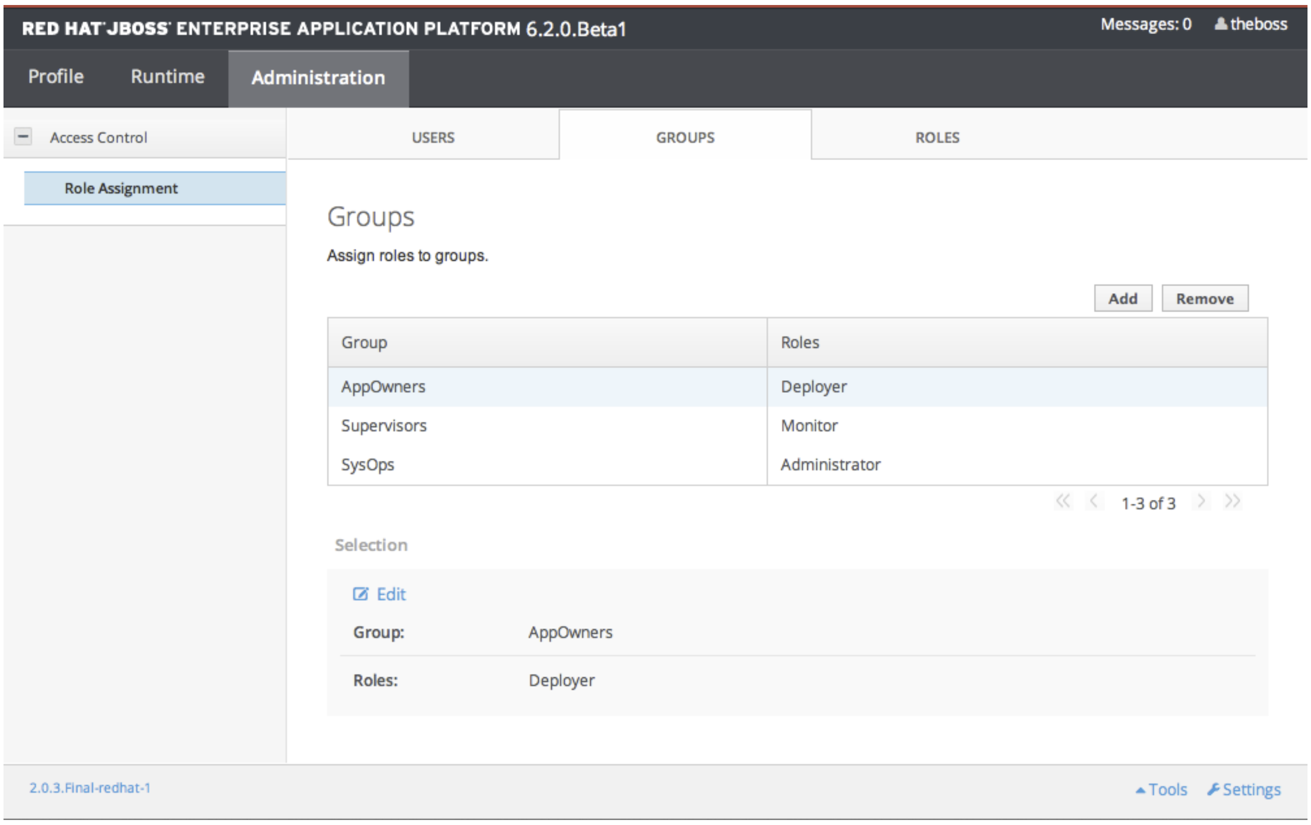
Roles can be assigned to a user based on the user's membership of a user group.

Groups to be included or excluded from a role can be configured in the Management Console and the **`jboss-cli.sh`** tool. This topic only shows using the Management Console.

Only users in the SuperUser or Administrator roles can perform this configuration.

The Group roles configuration in the management console can be found by following these steps:

1. Login to the Management Console.
2. Click on the Administration tab.
3. Expand the Access Control item on the left and select Role Assignment.
4. Select the GROUPS tab.



Groups

Assign roles to groups.

Group	Roles
AppOwners	Deployer
Supervisors	Monitor
SysOps	Administrator

1-3 of 3

Selection

Group: AppOwners

Roles: Deployer

Figure 11.4. Group Role Management in the Management Console

Procedure 11.13. Create a new role assignment for a group

1. Login to the Management console
2. Navigate to the GROUPS tab of the Role Assignment section.
3. Click the Add button at the top right of the user list. Add Group dialog appears

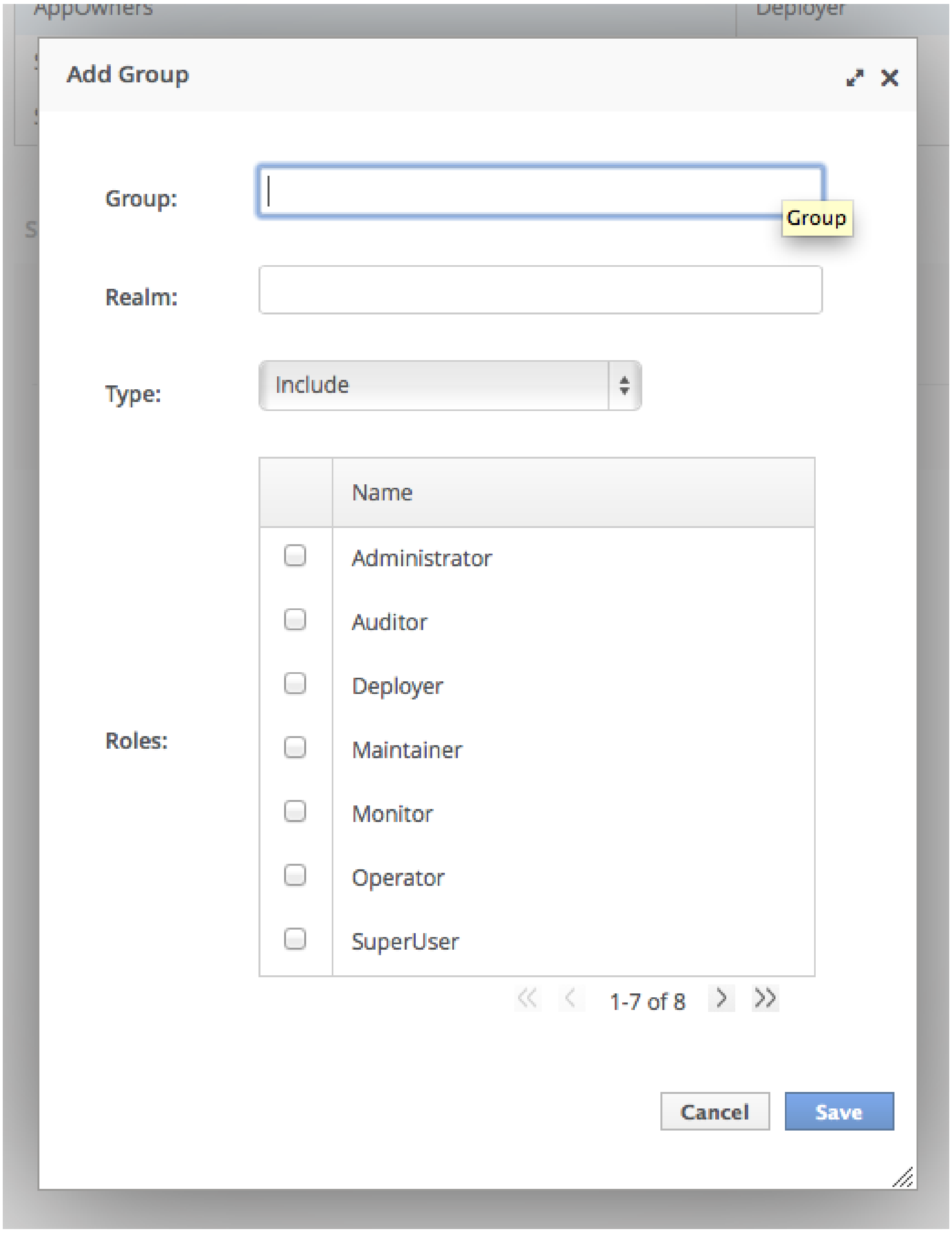


Figure 11.5. Add Group Dialog

4. Specify the group name, and optionally the realm.

5. Set the type menu to include or exclude.

6. Click the checkbox of the roles to include or exclude. You can use the Control key (Command key on OSX) to check multiple items.

7. Click save.

   When successful, the Add Group dialog closes, and the list of groups is updated to reflect the changes made. If unsuccessful a "Failed to save role assignment" message is displayed.

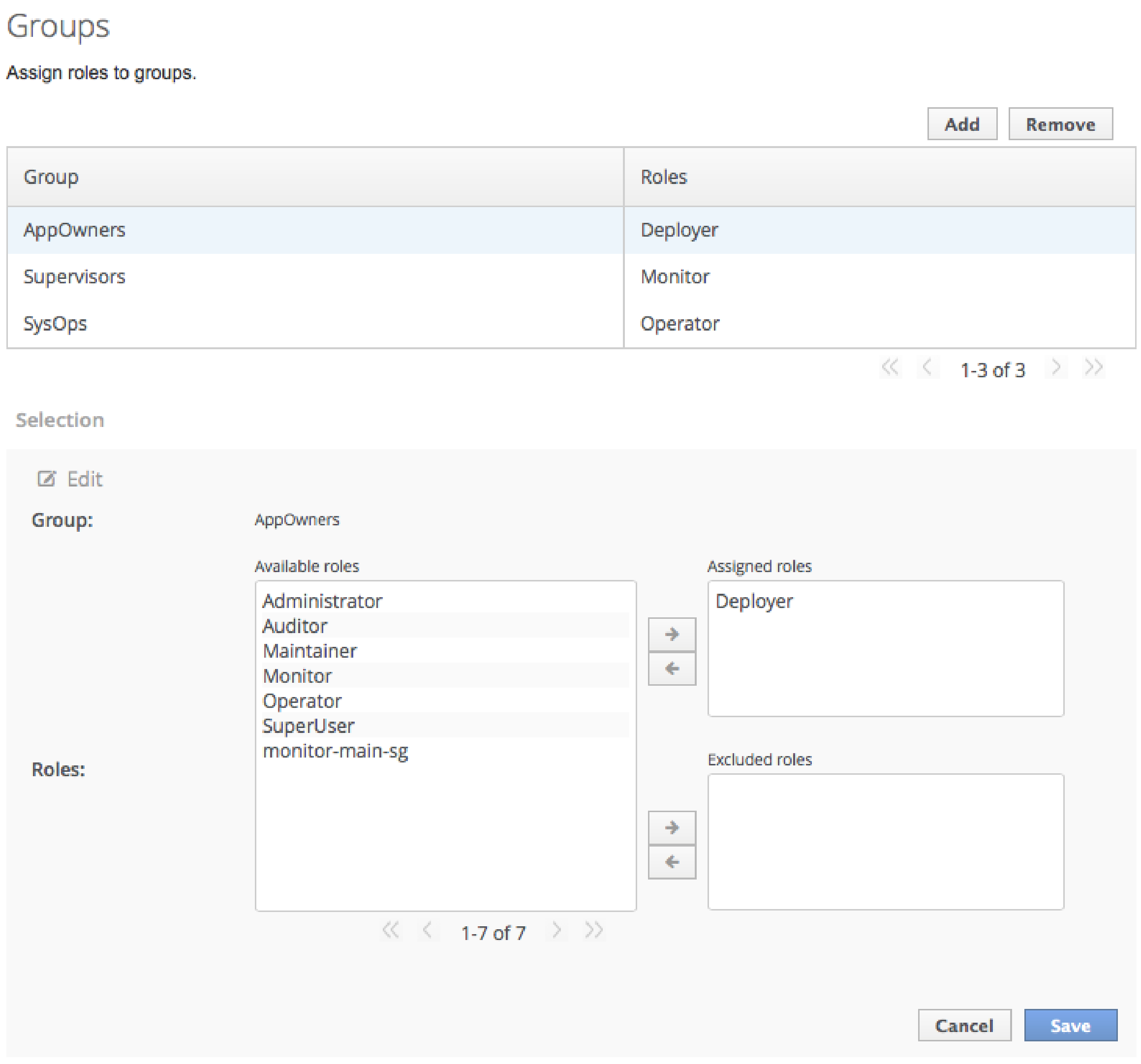
Procedure 11.14. Update a role assignment for a group

1. Login to the Management console.

2. Navigate to the GROUPS tab of the Role Assignment section.

3. Select the group from the list.

4. Click Edit. The Selection view enters Edit mode.


Groups

Assign roles to groups.

Group	Roles
AppOwners	Deployer
Supervisors	Monitor
SysOps	Operator

1-3 of 3

Selection

Group: AppOwners

Roles:

Available roles: Administrator, Auditor, Maintainer, Monitor, Operator, SuperUser, monitor-main-sg

1-7 of 7

Assigned roles: Deployer

Excluded roles:

Figure 11.6. Selection View Edit Mode

Here you can add and remove assigned and excluded roles from the group:

- To add assigned role, select the required role from the list of available roles on the left and click button with the right-facing arrow next to the assigned roles list. The role moves from the available list to the assigned list.
- To remove an assigned role, selected the required role from the assigned roles list on the right and click the button with the left-facing arrow next to the assigned roles list. The role moves from the assigned list to the available list.
- To add an excluded role, select the required role from the list of available roles on the left and click button with the right-facing arrow next to the excluded roles list. The role moves from the available list to the excluded list.
- To remove an excluded role, selected the required role from the excluded roles list on the right and click the button with the left-facing arrow next to the excluded roles list. The role moves from the excluded list to the available list.

5. Click save.

   When successful, the edit view closes, and the list of groups is updated to reflect the changes made. If unsuccessful a "Failed to save role assignment" message is displayed.

Procedure 11.15. Remove role assignment for a group

1. Login to the Management console.
2. Navigate to the GROUPS tab of the Role Assignment section.
3. Select the group from list.
4. Click the Remove button. The Remove Role Assignment confirmation prompt appears.
5. Click the Confirm button.

   When successful, the role will no longer appear in the list of user role assignments.

   Removing the group from the list of role assignments does not remove the user group from the system, nor does it guarantee that no roles will be assigned to members of that group. Each group member might still have a role assigned to them directly.

Report a bug

11.9.6. Configure Group Role Assignment with jboss-cli.sh

Groups to be included or excluded from a role can be configured in the Management Console and the **`jboss-cli.sh`** tool. This topic only shows using the **`jboss-cli.sh`** tool.

The configuration of mapping users and groups to roles is located in the management API at: **`/core-service=management/access=authorization`** as role-mapping elements.

Only users in the SuperUser or Administrator roles can perform this configuration.

Procedure 11.16. Viewing Group Role Assignment Configuration

1. Use the **`read-children-names`** operation to get a complete list of the configured roles:

```
/core-service=management/access=authorization:read-children-
names(child-type=role-mapping)
```

```
[standalone@localhost:9999 access=authorization] :read-children-
names(child-type=role-mapping)
{
    "outcome" => "success",
    "result" => [
        "ADMINISTRATOR",
        "DEPLOYER",
        "MAINTAINER",
        "MONITOR",
        "OPERATOR",
        "SuperUser"
    ]
}
```

2. Use the **`read-resource`** operation of a specified role-mapping to get the full details of a specific role:

```
/core-service=management/access=authorization/role-
mapping=ROLENAME:read-resource(recursive=true)
```

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=ADMINISTRATOR:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "include-all" => false,
        "exclude" => undefined,
        "include" => {
            "user-theboss" => {
                "name" => "theboss",
                "realm" => undefined,
                "type" => "USER"
            },
            "user-harold" => {
                "name" => "harold",
                "realm" => undefined,
                "type" => "USER"
            },
            "group-SysOps" => {
                "name" => "SysOps",
                "realm" => undefined,
                "type" => "GROUP"
            }
        }
    }
}
[standalone@localhost:9999 access=authorization]
```

Procedure 11.17. Add a new role

This procedure shows how to add a role-mapping entry for a role. This must be done before the role can be configured.

- Use the **add** operation to add a new role configuration.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME:add
```

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=AUDITOR:add
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

Procedure 11.18. Add a Group as included in a role

This procedure shows how to add a Group to the included list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be done first.

- Use the **add** operation to add a Group entry to the includes list of the role.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:add(name=GROUPNAME, type=GROUP)
```

ROLENAME is the name of the role being configured.

GROUPNAME is the name of the group being added to the include list.

ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as **group-*GROUPNAME***.

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=AUDITOR/include=group-investigators:add(name=investigators,
type=GROUP)
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

Procedure 11.19. Add a group as excluded in a role

This procedure shows how to add a group to the excluded list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be created first.

- Use the **add** operation to add a group entry to the excludes list of the role.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:add(name=GROUPNAME, type=GROUP)
```

ROLENAME is the name of the role being configured

GROUPNAME is the name of the group being added to the include list

ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as ***group-GROUPNAME***.

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=AUDITOR/exclude=group-supervisors:add(name=supervisors,
type=USER)
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

Procedure 11.20. Remove group role include configuration

This procedure shows how to remove a group include entry from a role mapping.

- Use the **remove** operation to remove the entry.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:remove
```

ROLENAME is the name of the role being configured

ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as ***group-GROUPNAME***.

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=AUDITOR/include=group-investigators:remove
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

Removing the group from the list of includes does not remove the group from the system, nor does it guarantee that the role won't be assigned to users in this group. The role might still be assigned to users in the group individually.

Procedure 11.21. Remove a user group exclude entry

This procedure shows how to remove a group exclude entry from a role mapping.

- Use the **remove** operation to remove the entry.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:remove
```

ROLENAME is the name of the role being configured.

ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as ***group-GROUPNAME***.

```
[standalone@localhost:9999 access=authorization] ./role-
mapping=AUDITOR/exclude=group-supervisors:remove
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

Removing the group from the list of excludes does not remove the group from the system. It also does not guarantee the role will be assigned to members of the group. Roles might still be excluded based on group membership.

Report a bug

11.9.7. About Authorization and Group Loading with LDAP

Within the LDAP directory it is expected that there are entries for the user accounts and that there are entries for the groups, these are then cross referenced by the use of attributes. The attributes used to cross reference between the two could be a reference from the user account over to the group entry or an attribute on the group referencing the users that are members of the group. On some servers both forms of cross reference exist.

It is also common that a user would be authenticating against the server using a simple user name, when it comes to searching for the group membership information depending on the directory server in use searches could be performed using this simple name or it could be performed using the distinguished name of the users entry in the directory.

The authentication step of a user connecting to the server would always happen first, only once it has been decided that the user is successfully authenticated does the server move onto loading a users groups. As the authentication step and the authorization step both use a connection to the LDAP server the realm contains an optimization that any connection used for authentication will be reused for the group loading step. As will be shown within the configuration steps below it is possible to define rules within the authorization section to convert a users simple user name to their distinguished name, this is potentially duplicating a search that would have occurred during the authentication step so if a user name to distinguished name search has already been performed the result of that search is cached and reused without requiring a repeat.

```
<authorization>
    <ldap connection="...">
       <username-to-dn> <!-- OPTIONAL -->
           <!-- Only one of the following. -->
           <username-is-dn />
           <username-filter base-dn="..." recursive="..." user-dn-
attribute="..." attribute="..." />
           <advanced-filter base-dn="..." recursive="..." user-dn-
attribute="..." filter="..." />
        </username-to-dn>
       <group-search group-name="..." iterative="..." group-dn-
attribute="..." group-name-attribute="..." >
           <!-- One of the following -->
           <group-to-principal base-dn="..." recursive="..." search-
by="...">
               <membership-filter principal-attribute="..." />
           </group-to-principal>
           <principal-to-group group-attribute="..." />
       </group-search>
    </ldap>
</authorization>
```

IMPORTANT

Some of these examples specify attributes are using the default values. They are shown here for clarity. Attributes that contain the default values are removed from the configuration when it is persisted by the server.

username-to-dn

As mentioned above there may sometimes be a need to define within the authorization configuration how to map from the user name supplied by the user being authenticated to the distinguished name of their entry within the LDAP directory. The **`username-to-dn`** element is how this is defined, this element is only required if both of the following are true:

- The authentication step was not against LDAP.
- The group searching is using the distinguished name during the searching.

Do try and keep the first bullet point in mind, as you read the examples below this will feel as though the authentication configuration is being duplicated and it is true that it is - if you are only using LDAP for authentication this is not required as the distinguished name will be obtained during authentication.

1:1 username-to-dn

This is the most basic form of the configuration and is used to specify that the user name entered by the remote user is actually the users distinguished name.

```
<username-to-dn>
   <username-is-dn />
</username-to-dn>
```

As this is defining a 1:1 mapping there is no additional configuration possible.

username-filter

The next option is very similar to the simple option described above for the authentication step, quite simply an attribute is specified that is searched for a match against the supplied user name.

```
<username-to-dn>
    <username-filter base-dn="dc=people,dc=harold,dc=example,dc=com"
recursive="false" attribute="sn" user-dn-attribute="dn" />
</username-to-dn>
```

The attributes that can be set here are:

- **`base-dn`**: The distinguished name of the context to begin the search.
- **`recursive`**: Whether the search will extend to sub contexts. Defaults to **`false`**.
- **`attribute`**: The attribute of the users entry to try and match against the supplied user name. Defaults to **`uid`**.
- **`user-dn-attribute`**: The attribute to read to obtain the users distinguished name. Defaults to **`dn`**.

advanced-filter

The final option is to specify an advanced filter, as in the authentication section this is an opportunity to use a custom filter to locate the users distinguished name.

```
<username-to-dn>
    <advanced-filter base-dn="dc=people,dc=harold,dc=example,dc=com"
recursive="false" filter="sAMAccountName={0}" user-dn-attribute="dn" />
</username-to-dn>
```

For the attributes that match the ones in the *username-filter* the meaning and default values are the same so I will not list them here again, this leaves one new attribute:

- **`filter`**: Custom filter used to search for a users entry where the user name will be substituted in the **`{0}`** place holder.

IMPORTANT

The XML must remain valid after the filter is defined so if any special characters are used such as **&** ensure the proper form is used. For example **`&amp;`** for the **&** character.

The Group Search

As described above there are two different styles that can be used when searching for group membership information. The first style is where the user's entry contains an attribute that references the groups the user is a member of. The second style is where the group contains an attribute referencing the users entry.

When there is a choice of which style to use Red Hat recommends that the configuration for a user's entry referencing the group is used. This is because with this method group information can be loaded by reading attributes of known distinguished names without having to perform any searches. The other approach requires extensive searches to identify the groups that reference the user.

Before describing the configuration here are a couple of LDIF examples to illustrate this.

Example 11.1. Principal to Group - LDIF example.

This example illustrates where we have a user **`TestUserOne`** who is a member of **`GroupOne`**, **`GroupOne`** is then in-turn a member of **`GroupFive`**. The group membership is shown by the use of a **`memberOf`** attribute which is set to the distinguished name of the group the user (or group) is a member of.

It is not shown here but a user could potentially have multiple **`memberOf`** attributes set, one for each group the user is directly a member of.

```
dn: uid=TestUserOne,ou=users,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
```

```
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
distinguishedName: uid=TestUserOne,ou=users,dc=principal-to-
group,dc=example,dc=org
memberOf: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
memberOf: uid=Slashy/Group,ou=groups,dc=principal-to-
group,dc=example,dc=org
userPassword::
e1NTSEF9WFpURzhLVjc4WVZBQUJNbEI3Ym96UVAva0RTNlFNWUpLOTdTMUE9PQ==

dn: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupOne
distinguishedName: uid=GroupOne,ou=groups,dc=principal-to-
group,dc=example,dc=org
memberOf: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupFive
distinguishedName: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org
```

Example 11.2. Group to Principal - LDIF Example

This example shows the same user **`TestUserOne`** who is a member of **`GroupOne`** which is in turn a member of **`GroupFive`** - however in this case it is an attribute **`uniqueMember`** from the group to the user being used for the cross reference.

Again the attribute used for the group membership cross reference can be repeated, if you look at *GroupFive* there is also a reference to another user *TestUserFive* which is not shown here.

```
dn: uid=TestUserOne,ou=users,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
userPassword::
```

```
e1NTSEF9SjR0OTRDR1ltaHc1VVZQOEJvbXhUYjl1dkFVd1lQTmRLSEdzaWc9PQ==

dn: uid=GroupOne,ou=groups,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group One
uid: GroupOne
uniqueMember: uid=TestUserOne,ou=users,dc=group-to-
principal,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=group-to-
principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group Five
uid: GroupFive
uniqueMember: uid=TestUserFive,ou=users,dc=group-to-
principal,dc=example,dc=org
uniqueMember: uid=GroupOne,ou=groups,dc=group-to-
principal,dc=example,dc=org
```

General Group Searching

Before looking at the examples for the two approaches shown above we first need to define the attributes common to both of these.

```
<group-search group-name="..." iterative="..." group-dn-attribute="..."
group-name-attribute="..." >
    ...
</group-search>
```

- **`group-name`**: This attribute is used to specify the form that should be used for the group name returned as the list of groups the user is a member of, this can either be the simple form of the group name or the groups distinguished name, if the distinguished name is required this attribute can be set to **`DISTINGUISHED_NAME`**. Defaults to **`SIMPLE`**.

- **`iterative`**: This attribute is used to indicate if after identifying the groups a user is a member of we should also iteratively search based on the groups to identify which groups the groups are a member of. If iterative searching is enabled we keep going until either we reach a group that is not a member if any other groups or a cycle is detected. Defaults to **`false`**.

Cyclic group membership is not a problem. A record of each search is kept to prevent groups that have already been searched from being searched again.

IMPORTANT

For iterative searching to work the group entries need to look the same as user entries, that is the same approach used to identify the groups a user is a member of is then used to identify the groups the group is a member of. This would not be possible if say once we are talking about group to group membership the name of the attribute used for the cross reference changes or if the direction of the reference changes.

- **`group-dn-attribute`**: On an entry for a group which attribute is it's distinguished name. Defaults to **`dn`**.
- **`group-name-attribute`**: On an entry for a group which attribute is it's simple name. Defaults to **`uid`**.

Example 11.3. Principal to Group Example Configuration

Based on the example LDIF from above here is an example configuration iteratively loading a users groups where the attribute used to cross reference is the **`memberOf`** attribute on the user.

```
<authorization>
    <ldap connection="LocalLdap">
        <username-to-dn>
            <username-filter base-dn="ou=users,dc=principal-to-
group,dc=example,dc=org" recursive="false" attribute="uid" user-dn-
attribute="dn" />
        </username-to-dn>
        <group-search group-name="SIMPLE" iterative="true" group-dn-
attribute="dn" group-name-attribute="uid">
            <principal-to-group group-attribute="memberOf" />
        </group-search>
    </ldap>
</authorization>
```

The most important aspect of this configuration is that the **`principal-to-group`** element has been added with a single attribute.

- **`group-attribute`**: The name of the attribute on the user entry that matches the distinguished name of the group the user is a member of. Defaults to **`memberOf`**.

Example 11.4. Group to Principal Example Configuration

This example shows an iterative search for the group to principal LDIF example shown above.

```
<authorization>
      <ldap connection="LocalLdap">
          <username-to-dn>
              <username-filter base-dn="ou=users,dc=group-to-
principal,dc=example,dc=org" recursive="false" attribute="uid" user-dn-
attribute="dn" />
          </username-to-dn>
          <group-search group-name="SIMPLE" iterative="true" group-dn-
attribute="dn" group-name-attribute="uid">
              <group-to-principal base-dn="ou=groups,dc=group-to-
principal,dc=example,dc=org" recursive="true" search-
by="DISTINGUISHED_NAME">
                  <membership-filter principal-attribute="uniqueMember"
/>
              </group-to-principal>
          </group-search>
      </ldap>
  </authorization>
```

Here an element **`group-to-principal`** is added, this element is used to define how searches for groups that reference the user entry will be performed, the following attributes are set:

- **`base-dn`**: The distinguished name of the context to use to begin the search.
- **`recursive`**: Whether sub-contexts also be searched. Defaults to **`false`**.
- **`search-by`**: The form of the role name used in searches. Valid valids are **`SIMPLE`** and **`DISTINGUISHED_NAME`**. Defaults to **`DISTINGUISHED_NAME`**.

Within the *group-to-principal* element there is a *membership-filter* element to define the cross reference.

- **`principal-attribute`**: The name of the attribute on the group entry that references the user entry. Defaults to **`member`**.

Report a bug

11.9.8. About Scoped Roles

Scoped Roles are user-defined roles that grant the permissions of one of the standard roles but only for one or more specified server groups or hosts. Scoped roles allow for management users to be granted permissions that are limited to only those server groups or hosts that are required.

Scoped roles can be created by users assigned the Administrator or SuperUser roles.

They are defined by five characteristics:

1. A unique name.
2. Which of the standard roles it is based on.
3. If it applies to Server Groups or Hosts
4. The list of server groups or hosts that it is restricted to.
5. If all users are automatically include. This defaults to false.

Once created a scoped role can be assigned to users and groups the same way that the standard roles are.

Creating a scoped role does not let you define new permissions. Scoped roles can only be used to apply the permissions of an existing role in a limited scope. For example, you could create a scoped role based on the Deployer role which is restricted to a single server group.

There are only two scopes that roles can be limited to, host and server group.

Host-scoped roles

A role that is host-scoped restricts the permissions of that role to one or more hosts. This means access is provided to the relevant **`/host=*/`** resource trees but resources that are specific to other hosts are hidden.

Server-Group-scoped roles

A role that is server-group-scoped restricts the permissions of that role to one or more server groups. Additionally the role permissions will also apply to the profile, socket binding group, server config and server resources that are associated with the specified server-groups. Any sub-resources within any of those that are not logically related to the server-group will not be visible to the user.

Both host and server-group scoped roles have permissions of the Monitor role for the remainder of the managed domain configuration.

Report a bug

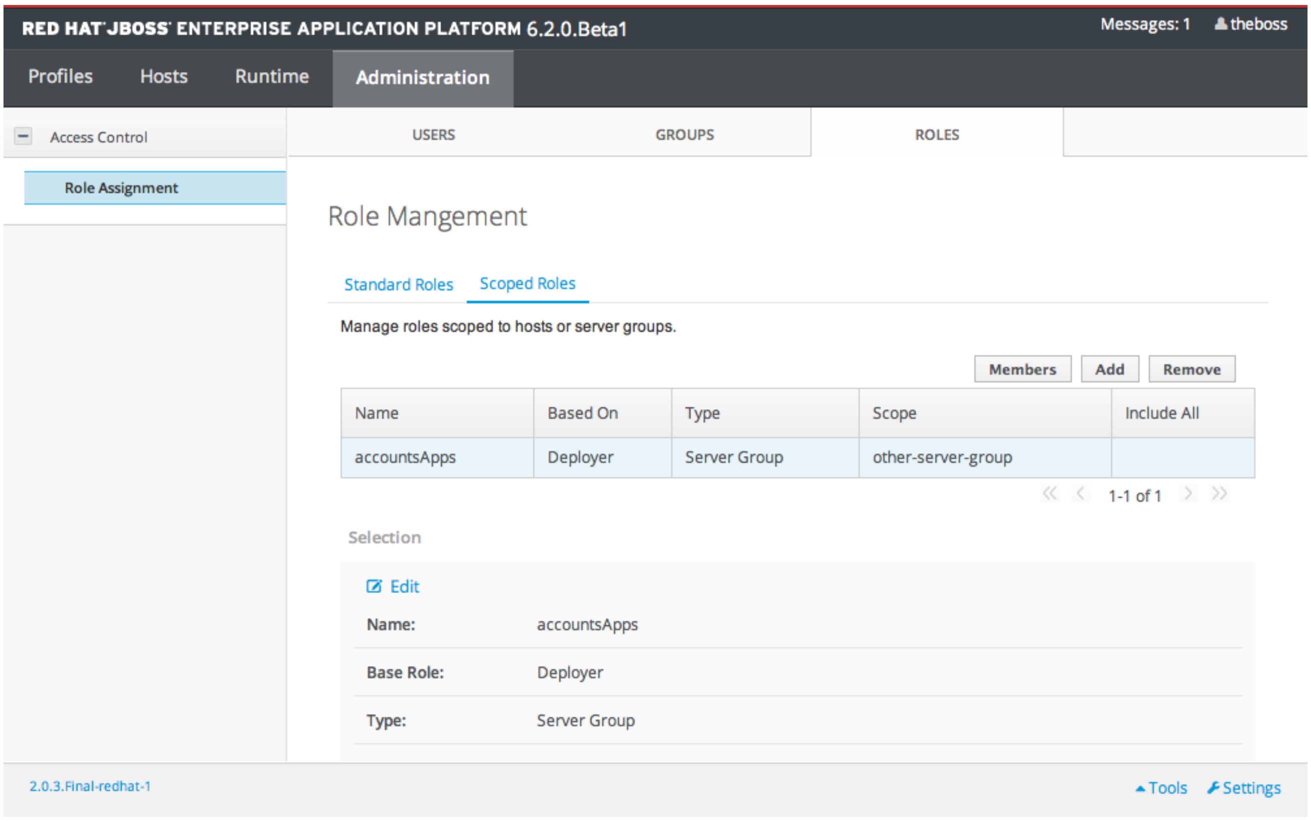
11.9.9. Creating Scoped Roles

Scoped Roles are user-defined roles that grant the permissions of one of the standard roles but only for one or more specified server groups or hosts. This topic shows how to create scoped roles.

Only users in the SuperUser or Administrator roles can perform this configuration.

Scoped Role configuration in the management console can be found by following these steps:

1. Login to the Management Console
2. Click on the Administration tab
3. Expand the Access Control item on the left and select Role Assignment
4. Select ROLES tab, and then the Scoped Roles tab within it.



Manage roles scoped to hosts or server groups.

Name	Based On	Type	Scope	Include All
accountsApps	Deployer	Server Group	other-server-group	

Name:	accountsApps
Base Role:	Deployer
Type:	Server Group

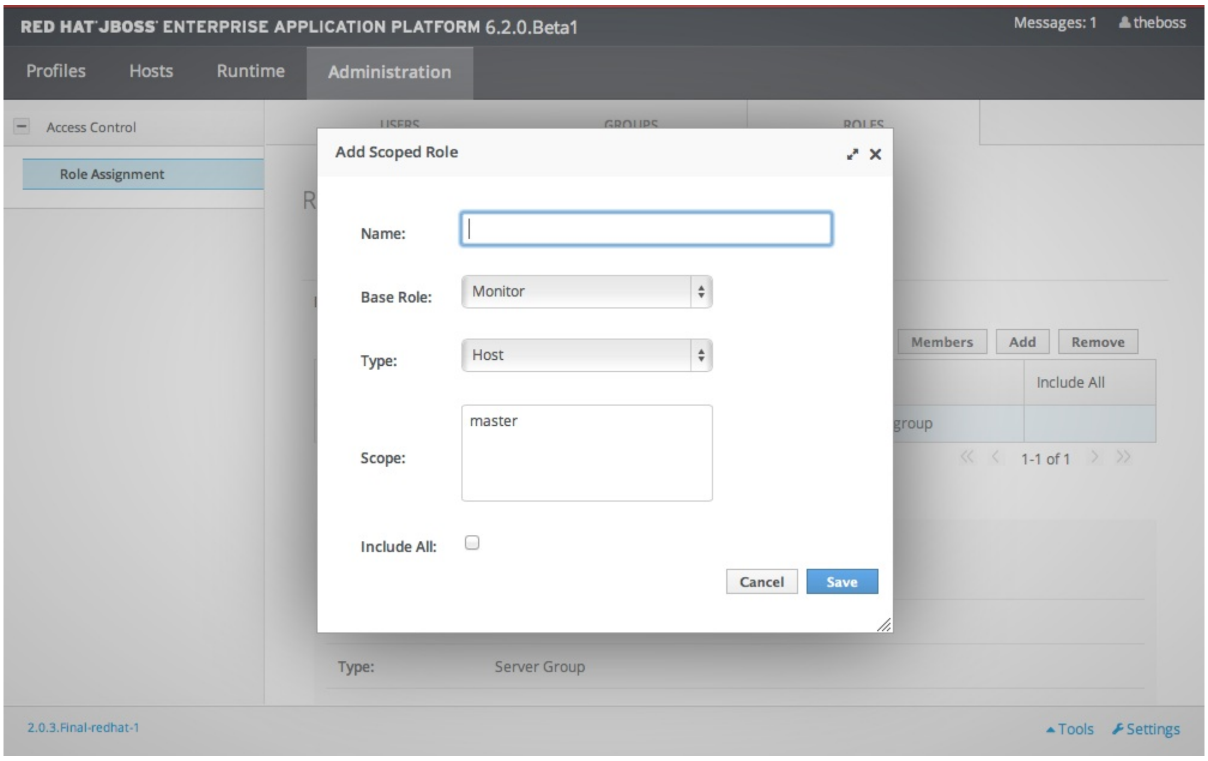
Figure 11.7. Scoped Role Configuration in the Management Console

The Scoped Roles section of the Management Console consists of two main areas, a table containing a list of the currently configured scoped roles, and the Selection panel which displays the details of the role currently selected in the table.

The following procedures show how to perform configuration tasks for Scoped Roles.

Procedure 11.22. Add a New Scoped Role

1. Login to the Management Console
2. Navigate tot he Scoped Roles area of the Roles tab.
3. Click the Add button. The Add Scoped Role dialog appears.



Name:
Base Role: Monitor
Type: Host
Scope: master
Include All: ☐

Type: Server Group

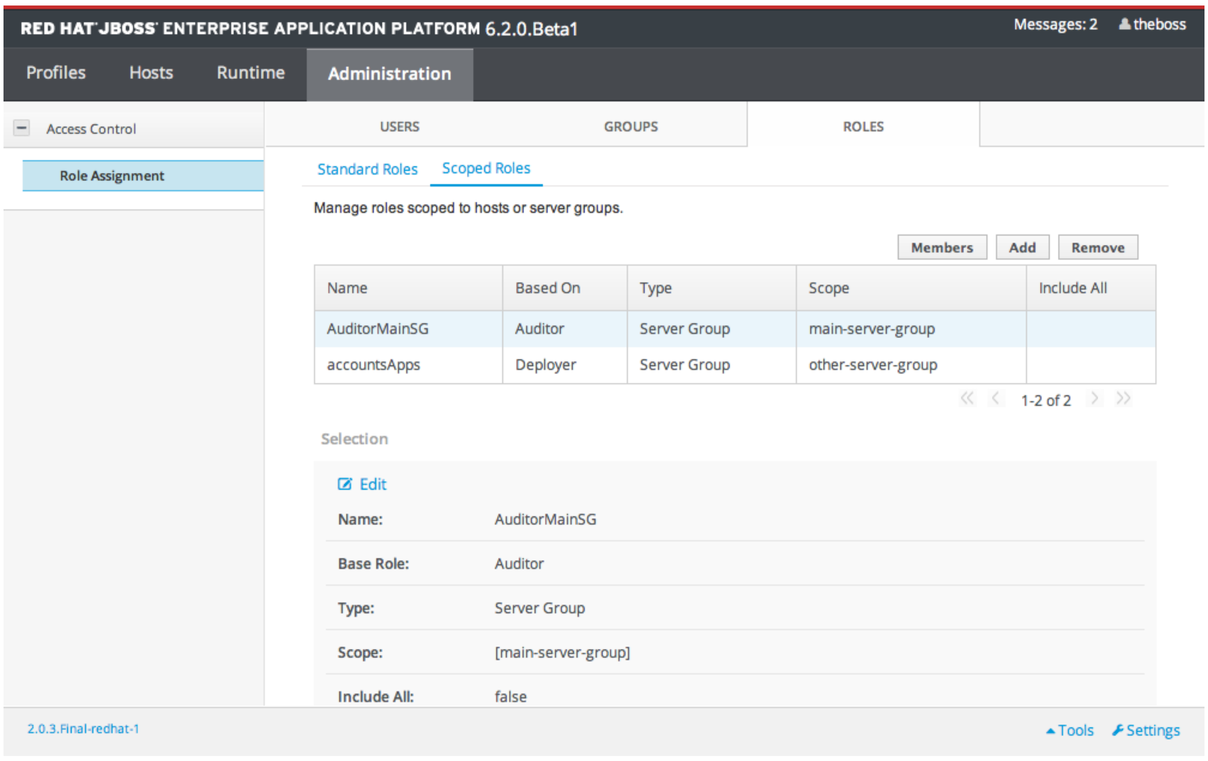
Figure 11.8. Add Scoped Role Dialog

4. Specify the following details:
   - **`Name`**, the unique name for the new scoped role.
   - **`Base Role`**, the role which this role will base its permissions on.
   - **`Type`**, whether this role will be restricted to hosts or server groups.
   - **`Scope`**, the list of hosts or server groups that the role is restricted to. Multiple entries can be selected.
   - **`Include All`**, should this role automatically include all users. Defaults to no.
5. Click the Save button and the dialog will close and the newly created role will appear in the table.

Procedure 11.23. Edit a Scoped Role

1. Login to the Management Console
2. Navigate to the Scoped Roles area of the Roles tab.

3. Click on the scoped role you want to edit in the table. The details of that role appears in the Selection panel below the table.



RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6.2.0.Beta1 — Messages: 2 — theboss

Profiles | Hosts | Runtime | Administration

Access Control — Role Assignment

USERS | GROUPS | ROLES

Standard Roles | Scoped Roles

Manage roles scoped to hosts or server groups.

Members | Add | Remove

Name	Based On	Type	Scope	Include All
AuditorMainSG	Auditor	Server Group	main-server-group	
accountsApps	Deployer	Server Group	other-server-group	

1-2 of 2

Selection

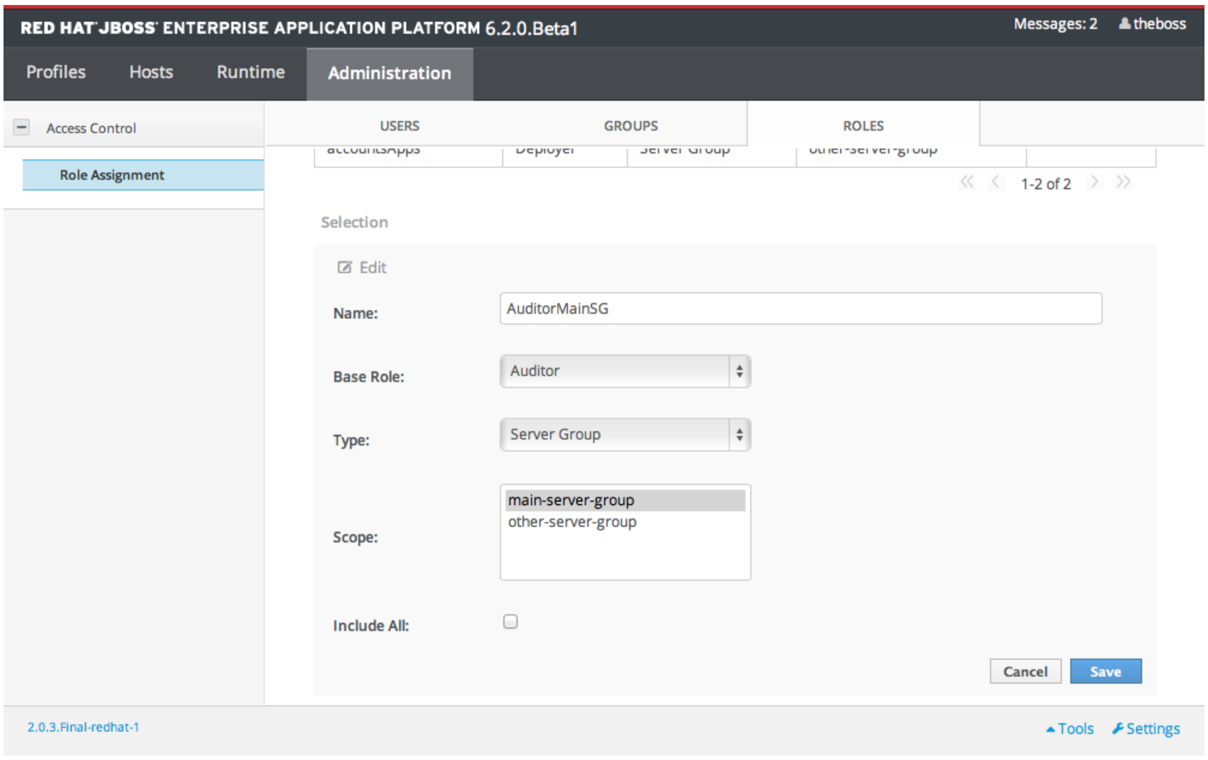
Edit

Name:	AuditorMainSG
Base Role:	Auditor
Type:	Server Group
Scope:	[main-server-group]
Include All:	false

2.0.3.Final-redhat-1 — Tools — Settings

Figure 11.9. Role Selected

4. Click the Edit link in the Selection panel. The Selection panel enters edit mode.



RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6.2.0.Beta1 — Messages: 2 — theboss

Profiles | Hosts | Runtime | Administration

Access Control — Role Assignment

USERS | GROUPS | ROLES

1-2 of 2

Selection

Edit

Name:	AuditorMainSG
Base Role:	Auditor
Type:	Server Group
Scope:	main-server-group (selected), other-server-group
Include All:	☐

Cancel | Save

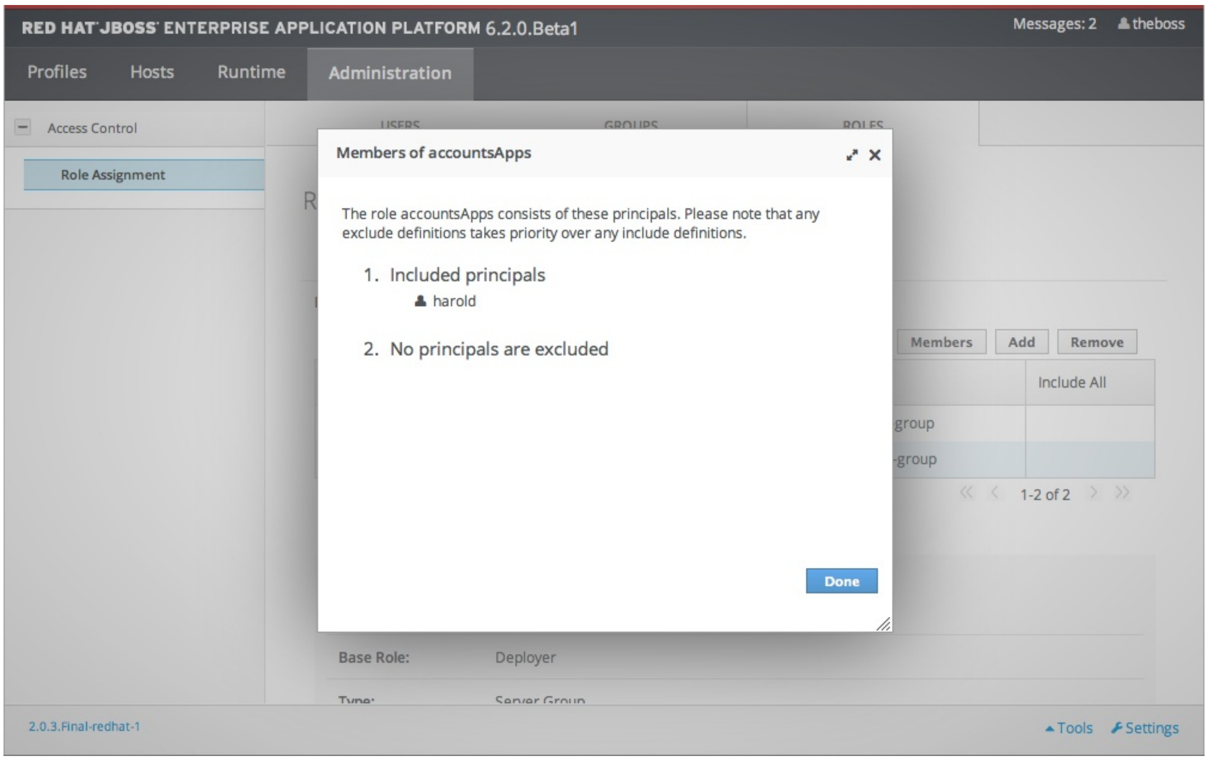
2.0.3.Final-redhat-1 — Tools — Settings

Figure 11.10. Selection Panel in Edit Mode

5. Update the details you need to change and click the Save button. The Selection panel returns to it's previous state. Both the Selection panel and table show the newly updated details.

Procedure 11.24. View Scoped Role Members

1. Login to the Management Console
2. Navigate to the Scoped Roles area of the Roles tab.
3. Click on the scoped role in the table that you want to view the Members of, then click the Members button. The Members of role dialog appears. It shows users and groups that are included or excluded from the role.


Members of accountsApps

The role accountsApps consists of these principals. Please note that any exclude definitions takes priority over any include definitions.

1. Included principals
   - harold
2. No principals are excluded

Figure 11.11. Role Membership Dialog

4. Click the Done button when you have finished reviewing this information.

Procedure 11.25. Delete a Scoped Role

IMPORTANT

A Scoped Role cannot be deleted if users or groups are assigned to it. Remove the role assignments first, and then delete it.

1. Login to the Management Console
2. Navigate to the Scoped Roles area of the Roles tab.
3. Select the scoped role to be removed in the table.
4. Click the Remove button. The Remove Scoped Role dialog appears.
5. Click the Confirm button.The dialog closes and the role is removed.

Report a bug

11.10. CONFIGURING CONSTRAINTS

11.10.1. Configure Sensitivity Constraints

Each Sensitivity Constraint defines a set of resources that are considered "sensitive". A sensitive resource is generally one that either should be secret, like passwords, or one that will have serious impact on the server, like networking, JVM configuration, or system properties. The access control system itself is also considered sensitive. Resource sensitivity limits which roles are able to read, write or address a specific resource.

Sensitivity constraint configuration is in the Management API at **`/core-service=management/access=authorization/constraint=sensitivity-classification`**.

Within the management model each Sensitivity Constraint is identified as a **`classification`**. The classifications are then grouped into **`types`**. There are 39 included classifications that are arranged into 13 types.

To configure a a sensitivity constraint, use the **`write-attribute`** operation to set the **`configured-requires-read`**, **`configured-requires-write`**, or **`configured-requires-addressable`** attribute. To make that type of operation sensitive set the value of the attribute to **`true`**, otherwise to make it nonsensitive set it to **`false`**. By default these attributes are not set and the values of **`default-requires-read`**, **`default-requires-write`**, and **`default-requires-addressable`** are used. Once the configured attribute is set it is that value that is used instead of the default. The default values cannot be changed.

Example 11.5. Make reading system properties a sensitive operation

```
[domain@localhost:9999 /] cd /core-
service=management/access=authorization/constraint=sensitivity-
classification/type=core/classification=system-property
[domain@localhost:9999 classification=system-property] :write-
attribute(name=configured-requires-read, value=true)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => {"main-server-group" => {"host" => {"master" => {
        "server-one" => {"response" => {"outcome" => "success"}},
        "server-two" => {"response" => {"outcome" => "success"}}
    }}}}
}
[domain@localhost:9999 classification=system-property] :read-resource
{
    "outcome" => "success",
    "result" => {
        "configured-requires-addressable" => undefined,
        "configured-requires-read" => true,
        "configured-requires-write" => undefined,
        "default-requires-addressable" => false,
        "default-requires-read" => false,
        "default-requires-write" => true,
        "applies-to" => {
            "/host=master/system-property=*" => undefined,
            "/host=master/core-service=platform-mbean/type=runtime" =>
undefined,
            "/server-group=*/system-property=*" => undefined,
            "/host=master/server-config=*/system-property=*" =>
undefined,
            "/host=master" => undefined,
            "/system-property=*" => undefined,
            "/" => undefined
        }
    }
}
[domain@localhost:9999 classification=system-property]
```

What roles will be able to perform what operations depending on the configuration of these attributes is summarized in Table 11.2, “Sensitivity Constraint Configuration outcomes”.

Table 11.2. Sensitivity Constraint Configuration outcomes

Value	`requires-read`	`requires-write`	`requires-addressable`
`true`	Read is sensitive. Only Auditor, Administrator, SuperUser can read.	Write is sensitive. Only Administrator and SuperUser can write	Addressing is sensitive. Only Auditor, Administrator, SuperUser can address.
`false`	Read is not sensitive. Any management user can read.	Write is not sensitive. Only Maintainer, Administrator and SuperUser can write. Deployers can also write the resource is an application resource.	Addressing is not sensitive. Any management user can address.

Report a bug

11.10.2. Configure Application Resource Constraints

Each Application Resource Constraint defines a set of resources, attributes and operations that are usually associated with the deployment of applications and services. When an application resource constraint is enabled management users of the Deployer role are granted access to the resources that it applies to.

Application constraint configuration is in the Management Model at **`/core-service=management/access=authorization/constraint=application-classification/`**.

Within the management model each Application Resource Constraint is identified as a **`classification`**. The classifications are then grouped into **`types`**. There are 14 included classifications that are arranged into 8 types. Each classification has an **`applies-to`** element which is a list of resource path patterns to which the classifications configuration applies.

By default the only Application Resource classification that is enabled is **`core`**. Core includes deployments, deployment overlays, and the deployment operations.

To enable an Application Resource, use the **`write-attribute`** operation to set the **`configured-application attribute`** of the classification to **`true`**. To disable an Application Resource, set this attribute to **`false`**. By default these attributes are not set and the value of **`default-application attribute`** is used. The default value cannot be changed.

Example 11.6. Enabling the logger-profile application resource classification

```
[domain@localhost:9999 /] cd /core-
service=management/access=authorization/constraint=application-
classification/type=logging/classification=logging-profile
[domain@localhost:9999 classification=logging-profile] :write-
attribute(name=configured-application, value=true)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => {"main-server-group" => {"host" => {"master" => {
        "server-one" => {"response" => {"outcome" => "success"}},
        "server-two" => {"response" => {"outcome" => "success"}}
    }}}}
}
[domain@localhost:9999 classification=logging-profile] :read-resource
{
    "outcome" => "success",
    "result" => {
        "configured-application" => true,
        "default-application" => false,
        "applies-to" => {"/profile=*/subsystem=logging/logging-
profile=*" => undefined}
    }
}
[domain@localhost:9999 classification=logging-profile]
```

IMPORTANT

Application Resource Constraints apply to all resources that match its configuration. For example, It is not possible to grant a Deployer user access to one datasource resource but not another. If this level of separation is required then it is recommended to configure the resources in different server groups and create different scoped Deployer roles for each group.

Report a bug

11.10.3. Configure the Vault Expression Constraint

By default, reading and writing vault expressions are sensitive operations. Configuring the Vault Expression Constraint allows you to set either or both of those operations to being nonsensitive. Changing this constraint allows a greater number of roles to read and write vault expressions.

The vault expression constraint is found in the management model at **`/core-service=management/access=authorization/constraint=vault-expression`**.

To configure the vault expression constraint, use the **`write-attribute`** operation to set the attributes of **`configured-requires-write`** and **`configured-requires-read`** to **`true`** or **`false`**. By default these are not set and the values of **`default-requires-read`** and **`default-requires-write`** are used. The default values cannot be changed.

Example 11.7. Making writing to vault expressions a nonsensitive operation

```
[domain@localhost:9999 /] cd /core-
service=management/access=authorization/constraint=vault-expression
[domain@localhost:9999 constraint=vault-expression] :write-
attribute(name=configured-requires-write, value=false)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => {"main-server-group" => {"host" => {"master" => {
        "server-one" => {"response" => {"outcome" => "success"}},
        "server-two" => {"response" => {"outcome" => "success"}}
    }}}}
}
[domain@localhost:9999 constraint=vault-expression] :read-resource
{
    "outcome" => "success",
    "result" => {
        "configured-requires-read" => undefined,
        "configured-requires-write" => false,
        "default-requires-read" => true,
        "default-requires-write" => true
    }
}
[domain@localhost:9999 constraint=vault-expression]
```

What roles will be able to read and write to vault expressions depending on this configuration is summarized in Table 11.3, “Vault Expression Constraint Configuration outcomes”.

Table 11.3. Vault Expression Constraint Configuration outcomes

Value	`requires-read`	`requires-write`
`true`	Read operation is sensitive. Only Auditor, Administrator, and SuperUser can read.	Write operation is sensitive. Only Administrator and SuperUser can write.
`false`	Read operation is not sensitive. All management users can read.	Write operation is not sensitive. Monitor, Administrator and SuperUser can write. Deployers can also write if the vault expression is in an Application Resource.

Report a bug

11.11. CONSTRAINTS REFERENCE

11.11.1. Application Resource Constraints Reference

Type: core

Classification: deployment-overlay

- default: true

PATH	Attributes	Operations
/deployment-overlay=*		
/deployment=*		
/		upload-deployment-stream, full-replace-deployment, upload-deployment-url, upload-deployment-bytes

Type: datasources

Classification: datasource

- default: false

PATH	Attributes	Operations
/deployment=*/subdeployment=*/subsystem=datasources/data-source=*		
/subsystem=datasources/data-source=*		
/subsystem=datasources/data-source=ExampleDS		
/deployment=*/subsystem=datasources/data-source=*		

Classification: jdbc-driver

- default: false

PATH	Attributes	Operations
/subsystem=datasources/jdbc-driver=*		

Classification: xa-data-source

- default: false

PATH	Attributes	Operations
/subsystem=datasources/xa-data-source=*		
/deployment=*/subsystem=datasources/xa-data-source=*		
/deployment=*/subdeployment=*/subsystem=datasources/xa-data-source=*		

Type: logging

Classification: logger

- default: false

PATH	Attributes	Operations
/subsystem=logging/logger=*		
/subsystem=logging/logging-profile=*/logger=*		

Classification: logging-profile

- default: false

PATH	Attributes	Operations
/subsystem=logging/logging-profile=*		

Type: mail

Classification: mail-session

- default: false

PATH	Attributes	Operations
/subsystem=mail/mail-session=*		

Type: naming

Classification: binding

- default: false

PATH	Attributes	Operations
/subsystem=naming/binding=*		

Type: resource-adapters

Classification: resource-adapters

- default: false

PATH	Attributes	Operations
/subsystem=resource-adapters/resource-adapter=*		

Type: security

Classification: security-domain

- default: false

PATH	Attributes	Operations
/subsystem=security/security-domain=*		

Report a bug

11.11.2. Sensitivity Constraints Reference

Type: core

Classification: access-control

- requires-addressable: true
- requires-read: true
- requires-write: true

PATH	attributes	operations
/core-service=management/access=authorization		
/subsystem=jmx	non-core-mbean-sensitivity	

Classification: credential

- requires-addressable: false
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=mail/mail-session=*/server=pop3	username , password	
/subsystem=mail/mail-session=*/server=imap	username, password	
/subsystem=datasources/xa-data-source=*	user-name, recovery-username, password, recovery-password	
/subsystem=mail/mail-session=*/custom=*	username, password	
/subsystem=datasources/data-source=*"	user-name, password	
/subsystem=remoting/remote-outbound-connection=*"	username	
/subsystem=mail/mail-session=*/server=smtp	username , password	

PATH	attributes	operations
/subsystem=web/connector=*/configuration=ssl	key-alias, password	
/subsystem=resource-adapters/resource-adapter=*/connection-definitions=*"	recovery-username, recovery-password	

Classification: domain-controller

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations

Classification: domain-names

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations

Classification: extensions

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/extension=*		

Classification: jvm

- requires-addressable: false

.

- requires-read: false
- requires-write: true

PATH	attributes	operations
/core-service=platform-mbean/type=runtime	input-arguments, boot-class-path, class-path, boot-class-path-supported, library-path	

Classification: management-interfaces

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/core-service=management/management-interface=native-interface		
/core-service=management/management-interface=http-interface		

Classification: module-loading

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/core-service=module-loading		

Classification: patching

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/core-service=patching/addon=*		
/core-service=patching/layer="*"		
/core-service=patching		

Classification: read-whole-config

- requires-addressable: false
- requires-read: true
- requires-write: true

PATH	attributes	operations
/		read-config-as-xml

Classification: security-domain

- requires-addressable: true
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=security/security-domain=*		

Classification: security-domain-ref

- requires-addressable: true
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=datasources/xa-data-source=*	security-domain	

PATH	attributes	operations
/subsystem=datasources/data-source=*	security-domain	
/subsystem=ejb3	default-security-domain	
/subsystem=resource-adapters/resource-adapter=*/connection-definitions=*	security-domain, recovery-security-domain, security-application, security-domain-and-application	

Classification: security-realm

- requires-addressable: true
- requires-read: true
- requires-write: true

PATH	attributes	operations
/core-service=management/security-realm=*		

Classification: security-realm-ref

- requires-addressable: true
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=remoting/connector=*	security-realm	
/core-service=management/management-interface=native-interface	security-realm	
/core-service=management/management-interface=http-interface	security-realm	

PATH	attributes	operations
/subsystem=remoting/remote-outbound-connection=*	security-realm	

Classification: security-vault

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/core-service=vault		

Classification: service-container

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/core-service=service-container		

Classification: snapshots

- requires-addressable: false
- requires-read: false
- requires-write: false

PATH	attributes	operations
/		take-snapshot, list-snapshots, delete-snapshot

Classification: socket-binding-ref

- requires-addressable: false

- requires-addressable: false
- requires-read: false
- requires-write: false

PATH	attributes	operations
/subsystem=mail/mail-session=*/server=pop3	outbound-socket-binding-ref	
/subsystem=mail/mail-session=*/server=imap	outbound-socket-binding-ref	
/subsystem=remoting/connector=*	socket-binding	
/subsystem=web/connector=*	socket-binding	
/subsystem=remoting/local-outbound-connection=*	outbound-socket-binding-ref	
/socket-binding-group=*/local-destination-outbound-socket-binding=*	socket-binding-ref	
/subsystem=remoting/remote-outbound-connection=*	outbound-socket-binding-ref	
/subsystem=mail/mail-session=*/server=smtp	outbound-socket-binding-ref	
/subsystem=transactions	process-id-socket-binding, status-socket-binding, socket-binding	

Classification: socket-config

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/interface=*		resolve-internet-address
/core-service=management/management-interface=native-interface	port, interface, socket-binding	
/socket-binding-group=*		
/core-service=management/management-interface=http-interface	port, secure-port, interface, secure-socket-binding, socket-binding	
/		resolve-internet-address
/subsystem=transactions	process-id-socket-max-ports	

Classification: system-property

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/core-service=platform-mbean/type=runtime	system-properties	
/system-property=*		
/		resolve-expression

Type: datasources

Classification: data-source-security

- requires-addressable: false

- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=datasources/xa-data-source=*	user-name, security-domain, password	
/subsystem=datasources/data-source=*	user-name, security-domain, password	

Type: jdr

Classification: jdr

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/subsystem=jdr		generate-jdr-report

Type: jmx

Classification: jmx

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/subsystem=jmx		

Type: mail

Classification: mail-server-security

- requires-addressable: false
- requires-read: false
- requires-write: true

PATH	attributes	operations
/subsystem=mail/mail-session=*/server=pop3	username, tls, ssl, password	
/subsystem=mail/mail-session=*/server=imap	username, tls, ssl, password	
/subsystem=mail/mail-session=*/custom=*	username, tls, ssl, password	
/subsystem=mail/mail-session=*/server=smtp	username, tls, ssl, password	

Type: naming

Classification: jndi-view

- requires-addressable: false
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=naming		jndi-view

Classification: naming-binding

- requires-addressable: false
- requires-read: false
- requires-write: false

PATH	attributes	operations
/subsystem=naming/binding=*		

Type: remoting

Classification: remoting-security

- requires-addressable: false
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=remoting/connector=*	authentication-provider, security-realm	
/subsystem=remoting/remote-outbound-connection=*	username, security-realm	
/subsystem=remoting/connector=*/security=sasl		

Type: resource-adapters

Classification: resource-adapter-security

- requires-addressable: false
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=resource-adapters/resource-adapter=*/connection-definitions=*	security-domain, recovery-username, recovery-security-domain, security-application, security-domain-and-application, recovery-password	

Type: security

Classification: misc-security

- requires-addressable: false

- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=security	deep-copy-subject-mode	

Type: web

Classification: web-access-log

- requires-addressable: false
- requires-read: false
- requires-write: false

PATH	attributes	operations
/subsystem=web/virtual-server=*/configuration=access-log		

Classification: web-connector

- requires-addressable: false
- requires-read: false
- requires-write: false

PATH	attributes	operations
/subsystem=web/connector=*		

Classification: web-ssl

- requires-addressable: false
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=web/connector=*/configuration=ssl		

Classification: web-sso

- requires-addressable: false
- requires-read: true
- requires-write: true

PATH	attributes	operations
/subsystem=web/virtual-server=*/configuration=sso		

Classification: web-valve

- requires-addressable: false
- requires-read: false
- requires-write: false

PATH	attributes	operations
/subsystem=web/valve=*		

Report a bug

CHAPTER 12. WEB, HTTP CONNECTORS, AND HTTP CLUSTERING

12.1. CONFIGURE A MOD_CLUSTER WORKER NODE

Summary

A mod_cluster worker node consists of an JBoss EAP server. This server can be part of a server group in a Managed Domain, or a standalone server. A separate process runs within JBoss EAP, which manages all of the nodes of the cluster. This is called the master. For more conceptual information about worker nodes, refer to *Worker Node* in the *Red Hat JBoss Enterprise Application Platform 6.1 Administration and Configuration Guide*. For an overview of HTTPD load balancing, refer to *Overview of HTTP Connectors* in the *Administration and Configuration Guide*.

The master is only configured once, via the **`mod_cluster`** subsystem. To configure the **`mod_cluster`** subsystem, refer to *Configure the mod_cluster Subsystem* in the *Administration and Configuration Guide*. Each worker node is configured separately, so repeat this procedure for each node you wish to add to the cluster.

If you use a managed domain, each server in a server group is a worker node which shares an identical configuration. Therefore, configuration is done to an entire server group. In a standalone server, configuration is done to a single JBoss EAP 6 instance. The configuration steps are otherwise identical.

Worker Node Configuration

- If you use a standalone server, it must be started with the **`standalone-ha`** profile.
- If you use a managed domain, your server group must use the **`ha`** or **`full-ha`** profile, and the **`ha-sockets`** or **`full-ha-sockets`** socket binding group. JBoss EAP 6 ships with a cluster-enabled server group called **`other-server-group`** which meets these requirements.

NOTE

Where Management CLI commands are given, they assume you use a managed domain. If you use a standalone server, remove the **`/profile=full-ha`** portion of the commands.

Procedure 12.1. Configure a Worker Node

1. **Configure the network interfaces.**
   By default, the network interfaces all default to **`127.0.0.1`**. Every physical host which hosts either a standalone server or one or more servers in a server group needs its interfaces to be configured to use its public IP address, which the other servers can see.

   To change the IP address of a JBoss EAP 6 host, you need to shut it down and edit its configuration file directly. This is because the Management API which drives the Management Console and Management CLI relies on a stable management address.

   Follow these steps to change the IP address on each server in your cluster to the master's public IP address.

   a. Shut down the server completely.

b. Edit either the **host.xml**, which is in ***EAP_HOME*/domain/configuration/** for a managed domain, or the **standalone-ha.xml** file, which is in ***EAP_HOME*/standalone/configuration/** for a standalone server.

c. Locate the **<interfaces>** element. Three interfaces are configured, **management**, **public**, and **unsecured**. For each of these, change the value **127.0.0.1** to the external IP address of the host.

d. For hosts that participate in a managed domain but are not the master, locate the **<host** element. Note that it does not have the closing **>** symbol, because it contains attributes. Change the value of its name attribute from **master** to a unique name, a different one per slave. This name will also be used for the slave to identify to the cluster, so make a note of it.

e. For newly-configured hosts which need to join a managed domain, find the **<domain-controller>** element. Comment out or remove the **<local />** element, and add the following line, changing the IP address (**X.X.X.X**) to the address of the domain controller. This step does not apply for a standalone server.

```
<remote host="X.X.X.X" port="${jboss.domain.master.port:9999}"
security-realm="ManagementRealm"/>
```

f. Save the file and exit.

2. **Configure authentication for each slave server.**
Each slave server needs a username and password created in the domain controller's or standalone master's **ManagementRealm**. On the domain controller or standalone master, run the ***EAP_HOME*/bin/add-user.sh** command. Add a user with the same username as the slave, to the **ManagementRealm**. When asked if this user will need to authenticate to an external JBoss AS instance, answer **yes**. An example of the input and output of the command is below, for a slave called **slave1**, with password **changeme**.

```
user:bin user$ ./add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a

Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : slave1
Password : changeme
Re-enter Password : changeme
About to add user 'slave1' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'slave1' to file '/home/user/jboss-eap-
6.0/standalone/configuration/mgmt-users.properties'
Added user 'slave1' to file '/home/user/jboss-eap-
6.0/domain/configuration/mgmt-users.properties'
Is this new user going to be used for one AS process to connect to
another AS process e.g. slave domain controller?
yes/no? yes
To represent the user add the following to the server-identities
definition <secret value="Y2hhbmdlbWU=" />
```

■

3. **Copy the Base64-encoded `<secret>` element from the `add-user.sh` output.**
   If you plan to specify the Base64-encoded password value for authentication, copy the **`<secret>`** element value from the last line of the **`add-user.sh`** output as you will need it in the step below.

4. **Modify the slave host's security realm to use the new authentication.**

   a. Re-open the slave host's **`host.xml`** or **`standalone-ha.xml`** file.

   b. Locate the **`<security-realms>`** element. This is where you configure the security realm.

   c. You can specify the secret value in one of the following ways:

      - **Specify the Base64-encoded password value in the configuration file.**

        i. Add the following block of XML code directly below the **`<security-realm name="ManagementRealm">`** line,

        ```
        <server-identities>
            <secret value="Y2hhbmdlbWU="/>
        </server-identities>
        ```

        ii. Replace the "Y2hhbmdlbWU=" with the secret value returned from the **`add-user.sh`** output in the previous step.

      - **Configure the host to get the password from the vault.**

        i. Use the **`vault.sh`** script to generate a masked password. It will generate a string like the following:
        `VAULT::secret::password::ODVmYmJjNGMtZDU2ZC00YmNlLWE4ODMtZjQ1NWNmNDU4ZDc1TElORV9CUkVBS3ZhdWx0`.

        You can find more information on the vault in the Password Vaults for Sensitive Strings section of this guide starting here: Section 10.11.1, “About Securing Sensitive Strings in Clear-Text Files”.

        ii. Add the following block of XML code directly below the **`<security-realm name="ManagementRealm">`** line.

        ```
        <server-identities>
            <secret
        value="${VAULT::secret::password::ODVmYmJjNGMtZDU2ZC00YmNlL
        WE4ODMtZjQ1NWNmNDU4ZDc1TElORV9CUkVBS3ZhdWx0}"/>
        </server-identities>
        ```

        Be sure to replace the secret value with the masked password generated in the previous step.

NOTE

When creating a password in the vault, it must be specified in plain text, not Base64-encoded.

- **Specify the password as a system property.**

  i. Add the following block of XML code directly below the **`<security-realm name="ManagementRealm">`** line

  ```
  <server-identities>
      <secret value="${server.identity.password}"/>
  </server-identities>
  ```

  ii. When you specify the password as a system property, you can configure the host in either of the following ways:

  - Start the server entering the password in plain text as a command line argument, for example:

    ```
    -Dserver.identity.password=changeme
    ```

    NOTE

    The password must be entered in plain text and will be visible to anyone who issues a **`ps -ef`** command.

  - Place the password in a properties file and pass the properties file URL as a command line argument.

    A. Add the key/value pair to a properties file. For example:

    ```
    server.identity.password=changeme
    ```

    B. Start the server with the command line arguments

    ```
    --properties=URL_TO_PROPERTIES_FILE
    ```

    .

d. Save and exit the file.

5. **Restart the server.**
   The slave will now authenticate to the master using its host name as the username and the encrypted string as its password.

Result

Your standalone server, or servers within a server group of a managed domain, are now configured as mod_cluster worker nodes. If you deploy a clustered application, its sessions are replicated to all cluster nodes for failover, and it can accept requests from an external HTTPD server or load balancer. Each node of the cluster discovers the other nodes using automatic discovery, by default.To configure automatic discovery, and the other specific settings of the **`mod_cluster`** subsystem, refer to *Configure the mod_cluster Subsystem* in the *Administration and Configuration Guide*. To configure the Apache HTTPD server, refer to *Use an External HTTPD as the Web Front-end for JBoss EAP 6 Applications* in the *Administration and Configuration Guide*.

Report a bug

CHAPTER 13. PATCH INSTALLATION

13.1. ABOUT PATCHES AND UPGRADES

The patching mechanism in JBoss EAP 6 applies updates which are made available to a specific 'minor' version of JBoss EAP 6, for example JBoss EAP 6.2. Patches can contain one-off, security, or cumulative updates.

Upgrading between major and minor releases of JBoss EAP (for example, from 6.1 to 6.2) requires a different process.

Report a bug

13.2. ABOUT PATCHING MECHANISMS

JBoss patches are released in two forms.

- Asynchronous updates: one-off patches which are released outside the normal update cycle of the existing product. These may include security patches, as well as other one-off patches provided by Red Hat Global Support Services (GSS) to fix specific issues.

- Planned updates: These include cumulative patches, as well as micro, minor or major upgrades of an existing product. Cumulative patches include all previously developed asynchronous updates for that version of the product.

Deciding whether a patch is released as part of a planned update or an asynchronous update depends on the severity of the issue being fixed. An issue of low impact is typically deferred, and is resolved in the next cumulative patch or minor release of the affected product. Issues of moderate or higher impact are typically addressed in order of importance as an asynchronous update to the affected product, and contain a fix for only a specific issue.

Cumulative and security patches for JBoss products are distributed in two forms: zip (for all products) and RPM (for a subset of products).

IMPORTANT

A JBoss product installation must always only be updated using one patch method: either zip or RPM patches.

Security updates for JBoss products are provided by an erratum (for both zip and RPM methods). The erratum encapsulates a list of the resolved flaws, their severity ratings, the affected products, textual description of the flaws, and a reference to the patches. Bug fix updates are not announced via an erratum.

For more information on how Red Hat rates JBoss security flaws, refer to: Section 13.6, “Severity and Impact Rating of JBoss Security Patches”

Red Hat maintains a mailing list for notifying subscribers about security related flaws. See Section 13.3, “Subscribe to Patch Mailing Lists”

Report a bug

13.3. SUBSCRIBE TO PATCH MAILING LISTS

Summary

The JBoss team at Red Hat maintains a mailing list for security announcements for Red Hat JBoss Enterprise Middleware products. This topic covers what you need to do to subscribe to this list.

Prerequisites

- None

Procedure 13.1. Subscribe to the JBoss Watch List

1. Click the following link to go to the JBoss Watch mailing list page: JBoss Watch Mailing List.
2. Enter your email address in the **`Subscribing to Jboss-watch-list`** section.
3. [You may also wish to enter your name and select a password. Doing so is optional but recommended.]
4. Press the **`Subscribe`** button to start the subscription process.
5. You can browse the archives of the mailing list by going to: JBoss Watch Mailing List Archives.

Result

After confirmation of your email address, you will be subscribed to receive security related announcements from the JBoss patch mailing list.

Report a bug

13.4. INSTALL PATCHES IN ZIP FORM

13.4.1. The `patch` Command

The **`patch`** command is used to apply downloaded zip patches to a single JBoss EAP 6 server instance. It cannot be used to automatically patch JBoss EAP 6 server instances across a managed domain, but individual server instances in a managed domain can be patched independently.

IMPORTANT

JBoss EAP 6 server instances which have been installed using the RPM method cannot be updated using the **`patch`** command. Refer to Section 13.5, “Install Patches in RPM form” to update RPM-installed JBoss EAP 6 servers.

NOTE

The **`patch`** command can only be used with patches produced for versions of JBoss EAP 6.2 and later. For patches for versions of JBoss EAP prior to 6.2, you should instead refer to the relevant version's documentation available at https://access.redhat.com/site/documentation/.

In addition to applying patches, the **`patch`** command can give basic information on the state of installed patches, and also provides a way to immediately rollback the application of a patch.

Before starting a patch application or rollback operation, the **`patch`** tool will check the modules and other miscellaneous files that it is changing for any user modifications. If a user modification is detected, and a conflict-handling switch has not been specified, the **patch** tool will abort the operation and warn that there is a conflict. The warning will include a list of the modules and other files that are in conflict. To complete the operation, the **patch** command must be re-run with a switch specifying how to resolve the conflict: either to preserve the user modifications, or to override them.

Table 13.1. patch Command Arguments and Switches

Argument or Switch	Description
apply	Applies a patch.
--override-all	If there is a conflict, the patch operation overrides any user modifications.
--override-modules	If there is a conflict as a result of any modified modules, this switch overrides those modifications with the contents of the patch operation.
--override=*path*(,*path*)	For specified miscellaneous files only, this will override the conflicting modified files with the files in the patch operation.
--preserve=*path*(,*path*)	For specified miscellaneous files only, this will preserve the conflicting modified files.
info	Returns information on currently installed patches.
rollback	Rollsback the application of a patch.
--reset-configuration=*TRUE*|*FALSE*	Required for rollback, this specifies whether to restore the server configuration files as part of the rollback operation.

Report a bug

13.4.2. Installing Patches in Zip Form Using the patch Command

Summary

This task describes how to use the **patch** command to install patches for JBoss EAP 6 that are in the zip format.

IMPORTANT

The **patch** command is a feature that was added in JBoss EAP 6.2. For versions of JBoss EAP prior to 6.2, the process to install patches in zip form is different, and you should instead refer to the relevant version's documentation available at https://access.redhat.com/site/documentation/.

Prerequisites

- Valid access and subscription to the Red Hat Customer Portal.
- A current subscription to a JBoss product installed in zip format.
- Access to the Management CLI for the server instance to be updated. Refer to *Launch the Management CLI* in the *Administration and Configuration Guide*.

Procedure 13.2. Apply a zip patch to a JBoss EAP 6 server instance using the `patch` command

> **WARNING**
>
> Before installing a patch, you should backup your JBoss product along with all customized configuration files.

1. Download the patch zip file from the Customer Portal at https://access.redhat.com/downloads/
2. From the Management CLI, apply the patch with the following command with the appropriate path to the patch file:

   ```
   [standalone@localhost:9999 /] patch apply /path/to/downloaded-
   patch.zip
   ```

   The **patch** tool will warn if there are any conflicts in attempting the apply the patch. Refer to Section 13.4.1, “The **patch** Command” for available switches to re-run the command to resolve any conflicts.
3. Restart the JBoss EAP 6 server instance for the patch to take effect:

   ```
   [standalone@localhost:9999 /] shutdown --restart=true
   ```

Result

The JBoss EAP 6 server instance is patched with the latest update.

Report a bug

13.4.3. Rollback the Application of a Patch in Zip Form Using the `patch` Command

Summary

This task describes how to use the **patch** command to rollback the application of a previously applied zip patch in JBoss EAP 6.

WARNING

Rolling back the application of a patch using the **`patch`** command is not intended as a general uninstall functionality. It is only intended to be used immediately after the application of a patch which had undesirable consequences.

IMPORTANT

The **`patch`** command is a feature that was added in JBoss EAP 6.2. For versions of JBoss EAP prior to 6.2, the process to rollback patches in zip form is different, and you should instead refer to the relevant version's documentation available at https://access.redhat.com/site/documentation/.

Prerequisites

- A patch that was previously applied using the **`patch`** command.
- Access to the Management CLI for the server instance. Refer to *Launch the Management CLI* in the *Administration and Configuration Guide*.

Procedure 13.3. Rollback a patch from a JBoss EAP 6 server instance using the `patch` command

1. From the Management CLI, use the **`patch info`** command to find the ID of the patch that is to be rolled back.
    - For cumulative patches, the patch ID is the value of the first **`cumulative-patch-id`** shown in the **`patch info`** output.
    - One-off security or bug fix patch IDs are listed as the value of the first **`patches`** shown in the **`patch info`** output, with the most recently applied one-off patch listed first.
2. From the Management CLI, rollback the patch with the appropriate patch ID from the previous step.

WARNING

Use caution when specifying the value of the **`--reset-configuration`** switch.

If set to **`TRUE`**, the patch rollback process will also rollback the JBoss EAP 6 server configuration files to their pre-patch state. Any changes that were made to the JBoss EAP 6 server configuration files after the patch was applied will be lost.

If set to **`FALSE`**, the server configuration files will not be rolled back. In this situation, it is possible that the server will not start after the rollback, as the patch may have altered configurations, such as namespaces, which may no longer be valid and have to be fixed manually.

```
[standalone@localhost:9999 /] patch rollback PATCH_ID --reset-
configuration=TRUE
```

The **`patch`** tool will warn if there are any conflicts in attempting the rollback the patch. Refer to Section 13.4.1, "The **`patch`** Command" for available switches to re-run the command to resolve any conflicts.

3. Restart the JBoss EAP 6 server instance for the patch rollback to take effect:

```
[standalone@localhost:9999 /] shutdown --restart=true
```

Result

The patch, and optionally also the server configuration files, are rolled back on the JBoss EAP 6 server instance.

Report a bug

13.5. INSTALL PATCHES IN RPM FORM

Summary

JBoss patches are distributed in two forms: ZIP (for all products) and RPM (for a subset of products). This task describes the steps you need to take to install the patches via the RPM format.

Prerequisites

- A valid subscription to the Red Hat Network.
- A current subscription to a JBoss product installed via an RPM package.

Procedure 13.4. Apply a patch to a JBoss product via the RPM method

Security updates for JBoss products are provided by errata (for both zip and RPM methods). The errata encapsulates a list of the resolved flaws, their severity ratings, the affected products, textual description of the flaws, and a reference to the patches.

For RPM distributions of JBoss products, the errata include references to the updated RPM packages. The patch can be installed by using **`yum`**.

WARNING

Before installing a patch, you must backup your JBoss product along with all customized configuration files.

1. Get notified about the security patch either via being a subscriber to the JBoss watch mailing list or by browsing the JBoss watch mailing list archives.

2. Read the errata for the security patch and confirm that it applies to a JBoss product in your environment.

3. If the security patch applies to a JBoss product in your environment, then follow the link to download the updated RPM package which is included in the errata.

4. Use

   ```
   yum update
   ```

   to install the patch.

   IMPORTANT

   When updating an RPM installation, your JBoss product is updated cumulatively with all RPM-released fixes.

Result

The JBoss product is patched with the latest update using the RPM format.

Report a bug

13.6. SEVERITY AND IMPACT RATING OF JBOSS SECURITY PATCHES

To communicate the risk of each JBoss security flaw, Red Hat uses a four-point severity scale of low, moderate, important and critical, in addition to Common Vulnerability Scoring System (CVSS) version 2 base scores which can be used to identify the impact of the flaw.

Table 13.2. Severity Ratings of JBoss Security Patches

Severity	Description

Severity	Description
Critical	This rating is given to flaws that could be easily exploited by a remote unauthenticated attacker and lead to system compromise (arbitrary code execution) without requiring user interaction. These are the types of vulnerabilities that can be exploited by worms. Flaws that require an authenticated remote user, a local user, or an unlikely configuration are not classed as critical impact.
Important	This rating is given to flaws that can easily compromise the confidentiality, integrity, or availability of resources. These are the types of vulnerabilities that allow local users to gain privileges, allow unauthenticated remote users to view resources that should otherwise be protected by authentication, allow authenticated remote users to execute arbitrary code, or allow local or remote users to cause a denial of service.
Moderate	This rating is given to flaws that may be more difficult to exploit but could still lead to some compromise of the confidentiality, integrity, or availability of resources, under certain circumstances. These are the types of vulnerabilities that could have had a critical impact or important impact but are less easily exploited based on a technical evaluation of the flaw, or affect unlikely configurations.
Low	This rating is given to all other issues that have a security impact. These are the types of vulnerabilities that are believed to require unlikely circumstances to be able to be exploited, or where a successful exploit would give minimal consequences.

The impact component of a CVSS v2 score is based on a combined assessment of three potential impacts: Confidentiality (C), Integrity (I) and Availability (A). Each of these can be rated as None (N), Partial (P) or Complete (C).

Because the JBoss server process runs as an unprivileged user and is isolated from the host operating system, JBoss security flaws are only rated as having impacts of either None (N) or Partial (P).

Example 13.1. CVSS v2 Impact Score

The example below shows a CVSS v2 impact score, where exploiting the flaw would have no impact on system confidentiality, partial impact on system integrity and complete impact on system availability (that is, the system would become completely unavailable for any use, for example, via a kernel crash).

```
C:N/I:P/A:C
```

Combined with the severity rating and the CVSS score, organizations can make informed decisions on the risk each issue places on their unique environment and schedule upgrades accordingly.

For more information about CVSS2, please see: CVSS2 Guide.

Report a bug

13.7. MANAGE SECURITY UPDATES FOR DEPENDENCIES BUNDLED INSIDE THE APPLICATIONS DEPLOYED ON JBOSS EAP

Red Hat provides security patches for all components that are part of the JBoss EAP distribution. However, many users of JBoss EAP deploy applications which bundle their own dependencies, rather than exclusively using components provided as part of the JBoss EAP distribution. For example, a deployed WAR file may include dependency JARs in the WEB-INF/lib/ directory. These JARs are out of scope for security patches provided by Red Hat. Managing security updates for dependencies bundled inside the applications deployed on JBoss EAP is the responsibility of the applications' maintainers. The following tools and data sources may assist in this effort, and are provided without any support or warranty.

Tools and Data Sources

JBoss patch mailing lists

Subscribing to the JBoss patch mailing lists will keep you informed regarding security flaws that have been fixed in JBoss products, allowing you to check whether your deployed applications are bundling vulnerable versions of the affected components.

Security advisory page for bundled components.

Many open source components have their own security advisory page. For example, struts 2 is a commonly-used component with many known security issues that is not provided as part of the JBoss EAP distribution. The struts 2 project maintains an upstream security advisory page, which should be monitored if your deployed applications bundle struts 2. Many commercially-provided components also maintain security advisory pages.

Regularly scan your deployed applications for known vulnerabilities

There are several commercial tools available to do this. There is also an open source tool called Victims, which is developed by Red Hat employees, but comes with no support or warranty. Victims provides plugins for several build and integration tools, which automatically scan applications for bundling known-vulnerable dependencies. Plugins are available for Maven, Ant and Jenkins. For more information about the Victims tool, see https://victi.ms/about.html.

See Also:

- Section 13.3, “Subscribe to Patch Mailing Lists”

Report a bug

PART III. SECURING APPLICATIONS

CHAPTER 14. APPLICATION SECURITY

14.1. ABOUT APPLICATION SECURITY

Securing your applications is a multi-faceted and important concern for every application developer. JBoss EAP 6 provides all the tools you need to write secure applications, including the following abilities:

- Section 5.9.1, “About Authentication”
- Section 5.11.1, “About Authorization”
- Section 5.13.1, “About Security Auditing”
- Section 5.14.1, “About Security Mapping”
- Section 2.1, “About Declarative Security”
- Section 14.4.2.1, “About EJB Method Permissions”
- Section 14.4.3.1, “About EJB Security Annotations”

See also Section 17.3, “Use a Security Domain in Your Application”.

Report a bug

14.2. ENABLING/DISABLING DESCRIPTOR BASED PROPERTY REPLACEMENT

Summary

Finite control over descriptor property replacement was introduced in **`jboss-as-ee_1_1.xsd`**. This task covers the steps required to configure descriptor based property replacement.

Prerequisites

- Start the JBoss Enterprise Application Platform instance.
- Launch the Management CLI.

Descriptor based property replacement flags have boolean values:

- When set to **`true`**, property replacements are enabled.
- When set to **`false`**, property replacements are disabled.

Procedure 14.1. `jboss-descriptor-property-replacement`

`jboss-descriptor-property-replacement` is used to enable or disable property replacement in the following descriptors:

- **`jboss-ejb3.xml`**
- **`jboss-app.xml`**
- **`jboss-web.xml`**

- ***-jms.xml**
- ***-ds.xml**

The default value for **jboss-descriptor-property-replacement** is **true**.

1. In the Management CLI, run the following command to determine the value of **jboss-descriptor-property-replacement**:

   ```
   /subsystem=ee:read-attribute(name="jboss-descriptor-property-
   replacement")
   ```

2. Run the following command to configure the behavior:

   ```
   /subsystem=ee:write-attribute(name="jboss-descriptor-property-
   replacement",value=VALUE)
   ```

Procedure 14.2. spec-descriptor-property-replacement

spec-descriptor-property-replacement is used to enable or disable property replacement in the following descriptors:

- **ejb-jar.xml**
- **persistence.xml**

The default value for **spec-descriptor-property-replacement** is **false**.

1. In the Management CLI, run the following command to confirm the value of **spec-descriptor-property-replacement**:

   ```
   /subsystem=ee:read-attribute(name="spec-descriptor-property-
   replacement")
   ```

2. Run the following command to configure the behavior:

   ```
   /subsystem=ee:write-attribute(name="spec-descriptor-property-
   replacement",value=VALUE)
   ```

Result

The descriptor based property replacement tags have been successfully configured.

Report a bug

14.3. DATASOURCE SECURITY

14.3.1. About Datasource Security

The preferred solution for datasource security is the use of either security domains or password vaults. Examples of each are included below. For more information, refer to:

- Security domains: Section 4.3.3.1, “About Security Domains”.

- Password vaults: Section 10.11.1, “About Securing Sensitive Strings in Clear-Text Files”.

Example 14.1. Security Domain Example

```
<security>
   <security-domain>mySecurityDomain</security-domain>
</security>
```

Example 14.2. Password Vault Example

```
<security>
  <user-name>admin</user-name>

<password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4M
mEtMWNlMDMyNDdmNmI2TElORV9CUkVBS3ZhdWx0}</password>
</security>
```

Report a bug

14.4. EJB APPLICATION SECURITY

14.4.1. Security Identity

14.4.1.1. About EJB Security Identity

The *security identity*, which is also known as *invocation identity*, refers to the **`<security-identity>`** tag in the security configuration. It refers to the identity another EJB must use when it invokes methods on components.

The invocation identity can be either the current caller, or it can be a specific role. In the first case, the **`<use-caller-identity>`** tag is present, and in the second case, the **`<run-as>`** tag is used.

For information about setting the security identity of an EJB, refer to Section 14.4.1.2, “Set the Security Identity of an EJB”.

Report a bug

14.4.1.2. Set the Security Identity of an EJB

Example 14.3. Set the security identity of an EJB to be the same as its caller

This example sets the security identity for method invocations made by an EJB to be the same as the current caller's identity. This behavior is the default if you do not specify a **`<security-identity>`** element declaration.

```
<ejb-jar>
  <enterprise-beans>
  <session>
  <ejb-name>ASessionBean</ejb-name>
  <!-- ... -->
```

```
    <security-identity>
      <use-caller-identity/>
    </security-identity>
    </session>
    <!-- ... -->
    </enterprise-beans>
  </ejb-jar>
```

Example 14.4. Set the security identity of an EJB to a specific role

To set the security identity to a specific role, use the **`<run-as>`** and **`<role-name>`** tags inside the **`<security-identity>`** tag.

```
<ejb-jar>
  <enterprise-beans>
  <session>
  <ejb-name>RunAsBean</ejb-name>
  <!-- ... -->
  <security-identity>
    <run-as>
    <description>A private internal role</description>
    <role-name>InternalRole</role-name>
    </run-as>
  </security-identity>
  </session>
  </enterprise-beans>
  <!-- ... -->
</ejb-jar>
```

By default, when you use **`<run-as>`**, a principal named **`anonymous`** is assigned to outgoing calls. To assign a different principal, uses the **`<run-as-principal>`**.

```
<session>
    <ejb-name>RunAsBean</ejb-name>
    <security-identity>
        <run-as-principal>internal</run-as-principal>
    </security-identity>
</session>
```

NOTE

You can also use the **`<run-as>`** and **`<run-as-principal>`** elements inside a servlet element.

See also:

- Section 14.4.1.1, “About EJB Security Identity”
- Section A.6, “EJB Security Parameter Reference”

Report a bug

14.4.2. EJB Method Permissions

14.4.2.1. About EJB Method Permissions

EJB provides a **`<method-permisison>`** element declaration. This declaration sets the roles which are allowed to invoke an EJB's interface methods. You can specify permissions for the following combinations:

- All home and component interface methods of the named EJB
- A specified method of the home or component interface of the named EJB
- A specified method within a set of methods with an overloaded name

For examples, see Section 14.4.2.2, “Use EJB Method Permissions”.

Report a bug

14.4.2.2. Use EJB Method Permissions

Overview

The **`<method-permission>`** element defines the logical roles that are allowed to access the EJB methods defined by **`<method>`** elements. Several examples demonstrate the syntax of the XML. Multiple method permission statements may be present, and they have a cumulative effect. The **`<method-permission>`** element is a child of the **`<assembly-descriptor>`** element of the **`<ejb-jar>`** descriptor.

The XML syntax is an alternative to using annotations for EJB method permissions.

Example 14.5. Allow roles to access all methods of an EJB

```
<method-permission>
  <description>The employee and temp-employee roles may access any
method
  of the EmployeeService bean </description>
  <role-name>employee</role-name>
  <role-name>temp-employee</role-name>
  <method>
    <ejb-name>EmployeeService</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
```

Example 14.6. Allow roles to access only specific methods of an EJB, and limiting which method parameters can be passed.

```
<method-permission>
  <description>The employee role may access the findByPrimaryKey,
  getEmployeeInfo, and the updateEmployeeInfo(String) method of
  the AcmePayroll bean </description>
  <role-name>employee</role-name>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>findByPrimaryKey</method-name>
  </method>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>getEmployeeInfo</method-name>
  </method>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>updateEmployeeInfo</method-name>
 <method-params>
   <method-param>java.lang.String</method-param>
 </method-params>
  </method>
</method-permission>
```

Example 14.7. Allow any authenticated user to access methods of EJBs

Using the **`<unchecked/>`** element allows any authenticated user to use the specified methods.

```
<method-permission>
  <description>Any authenticated user may access any method of the
  EmployeeServiceHelp bean</description>
  <unchecked/>
  <method>
 <ejb-name>EmployeeServiceHelp</ejb-name>
 <method-name>*</method-name>
  </method>
</method-permission>
```

Example 14.8. Completely exclude specific EJB methods from being used

```
<exclude-list>
  <description>No fireTheCTO methods of the EmployeeFiring bean may be
  used in this deployment</description>
  <method>
 <ejb-name>EmployeeFiring</ejb-name>
 <method-name>fireTheCTO</method-name>
  </method>
</exclude-list>
```

Example 14.9. A complete `<assembly-descriptor>` containing several `<method-permission>` blocks

```
<ejb-jar>
    <assembly-descriptor>
        <method-permission>
            <description>The employee and temp-employee roles may
access any
                method of the EmployeeService bean </description>
            <role-name>employee</role-name>
            <role-name>temp-employee</role-name>
            <method>
                <ejb-name>EmployeeService</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <method-permission>
            <description>The employee role may access the
findByPrimaryKey,
                getEmployeeInfo, and the updateEmployeeInfo(String)
method of
                the AcmePayroll bean </description>
            <role-name>employee</role-name>
            <method>
                <ejb-name>AcmePayroll</ejb-name>
                <method-name>findByPrimaryKey</method-name>
            </method>
            <method>
                <ejb-name>AcmePayroll</ejb-name>
                <method-name>getEmployeeInfo</method-name>
            </method>
            <method>
                <ejb-name>AcmePayroll</ejb-name>
                <method-name>updateEmployeeInfo</method-name>
                <method-params>
                    <method-param>java.lang.String</method-param>
                </method-params>
            </method>
        </method-permission>
        <method-permission>
            <description>The admin role may access any method of the
                EmployeeServiceAdmin bean </description>
            <role-name>admin</role-name>
            <method>
                <ejb-name>EmployeeServiceAdmin</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <method-permission>
            <description>Any authenticated user may access any method
of the
                EmployeeServiceHelp bean</description>
            <unchecked/>
            <method>
                <ejb-name>EmployeeServiceHelp</ejb-name>
                <method-name>*</method-name>
            </method>
        </method-permission>
        <exclude-list>
            <description>No fireTheCTO methods of the EmployeeFiring
bean may be
                used in this deployment</description>
            <method>
                <ejb-name>EmployeeFiring</ejb-name>
                <method-name>fireTheCTO</method-name>
            </method>
        </exclude-list>
    </assembly-descriptor>
</ejb-jar>
```

Report a bug

14.4.3. EJB Security Annotations

14.4.3.1. About EJB Security Annotations

EJBs use security annotations to pass information about security to the deployer. These include:

@DeclareRoles
: Declares which roles are available.

@RolesAllowed, @PermitAll, @DenyAll
: Specifies which method permissions are allowed. For information about method permissions, refer to Section 14.4.2.1, “About EJB Method Permissions”.

@RunAs
: Configures the propagated security identify of a component.

For more information, refer to Section 14.4.3.2, “Use EJB Security Annotations”.

Report a bug

14.4.3.2. Use EJB Security Annotations

Overview

You can use either XML descriptors or annotations to control which security roles are able to call methods in your Enterprise JavaBeans (EJBs). For information on using XML descriptors, refer to Section 14.4.2.2, “Use EJB Method Permissions”.

Annotations for Controlling Security Permissions of EJBs

@DeclareRoles
: Use @DeclareRoles to define which security roles to check permissions against. If no @DeclareRoles is present, the list is built automatically from the @RolesAllowed annotation.

@SecurityDomain
: Specifies the security domain to use for the EJB. If the EJB is annotated for authorization with **`@RolesAllowed`**, authorization will only apply if the EJB is annotated with a security domain.

@RolesAllowed, @PermitAll, @DenyAll
: Use @RolesAllowed to list which roles are allowed to access a method or methods. Use @PermitAll or @DenyAll to either permit or deny all roles from using a method or methods.

@RunAs

Use @RunAs to specify a role a method will always be run as.

Example 14.10. Security Annotations Example

```
@Stateless
@RolesAllowed({"admin"})
@SecurityDomain("other")
public class WelcomeEJB implements Welcome {
 @PermitAll
 public String WelcomeEveryone(String msg) {
  return "Welcome to " + msg;
 }
 @RunAs("tempemployee")
 public String GoodBye(String msg) {
     return "Goodbye, " + msg;
 }
 public String GoodbyeAdmin(String msg) {
  return "See you later, " + msg;
 }
}
```

In this code, all roles can access method **`WelcomeEveryone`**. The **`GoodBye`** method uses the **`tempemployee`** role when making calls. Only the **`admin`** role can access method **`GoodbyeAdmin`**, and any other methods with no security annotation.

Report a bug

14.4.4. Remote Access to EJBs

14.4.4.1. About Remote Method Access

JBoss Remoting is the framework which provides remote access to EJBs, JMX MBeans, and other similar services. It works within the following transport types, with or without SSL:

Supported Transport Types

- Socket / Secure Socket
- RMI / RMI over SSL
- HTTP / HTTPS
- Servlet / Secure Servlet
- Bisocket / Secure Bisocket

JBoss Remoting also provides automatic discovery via Multicast or JNDI.

It is used by many of the subsystems within JBoss EAP 6, and also enables you to design, implement, and deploy services that can be remotely invoked by clients over several different transport mechanisms. It also allows you to access existing services in JBoss EAP 6.

Data Marshalling

The Remoting system also provides data marshalling and unmarshalling services. Data marshalling refers to the ability to safely move data across network and platform boundaries, so that a separate system can perform work on it. The work is then sent back to the original system and behaves as though it were handled locally.

Architecture Overview

When you design a client application which uses Remoting, you direct your application to communicate with the server by configuring it to use a special type of resource locator called an **`InvokerLocator`**, which is a simple String with a URL-type format. The server listens for requests for remote resources on a **`connector`**, which is configured as part of the **`remoting`** subsystem. The **`connector`** hands the request off to a configured **`ServerInvocationHandler`**. Each **`ServerInvocationHandler`** implements a method **`invoke(InvocationRequest)`**, which knows how to handle the request.

The JBoss Remoting framework contains three layers that mirror each other on the client and server side.

JBoss Remoting Framework Layers

- The user interacts with the outer layer. On the client side, the outer layer is the **`Client`** class, which sends invocation requests. On the server side, it is the InvocationHandler, which is implemented by the user and receives invocation requests.
- The transport is controlled by the invoker layer.
- The lowest layer contains the marshaller and unmarshaller, which convert data formats to wire formats.

Report a bug

14.4.4.2. About Remoting Callbacks

When a Remoting client requests information from the server, it can block and wait for the server to reply, but this is often not the ideal behavior. To allow the client to listen for asynchronous events on the server, and continue doing other work while waiting for the server to finish the request, your application can ask the server to send a notification when it has finished. This is referred to as a callback. One client can add itself as a listener for asynchronous events generated on behalf of another client, as well. There are two different choices for how to receive callbacks: pull callbacks or push callbacks. Clients check for pull callbacks synchronously, but passively listen for push callbacks.

In essence, a callback works by the server sending an **`InvocationRequest`** to the client. Your server-side code works the same regardless of whether the callback is synchronous or asynchronous. Only the client needs to know the difference. The server's InvocationRequest sends a **`responseObject`** to the client. This is the payload that the client has requested. This may be a direct response to a request or an event notification.

Your server also tracks listeners using an **`m_listeners`** object. It contains a list of all listeners that have been added to your server handler. The **`ServerInvocationHandler`** interface includes methods that allow you to manage this list.

The client handles pull and push callback in different ways. In either case, it must implement a callback handler. A callback handler is an implementation of interface **`org.jboss.remoting.InvokerCallbackHandler`**, which processes the callback data. After implementing the callback handler, you either add yourself as a listener for a pull callback, or implement a callback server for a push callback.

Pull Callbacks

For a pull callback, your client adds itself to the server's list of listeners using the **`Client.addListener()`** method. It then polls the server periodically for synchronous delivery of callback data. This poll is performed using the **`Client.getCallbacks()`**.

Push Callback

A push callback requires your client application to run its own InvocationHandler. To do this, you need to run a Remoting service on the client itself. This is referred to as a *callback server*. The callback server accepts incoming requests asynchronously and processes them for the requester (in this case, the server). To register your client's callback server with the main server, pass the callback server's **`InvokerLocator`** as the second argument to the **`addListener`** method.

Report a bug

14.4.4.3. About Remoting Server Detection

Remoting servers and clients can automatically detect each other using JNDI or Multicast. A Remoting Detector is added to both the client and server, and a NetworkRegistry is added to the client.

The Detector on the server side periodically scans the InvokerRegistry and pulls all server invokers it has created. It uses this information to publish a detection message which contains the locator and subsystems supported by each server invoker. It publishes this message via a multicast broadcast or a binding into a JNDI server.

On the client side, the Detector receives the multicast message or periodically polls the JNDI server to retrieve detection messages. If the Detector notices that a detection message is for a newly-detected remoting server, it registers it into the NetworkRegistry. The Detector also updates the NetworkRegistry if it detects that a server is no longer available.

Report a bug

14.4.4.4. Configure the Remoting Subsystem

Overview

JBoss Remoting has three top-level configurable elements: the worker thread pool, one or more connectors, and a series of local and remote connection URIs. This topic presents an explanation of each configurable item, example CLI commands for how to configure each item, and an XML example of a fully-configured subsystem. This configuration only applies to the server. Most people will not need to configure the Remoting subsystem at all, unless they use custom connectors for their own applications. Applications which act as Remoting clients, such as EJBs, need separate configuration to connect to a specific connector.

NOTE

The Remoting subsystem configuration is not exposed to the web-based Management Console, but it is fully configurable from the command-line based Management CLI. Editing the XML by hand is not recommended.

Adapting the CLI Commands

The CLI commands are formulated for a managed domain, when configuring the **`default`** profile. To configure a different profile, substitute its name. For a standalone server, omit the **`/profile=default`** part of the command.

Configuration Outside the Remoting Subsystem

There are a few configuration aspects which are outside of the **`remoting`** subsystem:

Network Interface

The network interface used by the **`remoting`** subsystem is the **`unsecure`** interface defined in the **`domain/configuration/domain.xml`** or **`standalone/configuration/standalone.xml`**.

```
<interfaces>
   <interface name="management"/>
   <interface name="public"/>
   <interface name="unsecure"/>
</interfaces>
```

The per-host definition of the **`unsecure`** interface is defined in the **`host.xml`** in the same directory as the **`domain.xml`** or **`standalone.xml`**. This interface is also used by several other subsystems. Exercise caution when modifying it.

```
<interfaces>
   <interface name="management">
      <inet-address
value="${jboss.bind.address.management:127.0.0.1}"/>
   </interface>
   <interface name="public">
      <inet-address value="${jboss.bind.address:127.0.0.1}"/>
   </interface>
   <interface name="unsecure">
      <!-- Used for IIOP sockets in the standard configuration.
         To secure JacORB you need to setup SSL -->
      <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
   </interface>
</interfaces>
```

socket-binding

The default socket-binding used by the **`remoting`** subsystem binds to TCP port 4777. Refer to the documentation about socket bindings and socket binding groups for more information if you need to change this.

Remoting Connector Reference for EJB

The EJB subsystem contains a reference to the remoting connector for remote method invocations. The following is the default configuration:

```
<remote connector-ref="remoting-connector" thread-pool-name="default"/>
```

Secure Transport Configuration

Remoting transports use StartTLS to use a secure (HTTPS, Secure Servlet, etc) connection if the client requests it. The same socket binding (network port) is used for secured and unsecured connections, so no additional server-side configuration is necessary. The client requests the secure or unsecured transport, as its needs dictate. JBoss EAP 6 components which use Remoting, such as EJBs, the ORB, and the JMS provider, request secured interfaces by default.

WARNING

StartTLS works by activating a secure connection if the client requests it, and otherwise defaulting to an unsecured connection. It is inherently susceptible to a *Man in the Middle* style exploit, wherein an attacker intercepts the client's request and modifies it to request an unsecured connection. Clients must be written to fail appropriately if they do not receive a secure connection, unless an unsecured connection actually is an appropriate fall-back.

Worker Thread Pool

The worker thread pool is the group of threads which are available to process work which comes in through the Remoting connectors. It is a single element **`<worker-thread-pool>`**, and takes several attributes. Tune these attributes if you get network timeouts, run out of threads, or need to limit memory usage. Specific recommendations depend on your specific situation. Contact Red Hat Global Support Services for more information.

Table 14.1. Worker Thread Pool Attributes

Attribute	Description	CLI Command
read-threads	The number of read threads to create for the remoting worker. Defaults to **`1`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-read-threads,value=1)`**
write-threads	The number of write threads to create for the remoting worker. Defaults to **`1`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-write-threads,value=1)`**
task-keepalive	The number of milliseconds to keep non-core remoting worker task threads alive. Defaults to **`60`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-task-keepalive,value=60)`**
task-max-threads	The maximum number of threads for the remoting worker task thread pool. Defaults to **`16`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-task-max-threads,value=16)`**
task-core-threads	The number of core threads for the remoting worker task thread pool. Defaults to **`4`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-task-core-threads,value=4)`**

Attribute	Description	CLI Command
task-limit	The maximum number of remoting worker tasks to allow before rejecting. Defaults to **`16384`**.	**`/profile=default/subsystem=remoting/:write-attribute(name=worker-task-limit,value=16384)`**

Connector

The connector is the main Remoting configuration element. Multiple connectors are allowed. Each consists of a element **`<connector>`** element with several sub-elements, as well as a few possible attributes. The default connector is used by several subsystems of JBoss EAP 6. Specific settings for the elements and attributes of your custom connectors depend on your applications, so contact Red Hat Global Support Services for more information.

Table 14.2. Connector Attributes

Attribute	Description	CLI Command
socket-binding	The name of the socket binding to use for this connector.	**`/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=socket-binding,value=remoting)`**
authentication-provider	The Java Authentication Service Provider Interface for Containers (JASPIC) module to use with this connector. The module must be in the classpath.	**`/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=authentication-provider,value=myProvider)`**
security-realm	Optional. The security realm which contains your application's users, passwords, and roles. An EJB or Web Application can authenticate against a security realm. **`ApplicationRealm`** is available in a default JBoss EAP 6 installation.	**`/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=security-realm,value=ApplicationRealm)`**

Table 14.3. Connector Elements

Attribute	Description	CLI Command

Attribute	Description	CLI Command
sasl	Enclosing element for Simple Authentication and Security Layer (SASL) authentication mechanisms	**`N/A`**
properties	Contains one or more **`<property>`** elements, each with a **`name`** attribute and an optional **`value`** attribute.	**`/profile=default/subsystem=remoting/connector=remoting-connector/property=myProp/:add(value=myPropValue)`**

Outbound Connections

You can specify three different types of outbound connection:

- Outbound connection to a URI.
- Local outbound connection – connects to a local resource such as a socket.
- Remote outbound connection – connects to a remote resource and authenticates using a security realm.

All of the outbound connections are enclosed in an **`<outbound-connections>`** element. Each of these connection types takes an **`outbound-socket-binding-ref`** attribute. The outbound-connection takes a **`uri`** attribute. The remote outbound connection takes optional **`username`** and **`security-realm`** attributes to use for authorization.

Table 14.4. Outbound Connection Elements

Attribute	Description	CLI Command
outbound-connection	Generic outbound connection.	**`/profile=default/subsystem=remoting/outbound-connection=my-connection/:add(uri=http://my-connection)`**
local-outbound-connection	Outbound connection with a implicit local:// URI scheme.	**`/profile=default/subsystem=remoting/local-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting2)`**
remote-outbound-connection	Outbound connections for remote:// URI scheme, using basic/digest authentication with a security realm.	**`/profile=default/subsystem=remoting/remote-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting,username=myUser,security-realm=ApplicationRealm)`**

SASL Elements

Before defining the SASL child elements, you need to create the initial SASL element. Use the following command:

```
/profile=default/subsystem=remoting/connector=remoting-
connector/security=sasl:add
```

The child elements of the SASL element are described in the table below.

Attribute	Description	CLI Command
include-mechanisms	Contains a **`value`** attribute, which is a space-separated list of SASL mechanisms.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=include-mechanisms,value=["DIGEST","PLAIN","GSSAPI"])`
qop	Contains a **`value`** attribute, which is a space-separated list of SASL Quality of protection values, in decreasing order of preference.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=qop,value=["auth"])`
strength	Contains a **`value`** attribute, which is a space-separated list of SASL cipher strength values, in decreasing order of preference.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=strength,value=["medium"])`
reuse-session	Contains a **`value`** attribute which is a boolean value. If true, attempt to reuse sessions.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=reuse-session,value=false)`

Attribute	Description	CLI Command
server-auth	Contains a **`value`** attribute which is a boolean value. If true, the server authenticates to the client.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=server-auth,value=false)`

policy	An enclosing element which contains zero or more of the following elements, which each take a single **`value`**.
	• forward-secrecy – whether mechanisms are required to implement forward secrecy (breaking into one session will not automatically provide information for breaking into future sessions)
	• no-active – whether mechanisms susceptible to non-dictionary attacks are permitted. A value of **`false`** permits, and **`true`** denies.
	• no-anonymous – whether mechanisms that accept anonymous login are permitted. A value of **`false`** permits, and **`true`** denies.
	• no-dictionary – whether mechanisms susceptible to passive dictionary attacks are allowed. A value of **`false`** permits, and **`true`** denies.
	• no-plain-text – whether mechanisms which are susceptible to simple plain passive attacks are allowed. A value of **`false`** permits, and **`true`** denies.
	• pass-credentials – whether mechanisms which pass client credentials are allowed.
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:add`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=forward-secrecy,value=true)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-active,value=false)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-anonymous,value=false)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-`

Attribute	Description	CLI Command

		`policy=policy:write-attribute(name=no-dictionary,value=true)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-plain-text,value=false)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=pass-credentials,value=true)`

properties	Contains one or more **`<property>`** elements, each with a **`name`** attribute and an optional **`value`** attribute.	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/property=myprop:add(value=1)`
		
		`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/property=myprop2:add(value=2)`

Example 14.11. Example Configurations

This example shows the default remoting subsystem that ships with JBoss EAP 6.

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <connector name="remoting-connector" socket-binding="remoting"
security-realm="ApplicationRealm"/>
</subsystem>
```

This example contains many hypothetical values, and is presented to put the elements and attributes discussed previously into context.

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <worker-thread-pool read-threads="1" task-keepalive="60' task-max-
threads="16" task-core-thread="4" task-limit="16384" write-threads="1"
/>
    <connector name="remoting-connector" socket-binding="remoting"
security-realm="ApplicationRealm">
        <sasl>
            <include-mechanisms value="GSSAPI PLAIN DIGEST-MD5" />
            <qop value="auth" />
            <strength value="medium" />
            <reuse-session value="false" />
            <server-auth value="false" />
            <policy>
                <forward-secrecy value="true" />
                <no-active value="false" />
                <no-anonymous value="false" />
                <no-dictionary value="true" />
                <no-plain-text value="false" />
                <pass-credentials value="true" />
            </policy>
            <properties>
                <property name="myprop1" value="1" />
                <property name="myprop2" value="2" />
            </properties>
        </sasl>
        <authentication-provider name="myprovider" />
        <properties>
            <property name="myprop3" value="propValue" />
        </properties>
    </connector>
    <outbound-connections>
        <outbound-connection name="my-outbound-connection"
uri="http://myhost:7777/"/>
        <remote-outbound-connection name="my-remote-connection"
outbound-socket-binding-ref="my-remote-socket" username="myUser"
security-realm="ApplicationRealm"/>
        <local-outbound-connection name="myLocalConnection" outbound-
socket-binding-ref="my-outbound-socket"/>
    </outbound-connections>
</subsystem>
```

Configuration Aspects Not Yet Documented

- JNDI and Multicast Automatic Detection

Report a bug

14.4.4.5. Use Security Realms with Remote EJB Clients

One way to add security to clients which invoke EJBs remotely is to use security realms. A security realm is a simple database of username/password pairs and username/role pairs. The terminology is also used in the context of web containers, with a slightly different meaning.

To authenticate an EJB to a specific username and password which exists in a security realm, follow these steps:

- Add a new security realm to the domain controller or standalone server.
- Add the following parameters to the **`jboss-ejb-client.properties`** file, which is in the classpath of the application. This example assumes the connection is referred to as **`default`** by the other parameters in the file.

  ```
  remote.connection.default.username=appuser
  remote.connection.default.password=apppassword
  ```

- Create a custom Remoting connector on the domain or standalone server, which uses your new security realm.
- Deploy your EJB to the server group which is configured to use the profile with the custom Remoting connector, or to your standalone server if you are not using a managed domain.

Report a bug

14.4.4.6. Add a New Security Realm

1. **Run the Management CLI.**
   Start the **`jboss-cli.sh`** or **`jboss-cli.bat`** command and connect to the server.
2. **Create the new security realm itself.**
   Run the following command to create a new security realm named **`MyDomainRealm`** on a domain controller or a standalone server.

   ```
   /host=master/core-service=management/security-
   realm=MyDomainRealm:add()
   ```

3. **Create the references to the properties file which will store information about the new role.**
   Run the following command to create a pointer a file named **`myfile.properties`**, which will contain the properties pertaining to the new role.

   NOTE

   The newly-created properties file is not managed by the included **`add-user.sh`** and **`add-user.bat`** scripts. It must be managed externally.

   ```
   /host=master/core-service=management/security-
   realm=MyDomainRealm/authentication=properties:add(path=myfile.proper
   ties)
   ```

Result

Your new security realm is created. When you add users and roles to this new realm, the information will be stored in a separate file from the default security realms. You can manage this new file using your own applications or procedures.

Report a bug

14.4.4.7. Add a User to a Security Realm

1. **Run the `add-user.sh` or `add-user.bat` command.**
   Open a terminal and change directories to the ***`EAP_HOME`*****`/bin/`** directory. If you run Red Hat Enterprise Linux or another UNIX-like operating system, run **`add-user.sh`**. If you run Microsoft Windows Server, run **`add-user.bat`**.

2. **Choose whether to add a Management User or Application User.**
   For this procedure, type **`b`** to add an Application User.

3. **Choose the realm the user will be added to.**
   By default, the only available realm is **`ApplicationRealm`**. If you have added a custom realm, you can type its name instead.

4. **Type the username, password, and roles, when prompted.**
   Type the desired username, password, and optional roles when prompted. Verify your choice by typing **`yes`**, or type **`no`** to cancel the changes. The changes are written to each of the properties files for the security realm.

Report a bug

14.4.4.8. About Remote EJB Access Using SSL Encryption

By default, the network traffic for Remote Method Invocation (RMI) of EJB2 and EJB3 Beans is not encrypted. In instances where encryption is required, Secure Sockets Layer (SSL) can be utilized so that the connection between the client and server is encrypted. Using SSL also has the added benefit of allowing the network traffic to traverse firewalls that block the RMI port.

Report a bug

14.5. JAX-RS APPLICATION SECURITY

14.5.1. Enable Role-Based Security for a RESTEasy JAX-RS Web Service

Summary

RESTEasy supports the @RolesAllowed, @PermitAll, and @DenyAll annotations on JAX-RS methods. However, it does not recognize these annotations by default. Follow these steps to configure the **`web.xml`** file and enable role-based security.

WARNING

Do not activate role-based security if the application uses EJBs. The EJB container will provide the functionality, instead of RESTEasy.

Procedure 14.3. Enable Role-Based Security for a RESTEasy JAX-RS Web Service

1. Open the **`web.xml`** file for the application in a text editor.

2. Add the following <context-param> to the file, within the **`web-app`** tags:

```
<context-param>
    <param-name>resteasy.role.based.security</param-name>
    <param-value>true</param-value>
</context-param>
```

3. Declare all roles used within the RESTEasy JAX-RS WAR file, using the <security-role> tags:

```
<security-role>
    <role-name>ROLE_NAME</role-name>
</security-role>
<security-role>
    <role-name>ROLE_NAME</role-name>
</security-role>
```

4. Authorize access to all URLs handled by the JAX-RS runtime for all roles:

```
<security-constraint>
    <web-resource-collection>
 <web-resource-name>Resteasy</web-resource-name>
 <url-pattern>/PATH</url-pattern>
    </web-resource-collection>
    <auth-constraint>
 <role-name>ROLE_NAME</role-name>
 <role-name>ROLE_NAME</role-name>
    </auth-constraint>
</security-constraint>
```

Result

Role-based security has been enabled within the application, with a set of defined roles.

Example 14.12. Example Role-Based Security Configuration

```
<web-app>

    <context-param>
 <param-name>resteasy.role.based.security</param-name>
 <param-value>true</param-value>
    </context-param>
```

```
   <servlet-mapping>
<servlet-name>Resteasy</servlet-name>
<url-pattern>/*</url-pattern>
   </servlet-mapping>

   <security-constraint>
<web-resource-collection>
    <web-resource-name>Resteasy</web-resource-name>
    <url-pattern>/security</url-pattern>
</web-resource-collection>
<auth-constraint>
    <role-name>admin</role-name>
    <role-name>user</role-name>
</auth-constraint>
   </security-constraint>

   <security-role>
<role-name>admin</role-name>
   </security-role>
   <security-role>
<role-name>user</role-name>
   </security-role>

</web-app>
```

Report a bug

14.5.2. Secure a JAX-RS Web Service using Annotations

Summary

This topic covers the steps to secure a JAX-RS web service using the supported security annotations

Procedure 14.4. Secure a JAX-RS Web Service using Supported Security Annotations

1. Enable role-based security. For more information, refer to: Section 14.5.1, “Enable Role-Based Security for a RESTEasy JAX-RS Web Service”

2. Add security annotations to the JAX-RS web service. RESTEasy supports the following annotations:

   @RolesAllowed

   Defines which roles can access the method. All roles should be defined in the **`web.xml`** file.

   @PermitAll

   Allows all roles defined in the **`web.xml`** file to access the method.

   @DenyAll

   Denies all access to the method.

Report a bug

CHAPTER 15. LOGIN MODULES

Report a bug

15.1. USING MODULES

JBoss EAP 6 includes several bundled login modules suitable for most user management needs. JBoss EAP 6 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 for very customized needs.

More login modules and their options can be found in Appendix A.1.

Report a bug

15.1.1. 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.

Section 15.1.4, “Ldap Login Module” describes how to combine LDAP and a relational database, allowing a user to be authenticated by either system. 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 <module-option> **`password-stacking`** attribute to **`useFirstPass`**. 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.

NOTE

When using password stacking, set all modules to be required. This ensures that all modules are considered, and have the chance to contribute roles to the authorization process.

Example 15.1. Password Stacking Sample

This management CLI example shows how password stacking could be used.

```
/subsystem=security/security-
domain=pwdStack/authentication=classic/login-module=Ldap:add( \
  code=Ldap, \
  flag=required, \
  module-options=[ \
    ("password-stacking"=>"useFirstPass"), \
    ... Ldap login module configuration
  ])
/subsystem=security/security-
domain=pwdStack/authentication=classic/login-module=Database:add( \
  code=Database, \
  flag=required, \
  module-options=[ \
    ("password-stacking"=>"useFirstPass"), \
    ... Database login module configuration
  ])
```

Report a bug

15.1.2. 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.

IMPORTANT

Red Hat JBoss Enterprise Application Platform Common Criteria certified release only supports SHA-256 for password hashing.

Example 15.2. Password Hashing

The following is a login module configuration that assigns unauthenticated users the principal name **nobody** and contains based64-encoded, SHA-256 hashes of the passwords in a **usersb64.properties** file. The **usersb64.properties** file is part of the deployment classpath.

```
/subsystem=security/security-domain=testUsersRoles:add
/subsystem=security/security-
domain=testUsersRoles/authentication=classic:add
/subsystem=security/security-
domain=testUsersRoles/authentication=classic/login-
module=UsersRoles:add( \
  code=UsersRoles, \
  flag=required, \
  module-options=[ \
    ("usersProperties"=>"usersb64.properties"), \
    ("rolesProperties"=>"test-users-roles.properties"), \
    ("unauthenticatedIdentity"=>"nobody"), \
    ("hashAlgorithm"=>"SHA-256"), \
    ("hashEncoding"=>"base64") \
  ])
```

hashAlgorithm

Name of the **java.security.MessageDigest** algorithm to use to hash the password. There is no default so this option must be specified to enable hashing. Typical values are **SHA-256**, **SHA-1** and **MD5**.

hashEncoding

String that specifies one of three encoding types: **base64**, **hex** or **rfc2617**. The default is **base64**.

hashCharset

Encoding character set used to convert the clear text password to a byte array. The platform default encoding is the default.

hashUserPassword

Specifies the hashing algorithm must be applied to the password the user submits. The hashed user password is compared against the value in the login module, which is expected to be a hash of the password. The default is **true**.

hashStorePassword

Specifies the hashing algorithm must be applied to the password stored on the server side. This is used for digest authentication, where the user submits a hash of the user password along with a request-specific tokens from the server to be compare. The hash algorithm (for digest, this would be **rfc2617**) is utilized to compute a server-side hash, which should match the hashed value sent from the client.

If you must generate passwords in code, the **org.jboss.security.auth.spi.Util** class provides a static helper method that will hash a password using the specified encoding. The following example produces a base64-encoded, MD5 hashed password.

```
String hashedPassword = Util.createPasswordHash("SHA-256",
 Util.BASE64_ENCODING, null, null, "password");
```

OpenSSL provides an alternative way to quickly generate hashed passwords at the command-line. The following example also produces a base64-encoded, SHA-256 hashed password. Here the password in plain text - **password** - is piped into the OpenSSL digest function then piped into another OpenSSL function to convert into base64-encoded format.

```
echo -n password | openssl dgst -sha256 -binary | openssl base64
```

In both cases, the hashed version of the password is the same: **XohImNooBHFR0OVvjcYpJ3NgPQ1qq73WKhHvch0VQtg=**. This value must be stored in the users' properties file specified in the security domain - **usersb64.properties** - in the example above.

Report a bug

15.1.3. Unauthenticated Identity

Not all requests are 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.

- **unauthenticatedIdentity**: This defines the principal name that should be assigned to requests that contain no authentication information.

Report a bug

15.1.4. Ldap Login Module

`Ldap` login module is a **`LoginModule`** implementation that authenticates against a Lightweight Directory Access Protocol (LDAP) server. Use the **`Ldap`** login module if your user name and credentials are stored in an LDAP server that is accessible using a Java Naming and Directory Interface (JNDI) LDAP provider.

NOTE

If you wish to use LDAP with the SPNEGO authentication or skip some of the authentication phases while using an LDAP server, consider using the **`AdvancedLdap`** login module chained with the SPNEGOUsers login module or only the **`AdvancedLdap`** login module.

Distinguished Name (DN)

In Lightweight Directory Access Protocol (LDAP), the distinguished name 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.

NOTE

This login module also supports unauthenticated identity and password stacking.

The LDAP connectivity information is provided as configuration options that are passed through to the environment object used to create JNDI initial context. The standard LDAP JNDI properties used include the following:

java.naming.factory.initial

`InitialContextFactory` implementation class name. This defaults to the Sun LDAP provider implementation **`com.sun.jndi.ldap.LdapCtxFactory`**.

java.naming.provider.url

LDAP URL for the LDAP server.

java.naming.security.authentication

Security protocol level to use. The available values include **none**, **`simple`**, and **`strong`**. If the property is undefined, the behavior is determined by the service provider.

java.naming.security.protocol

Transport protocol to use for secure access. Set this configuration option to the type of service provider (for example, SSL). If the property is undefined, the behavior is determined by the service provider.

java.naming.security.principal

Specifies the identity of the Principal for authenticating the caller to the service. This is built from other properties as described below.

java.naming.security.credentials

Specifies the credentials of the Principal for authenticating the caller to the service. Credentials can take the form of a hashed password, a clear-text password, a key, or a certificate. If the property is undefined, the behavior is determined by the service provider.

For details of Ldap login module configuration options see Section A.1, "Included Authentication Modules".

NOTE

In certain directory schemas (e.g., Microsoft Active Directory), role attributes in the user object are stored as DNs to role objects instead of simple names. For implementations that use this schema type, roleAttributeIsDN must be set to **`true`**.

User authentication is performed by connecting to the LDAP server, based on the login module configuration options. Connecting to the LDAP server is done by creating an **`InitialLdapContext`** with an environment composed of the LDAP JNDI properties described previously in this section.

The Context.SECURITY_PRINCIPAL is set to the distinguished name of the user obtained by the callback handler in combination with the principalDNPrefix and principalDNSuffix option values, and the Context.SECURITY_CREDENTIALS property is set to the respective String password.

Once authentication has succeeded (**`InitialLdapContext`** instance is created), the user's roles are queried by performing a search on the **`rolesCtxDN`** location with search attributes set to the roleAttributeName and uidAttributeName option values. The roles names are obtaining by invoking the **`toString`** method on the role attributes in the search result set.

Example 15.3. LDAP Login Module Security Domain

This management CLI example shows how to use the parameters in a security domain authentication configuration.

```
/subsystem=security/security-domain=testLDAP:add(cache-type=default)
/subsystem=security/security-domain=testLDAP/authentication=classic:add
/subsystem=security/security-
domain=testLDAP/authentication=classic/login-module=Ldap:add( \
  code=Ldap, \
  flag=required, \
  module-options=[ \
    ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"),
\
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org:1389/"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("principalDNPrefix"=>"uid="), \
    ("principalDNSuffix"=>",ou=People,dc=jboss,dc=org"), \
    ("rolesCtxDN"=>"ou=Roles,dc=jboss,dc=org"), \
    ("uidAttributeID"=>"member"), \
    ("matchOnUserDN"=>true), \
    ("roleAttributeID"=>"cn"), \
    ("roleAttributeIsDN"=>false) \
  ])
```

The ***java.naming.factory.initial***, ***java.naming.factory.url*** and ***java.naming.security*** options in the testLDAP security domain configuration indicate the following conditions:

- The Sun LDAP JNDI provider implementation will be used
- The LDAP server is located on host **ldaphost.jboss.org** on port 1389
- The LDAP simple authentication method will be use to connect to the LDAP server.

The login module attempts to connect to the LDAP server using a Distinguished Name (DN) representing the user it is trying to authenticate. This DN is constructed from the passed principalDNPrefix, the user name of the user and the principalDNSuffix as described above. In Example 15.4, “LDIF File Example”, the user name **jsmith** would map to **uid=jsmith,ou=People,dc=jboss,dc=org**.

NOTE

The example assumes the LDAP server authenticates users using the **userPassword** attribute of the user's entry (**theduke** in this example). Most LDAP servers operate in this manner, however if your LDAP server handles authentication differently you must ensure LDAP is configured according to your production environment requirements.

Once authentication succeeds, the roles on which authorization will be based are retrieved by performing a subtree search of the **rolesCtxDN** for entries whose uidAttributeID match the user. If matchOnUserDN is true, the search will be based on the full DN of the user. Otherwise the search will be based on the actual user name entered. In this example, the search is under **ou=Roles,dc=jboss,dc=org** for any entries that have a **member** attribute equal to **uid=jsmith,ou=People,dc=jboss,dc=org**. The search would locate **cn=JBossAdmin** under the roles entry.

The search returns the attribute specified in the roleAttributeID option. In this example, the attribute is **cn**. The value returned would be **JBossAdmin**, so the **jsmith** user is assigned to the **JBossAdmin** role.

A local LDAP server often provides identity and authentication services, but is unable to use authorization services. This is because application roles do not always map well onto LDAP groups, and LDAP administrators are often hesitant to allow external application-specific data in central LDAP servers. The LDAP authentication module is often paired with another login module, such as the database login module, that can provide roles more suitable to the application being developed.

An LDAP Data Interchange Format (LDIF) file representing the structure of the directory this data operates against is shown in Example 15.4, “LDIF File Example”.

LDAP Data Interchange Format (LDIF)

Plain text data interchange format used to represent LDAP directory content and update requests. Directory content is represented as one record for each object or update request. Content consists of add, modify, delete, and rename requests.

Example 15.4. LDIF File Example

```
dn: dc=jboss,dc=org
objectclass: top
objectclass: dcObject
objectclass: organization
dc: jboss
o: JBoss

dn: ou=People,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: People

dn: uid=jsmith,ou=People,dc=jboss,dc=org
objectclass: top
objectclass: uidObject
objectclass: person
uid: jsmith
cn: John
sn: Smith
userPassword: theduke

dn: ou=Roles,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=JBossAdmin,ou=Roles,dc=jboss,dc=org
objectclass: top
objectclass: groupOfNames
cn: JBossAdmin
member: uid=jsmith,ou=People,dc=jboss,dc=org
description: the JBossAdmin group
```

Report a bug

15.1.5. LdapExtended Login Module

Distinguished Name (DN)

In Lightweight Directory Access Protocol (LDAP), the distinguished name 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.

The LdapExtended (**`org.jboss.security.auth.spi.LdapExtLoginModule)`** searches for the user to bind, as well as the associated roles, for authentication. The roles query recursively follows DNs to navigate a hierarchical role structure.

The LoginModule options include whatever options are supported by the chosen LDAP JNDI provider supports. Examples of standard property names are:

- Context.INITIAL_CONTEXT_FACTORY = "java.naming.factory.initial"

- Context.SECURITY_PROTOCOL = "java.naming.security.protocol"
- Context.PROVIDER_URL = "java.naming.provider.url"
- Context.SECURITY_AUTHENTICATION = "java.naming.security.authentication"
- Context.REFERRAL = "java.naming.referral"

Login module implementation logic follows the order below:

1. The initial LDAP server bind is authenticated using the bindDN and bindCredential properties. The bindDN is a user with permissions to search both the baseCtxDN and rolesCtxDN trees for the user and roles. The user DN to authenticate against is queried using the filter specified by the baseFilter property.

2. The resulting userDN is authenticated by binding to the LDAP server using the userDN as the InitialLdapContext environment Context.SECURITY_PRINCIPAL. The Context.SECURITY_CREDENTIALS property is either set to the String password obtained by the callback handler.

3. If this is successful, the associated user roles are queried using the rolesCtxDN, roleAttributeID, roleAttributeIsDN, roleNameAttributeID, and roleFilter options.

For details of LdapExtended login module options see Section A.1, “Included Authentication Modules”.

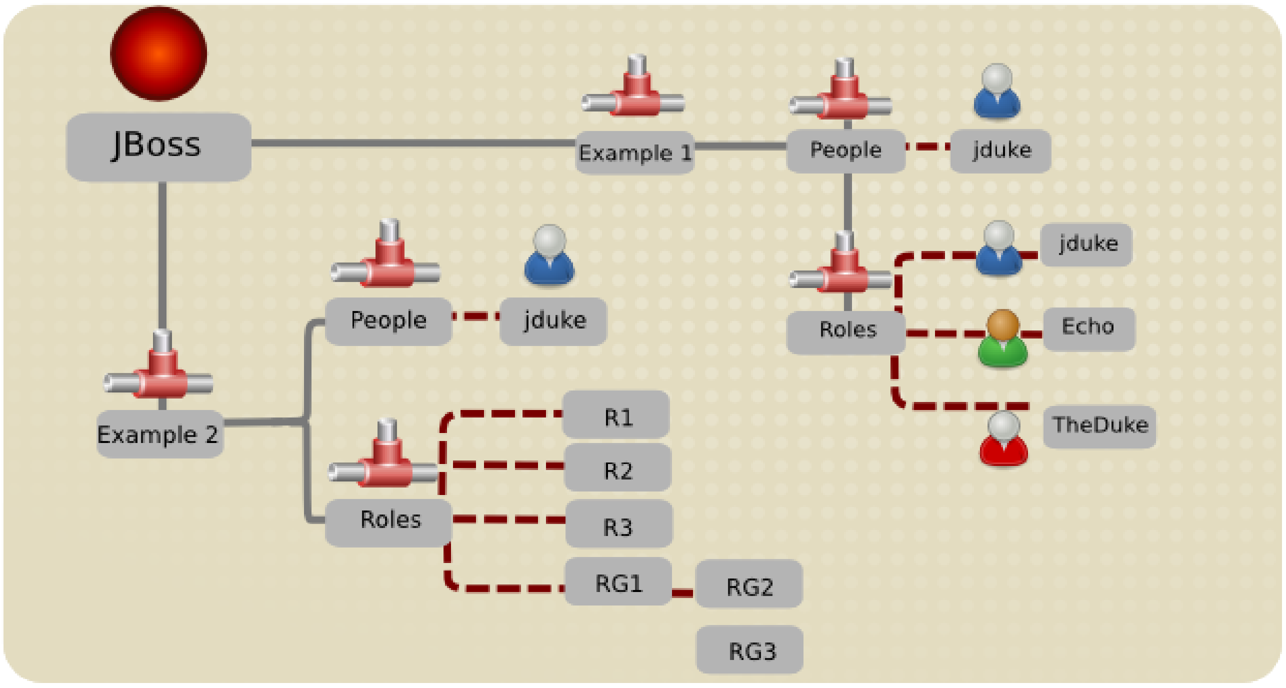


Figure 15.1. LDAP Structure Example

Example 15.5. Example 2 LDAP Configuration

```
version: 1
dn: o=example2,dc=jboss,dc=org
objectClass: top
objectClass: organization
o: example2

dn: ou=People,o=example2,dc=jboss,dc=org
```

```
objectClass: top
objectClass: organizationalUnit
ou: People

dn: uid=jduke,ou=People,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: uidObject
objectClass: person
objectClass: inetOrgPerson
cn: Java Duke
employeeNumber: judke-123
sn: Duke
uid: jduke
userPassword:: dGhlZHVrZQ==

dn: uid=jduke2,ou=People,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: uidObject
objectClass: person
objectClass: inetOrgPerson
cn: Java Duke2
employeeNumber: judke2-123
sn: Duke2
uid: jduke2
userPassword:: dGhlZHVrZTI=

dn: ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Roles

dn: uid=jduke,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupUserEx
memberOf: cn=Echo,ou=Roles,o=example2,dc=jboss,dc=org
memberOf: cn=TheDuke,ou=Roles,o=example2,dc=jboss,dc=org
uid: jduke

dn: uid=jduke2,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupUserEx
memberOf: cn=Echo2,ou=Roles,o=example2,dc=jboss,dc=org
memberOf: cn=TheDuke2,ou=Roles,o=example2,dc=jboss,dc=org
uid: jduke2

dn: cn=Echo,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: Echo
description: the echo role
member: uid=jduke,ou=People,dc=jboss,dc=org

dn: cn=TheDuke,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: TheDuke
```

```
description: the duke role
member: uid=jduke,ou=People,o=example2,dc=jboss,dc=org

dn: cn=Echo2,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: Echo2
description: the Echo2 role
member: uid=jduke2,ou=People,dc=jboss,dc=org

dn: cn=TheDuke2,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: TheDuke2
description: the duke2 role
member: uid=jduke2,ou=People,o=example2,dc=jboss,dc=org

dn: cn=JBossAdmin,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: JBossAdmin
description: the JBossAdmin group
member: uid=jduke,ou=People,dc=jboss,dc=org
```

The module configuration for this LDAP structure example is outlined in the following management CLI command.

```
/subsystem=security/security-
domain=testLdapExample2/authentication=classic/login-
module=LdapExtended:add( \
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
    ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"),
\
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("bindDN"=>"cn=Root,dc=jboss,dc=org"), \
    ("bindCredential"=>"secret1"), \
    ("baseCtxDN"=>"ou=People,o=example2,dc=jboss,dc=org"), \
    ("baseFilter"=>"(uid={0})"), \
    ("rolesCtxDN"=>"ou=Roles,o=example2,dc=jboss,dc=org"), \
    ("roleFilter"=>"(uid={0})"), \
    ("roleAttributeIsDN"=>"true"), \
    ("roleAttributeID"=>"memberOf"), \
    ("roleNameAttributeID"=>"cn") \
  ])
```

Example 15.6. Example 3 LDAP Configuration

```
dn: o=example3,dc=jboss,dc=org
objectclass: top
```

```
objectclass: organization
o: example3

dn: ou=People,o=example3,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: People

dn: uid=jduke,ou=People,o=example3,dc=jboss,dc=org
objectclass: top
objectclass: uidObject
objectclass: person
objectClass: inetOrgPerson
uid: jduke
employeeNumber: judke-123
cn: Java Duke
sn: Duke
userPassword: theduke

dn: ou=Roles,o=example3,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Roles

dn: uid=jduke,ou=Roles,o=example3,dc=jboss,dc=org
objectClass: top
objectClass: groupUserEx
memberOf: cn=Echo,ou=Roles,o=example3,dc=jboss,dc=org
memberOf: cn=TheDuke,ou=Roles,o=example3,dc=jboss,dc=org
uid: jduke

dn: cn=Echo,ou=Roles,o=example3,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: Echo
description: the JBossAdmin group
member: uid=jduke,ou=People,o=example3,dc=jboss,dc=org

dn: cn=TheDuke,ou=Roles,o=example3,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: TheDuke
member: uid=jduke,ou=People,o=example3,dc=jboss,dc=org
```

The module configuration for this LDAP structure example is outlined in the following management CLI command.

```
/subsystem=security/security-
domain=testLdapExample3/authentication=classic/login-
module=LdapExtended:add( \
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
    ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"), \
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("bindDN"=>"cn=Root,dc=jboss,dc=org"), \
    ("bindCredential"=>"secret1"), \
    ("baseCtxDN"=>"ou=People,o=example3,dc=jboss,dc=org"), \
    ("baseFilter"=>"(cn={0})"), \
    ("rolesCtxDN"=>"ou=Roles,o=example3,dc=jboss,dc=org"), \
    ("roleFilter"=>"(member={1})"), \
    ("roleAttributeID"=>"cn") \
  ])
```

Example 15.7. Example 4 LDAP Configuration

```
dn: o=example4,dc=jboss,dc=org
objectclass: top
objectclass: organization
o: example4

dn: ou=People,o=example4,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: People

dn: uid=jduke,ou=People,o=example4,dc=jboss,dc=org
objectClass: top
objectClass: uidObject
objectClass: person
objectClass: inetOrgPerson
cn: Java Duke
employeeNumber: jduke-123
sn: Duke
uid: jduke
userPassword:: dGhlZHVrZQ==

dn: ou=Roles,o=example4,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Roles

dn: cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: RG1
member: cn=empty

dn: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: RG2
member: cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
member: uid=jduke,ou=People,o=example4,dc=jboss,dc=org

dn: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
```

```
objectClass: groupOfNames
objectClass: top
cn: RG3
member: cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R1,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R1
member: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R2,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R2
member: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R3,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R3
member: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
member: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R4,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R4
member: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R5,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R5
member: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
member: uid=jduke,ou=People,o=example4,dc=jboss,dc=org
```

The module configuration for this LDAP structure example is outlined in the code sample.

```
/subsystem=security/security-
domain=testLdapExample4/authentication=classic/login-
module=LdapExtended:add( \
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
    ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"),
\
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("bindDN"=>"cn=Root,dc=jboss,dc=org"), \
    ("bindCredential"=>"secret1"), \
    ("baseCtxDN"=>"ou=People,o=example4,dc=jboss,dc=org"), \
    ("baseFilter"=>"(cn={0})"), \
    ("rolesCtxDN"=>"ou=Roles,o=example4,dc=jboss,dc=org"), \
    ("roleFilter"=>"(member={1})"), \
    ("roleRecursion"=>"1"), \
    ("roleAttributeID"=>"memberOf") \
  ])
```

Example 15.8. Default Active Directory Configuration

The example below represents the configuration for a default Active Directory configuration.

Some Active Directory configurations may require searching against the Global Catalog on port 3268 instead of the usual port 389. This is most likely when the Active Directory forest includes multiple domains.

```
/subsystem=security/security-
domain=AD_Default/authentication=classic/login-module=LdapExtended:add(
\
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
    ("bindDN"=>"JBOSS\searchuser"), \
    ("bindCredential"=>"password"), \
    ("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
    ("baseFilter"=>"(sAMAccountName={0})"), \
    ("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
    ("roleFilter"=>"(sAMAccountName={0})"), \
    ("roleAttributeID"=>"memberOf"), \
    ("roleAttributeIsDN"=>"true"), \
    ("roleNameAttributeID"=>"cn"), \
    ("searchScope"=>"ONELEVEL_SCOPE"), \
    ("allowEmptyPasswords"=>"false") \
  ])
```

Example 15.9. Recursive Roles Active Directory Configuration

The example below implements a recursive role search within Active Directory. The key difference between this example and the default Active Directory example is that the role search has been replaced to search the member attribute using the DN of the user. The login module then uses the DN of the role to find groups of which the group is a member.

```
/subsystem=security/security-
domain=AD_Recursive/authentication=classic/login-
module=LdapExtended:add( \
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
    ("java.naming.referral"=>"follow"), \
    ("bindDN"=>"JBOSS\searchuser"), \
    ("bindCredential"=>"password"), \
    ("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
    ("baseFilter"=>"(sAMAccountName={0})"), \
    ("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
    ("roleFilter"=>"(member={1})"), \
    ("roleAttributeID"=>"cn"), \
    ("roleAttributeIsDN"=>"false"), \
    ("roleRecursion"=>"2"), \
    ("searchScope"=>"ONELEVEL_SCOPE"), \
    ("allowEmptyPasswords"=>"false") \
  ])
```

Report a bug

15.1.6. UsersRoles Login Module

`UsersRoles` login module is a simple login module that supports multiple users and user roles loaded from Java properties files. The default username-to-password mapping filename is **`users.properties`** and the default username-to-roles mapping filename is **`roles.properties`**.

For details of UsersRoles login module options see Section A.1, “Included Authentication Modules” .

This login module supports password stacking, password hashing, and unauthenticated identity.

The properties files are loaded during initialization using the initialize method thread context class loader. This means that these files can be placed on the classpath of the Java EE deployment (for example, into the **`WEB-INF/classes`** folder in the **`WAR`** archive), or into any directory on the server classpath. 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.

Example 15.10. UserRoles Login Module

```
/subsystem=security/security-domain=ejb3-
sampleapp/authentication=classic/login-module=UsersRoles:add( \
  code=UsersRoles, \
  flag=required, \
  module-options=[ \
    ("usersProperties"=>"ejb3-sampleapp-users.properties"), \
    ("rolesProperties"=>"ejb3-sampleapp-roles.properties") \
  ])
```

In Example 15.10, “UserRoles Login Module”, the **`ejb3-sampleapp-users.properties`** file uses a **`username=password`** format with each user entry on a separate line:

```
username1=password1
username2=password2
...
```

The **`ejb3-sampleapp-roles.properties`** file referenced in Example 15.10, “UserRoles Login Module” uses the pattern **`username=role1,role2,`** with an optional group name value. For example:

```
username1=role1,role2,...
username1.RoleGroup1=role3,role4,...
username2=role1,role3,...
```

The user name.XXX property name pattern present in **`ejb3-sampleapp-roles.properties`** is used to assign the user name roles to a particular named group of roles where the **`XXX`** portion of the property name is the group name. The user name=... form is an abbreviation for user name.Roles=..., where the **`Roles`** group name is the standard name the **`JBossAuthorizationManager`** expects to contain the roles which define the permissions of users.

The following would be equivalent definitions for the **`jduke`** user name:

```
jduke=TheDuke,AnimatedCharacter
jduke.Roles=TheDuke,AnimatedCharacter
```

Report a bug

15.1.7. Database Login Module

The **`Database`** login module is a Java Database Connectivity-based (JDBC) login module that supports authentication and role mapping. Use this login module if you have your user name, password and role information stored in a relational database.

NOTE

This module supports password stacking, password hashing and unauthenticated identity.

The **`Database`** login module is based on two logical tables:

```
Table Principals(PrincipalID text, Password text)
Table Roles(PrincipalID text, Role text, RoleGroup text)
```

The **`Principals`** table associates the user **`PrincipalID`** with the valid password and the **`Roles`** table associates the user **`PrincipalID`** with its role sets. The roles used for user permissions must be contained in rows with a **`RoleGroup`** column value of **`Roles`**.

The tables are logical in that you can specify the SQL query that the login module uses. The only requirement is that the **`java.sql.ResultSet`** has the same logical structure as the **`Principals`** and **`Roles`** tables described previously. The actual names of the tables and columns are not relevant as the results are accessed based on the column index.

To clarify this notion, consider a database with two tables, **`Principals`** and **`Roles`**, as already declared. The following statements populate the tables with the following data:

- **`PrincipalID`** **`java`** with a **`Password`** of **`echoman`** in the **`Principals`** table
- **`PrincipalID`** **`java`** with a role named **`Echo`** in the **`RolesRoleGroup`** in the **`Roles`** table
- **`PrincipalID`** **`java`** with a role named **`caller_java`** in the **`CallerPrincipalRoleGroup`** in the **`Roles`** table

```
INSERT INTO Principals VALUES('java', 'echoman')
INSERT INTO Roles VALUES('java', 'Echo', 'Roles')
INSERT INTO Roles VALUES('java', 'caller_java', 'CallerPrincipal')
```

For details of Database login module options see Section A.1, “Included Authentication Modules”.

An example **`Database login module`** configuration could be constructed as follows:

```
CREATE TABLE Users(username VARCHAR(64) PRIMARY KEY, passwd VARCHAR(64))
CREATE TABLE UserRoles(username VARCHAR(64), role VARCHAR(32))
```

A corresponding login module configuration in a security domain:

```
/subsystem=security/security-domain=testDB/authentication=classic/login-
module=Database:add( \
  code=Database, \
  flag=required, \
  module-options=[ \
    ("dsJndiName"=>"java:/MyDatabaseDS"), \
    ("principalsQuery"=>"select passwd from Users where username=?"), \
    ("rolesQuery"=>"select role, 'Roles' from UserRoles where username=?")
\
  ])
```

Report a bug

15.1.8. 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: you must combine it with another login module capable of acquiring authorization roles to completely define access to a secured web or EJB component. Two subclasses of this login module, **`CertRolesLoginModule`** and **`DatabaseCertLoginModule`** extend the behavior to obtain the authorization roles from either a properties file or database.

For details of **`Certificate`** login module options see Section A.1, “Included Authentication Modules”.

The **`Certificate`** login module needs a **`KeyStore`** to perform user validation. This is obtained from a JSSE configuration of linked security domain as shown in the following configuration fragment:

```
/subsystem=security/security-domain=trust-domain:add
/subsystem=security/security-domain=trust-domain/jsse=classic:add( \
  truststore={ \
    password=>pass1234, \
    url=>/home/jbosseap/trusted-clients.jks \
  })

/subsystem=security/security-domain=testCert:add
/subsystem=security/security-domain=testCert/authentication=classic:add
/subsystem=security/security-domain=testCert/authentication=classic/login-
module=Certificate:add( \
  code=Certificate, \
  flag=required, \
  module-options=[ \
    ("securityDomain"=>"trust-domain"), \
  ])
```

Procedure 15.1. Secure Web Applications with Certificates and Role-based Authorization

This procedure describes how to secure a web application, such as the **`user-app.war`**, using client certificates and role-based authorization. In this example the **`CertificateRoles`** login module is used for authentication and authorization. Both the **`trusted-clients.keystore`** and the **`app-roles.properties`** require an entry that maps to the principal associated with the client certificate.

By default, the principal is created using the client certificate distinguished name, such as the DN specified in Example 15.11, “Certificate Example”.

1. **Declare Resources and Roles**
   Modify **`web.xml`** to declare the resources to be secured along with the allowed roles and security domain to be used for authentication and authorization.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        version="3.0">

        <security-constraint>
                <web-resource-collection>
                        <web-resource-name>Protect App</web-
resource-name>
                        <url-pattern>/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <role-name>Admin</role-name>
                </auth-constraint>
        </security-constraint>

        <login-config>
                <auth-method>CLIENT-CERT</auth-method>
                <realm-name>Secured area</realm-name>
        </login-config>

        <security-role>
                <role-name>Admin</role-name>
        </security-role>
</web-app>
```

2. **Specify the Security Domain**
   In the **`jboss-web.xml`** file, specify the required security domain.

```
<jboss-web>
  <security-domain>app-sec-domain</security-domain>
</jboss-web>
```

3. **Configure Login Module**
   Define the login module configuration for the **app-sec-domain** domain you just specified using the management CLI.

```
[
/subsystem=security/security-domain=trust-domain:add
/subsystem=security/security-domain=trust-domain/jsse=classic:add( \
  truststore={ \
    password=>pass1234, \
    url=>/home/jbosseap/trusted-clients.jks \
  })

/subsystem=security/security-domain=app-sec-domain:add
/subsystem=security/security-domain=app-sec-
domain/authentication=classic:add
/subsystem=security/security-domain=app-sec-
domain/authentication=classic/login-module=CertificateRoles:add( \
  code=CertificateRoles, \
  flag=required, \
  module-options=[ \
    ("securityDomain"=>"trust-domain"), \
    ("rolesProperties"=>"app-roles.properties") \
  ])
```

Example 15.11. Certificate Example

```
[conf]$ keytool -printcert -file valid-client-cert.crt
Owner: CN=valid-client, OU=Security QE, OU=JBoss, O=Red Hat, C=CZ
Issuer: CN=EAP Certification Authority, OU=Security QE, OU=JBoss, O=Red
Hat, C=CZ
Serial number: 2
Valid from: Mon Mar 24 18:21:55 CET 2014 until: Tue Mar 24 18:21:55 CET
2015
Certificate fingerprints:
         MD5:  0C:54:AE:6E:29:ED:E4:EF:46:B5:14:30:F2:E0:2A:CB
         SHA1:
D6:FB:19:E7:11:28:6C:DE:01:F2:92:2F:22:EF:BB:5D:BF:73:25:3D
         SHA256:
CD:B7:B1:72:A3:02:42:55:A3:1C:30:E1:A6:F0:20:B0:2C:0F:23:4F:7A:8E:2F:2D:
FA:AF:55:3E:A7:9B:2B:F4
         Signature algorithm name: SHA1withRSA
         Version: 3
```

The **trusted-clients.keystore** would need the certificate in Example 15.11, “Certificate Example” stored with an alias of **CN=valid-client, OU=Security QE, OU=JBoss, O=Red Hat, C=CZ**. The **app-roles.properties** must have the same entry. Since the DN contains characters that are normally treated as delimiters, you must escape the problem characters using a backslash ('****') as illustrated below.

```
# A sample app-roles.properties file
CN\=valid-client,\ OU\=Security\ QE,\ OU\=JBoss,\ O\=Red\ Hat,\ C\=CZ
```

▪

Report a bug

15.1.9. 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.

NOTE

This module supports password stacking.

This login module is useful when you need to provide a fixed identity to a service, and in development environments when you want to test the security associated with a given principal and associated roles.

For details of Identity login module options see Section A.1, “Included Authentication Modules”.

A sample security domain configuration is described below. It authenticates all users as the principal named **`jduke`** and assigns role names of **`TheDuke`**, and **`AnimatedCharacter`**:.

```
/subsystem=security/security-domain=testIdentity:add
/subsystem=security/security-
domain=testIdentity/authentication=classic:add
/subsystem=security/security-
domain=testIdentity/authentication=classic/login-module=Identity:add( \
  code=Identity, \
  flag=required, \
  module-options=[ \
    ("principal"=>"jduke"), \
    ("roles"=>"TheDuke,AnimatedCharacter") \
  ])
```

Report a bug

15.1.10. 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.

For details of RunAs login module options see Section A.1, “Included Authentication Modules”.

Report a bug

15.1.10.1. RunAsIdentity Creation

In order for JBoss EAP 6 to secure access to EJB methods, the identity of the user must be known at the time the method call is made.

A user's identity in the server is represented either by a **`javax.security.auth.Subject`** instance or an **`org.jboss.security.RunAsIdentity`** instance. Both these classes store one or more principals that represent the identity and a list of roles that the identity possesses. In the case of the **`javax.security.auth.Subject`** a list of credentials is also stored.

In the <assembly-descriptor> section of the **`ejb-jar.xml`** deployment descriptor, you specify one or more roles that a user must have to access the various EJB methods. A comparison of these lists reveals whether the user has one of the roles necessary to access the EJB method.

Example 15.12. org.jboss.security.RunAsIdentity Creation

In the **`ejb-jar.xml`** file, you specify a <security-identity> element with a <run-as> role defined as a child of the <session> element.

```
<session>
   ...
   <security-identity>
      <run-as>
         <role-name>Admin</role-name>
      </run-as>
   </security-identity>
   ...
</session>
```

This declaration signifies that an **`Admin`** RunAsIdentity role must be created.

To name a principal for the **`Admin`** role, you define a **`<run-as-principal>`** element in the **`jboss-ejb3.xml`** file.

```

<jboss:ejb-jar
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
        xmlns:s="urn:security:1.1"
        version="3.1" impl-version="2.0">
    <assembly-descriptor>
        <s:security>
            <ejb-name>WhoAmIBean</ejb-name>
            <s:run-as-principal>John</s:run-as-principal>
        </s:security>
    </assembly-descriptor>
</jboss:ejb-jar>

```

The **`<security-identity>`** element in both the **`ejb-jar.xml`** and **`<security>`** element in the **`jboss-ejb3.xml`** files are parsed at deployment time. The **`<run-as>`** role name and the **`<run-as-principal>`** name are then stored in the **`org.jboss.metadata.ejb.spec.SecurityIdentityMetaData`** class.

Example 15.13. Assigning multiple roles to a RunAsIdentity

You can assign more roles to RunAsIdentity by mapping roles to principals in the **jboss-ejb3.xml** deployment descriptor **<assembly-descriptor>** element group.

```
<jboss:ejb-jar xmlns:sr="urn:security-role"
        ...>
        <assembly-descriptor>
            ...
                <sr:security-role>
                        <sr:role-name>Support</sr:role-name>
                        <sr:principal-name>John</sr:principal-name>
                        <sr:principal-name>Jill</sr:principal-name>
                        <sr:principal-name>Tony</sr:principal-name>
                </sr:security-role>
        </assembly-descriptor>
</jboss:ejb-jar>
```

In Example 15.12, “org.jboss.security.RunAsIdentity Creation”, the **<run-as-principal>** of **John** was created. The configuration in this example extends the **Admin** role, by adding the **Support** role. The new role contains extra principals, including the originally defined principal **John**.

The **<security-role>** element in both the **ejb-jar.xml** and **jboss-ejb3.xml** files are parsed at deployment time. The **<role-name>** and the **<principal-name>** data is stored in the **org.jboss.metadata.ejb.spec.SecurityIdentityMetaData** class.

Report a bug

15.1.11. 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 PicketBox transparently) must use **Client** login module.

Note that 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. If you need to perform client-side authentication of users you would need to configure another login module in addition to the **Client** login module.

For details of Client login module options see Section A.1, “Included Authentication Modules”.

Report a bug

15.1.12. SPNEGO Login Module

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.

For details of SPNEGO login module options see Section A.1, “Included Authentication Modules”.

The JBoss Negotiation module is not included as a standard dependency for deployed applications. To use the **`SPNEGO`** or **`AdvancedLdap`** login modules in your project, then you must add the dependency manually by editing **`META-INF/jboss-deployment-structure.xml`** deployment descriptor.

Example 15.14. Add JBoss Negotiation Module as a Dependency

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.jboss.security.negotiation" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

Report a bug

15.1.13. RoleMapping Login Module

`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 "A" has the roles "ldapAdmin" 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 the user **`A`**.

For details of **`RoleMapping`** login module options see Section A.1, “Included Authentication Modules”.

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.

Example 15.15. Defining mapped roles

```
/subsystem=security/security-domain=test-domain-2/:add
/subsystem=security/security-domain=test-domain-
2/authentication=classic:add
/subsystem=security/security-domain=test-domain-
2/authentication=classic/login-module=test-2-lm/:add(\
flag=required,\
code=UsersRoles,\
module-options=[("usersProperties"=>"users.properties"),
("rolesProperties"=>"roles.properties")]\
)
/subsystem=security/security-domain=test-domain-
2/authentication=classic/login-module=test2-map/:add(\
flag=optional,\
code=RoleMapping,\
module-options=[("rolesProperties"=>"rolesMapping-roles.properties")]\
)
```

Another example achieving the same result, but using the mapping module. This is the preferred method of role mapping:

Example 15.16. Preferred method of defining mapped roles

```
/subsystem=security/security-domain=test-domain-2/:add
/subsystem=security/security-domain=test-domain-
2/authentication=classic:add
/subsystem=security/security-domain=test-domain-
2/authentication=classic/login-module=test-2-lm/:add(\
flag=required,\
code=UsersRoles,\
module-options=[("usersProperties"=>"users.properties"),
("rolesProperties"=>"roles.properties")]\
)
/subsystem=security/security-domain=test-domain-
2/mapping=classic/mapping-module=test2-map/:add(\
code=PropertiesRoles,type=role,\
module-options=[("rolesProperties"=>"rolesMapping-roles.properties")]\
)
```

Example 15.17. Properties File used by a RoleMappingLoginModule

```
ldapAdmin=admin, testAdmin
```

If the authenticated subject contains role **`ldapAdmin`**, then the roles **`admin`** and **`testAdmin`** are added to or substitute the authenticated subject depending on the replaceRole property value.

Report a bug

15.2. CUSTOM MODULES

If the login modules bundled with the PicketBox framework do not work with your security environment, you can write your own custom login module implementation. The **`AuthenticationManager`** requires a particular usage pattern of the **`Subject`** principals set. You must understand the JAAS Subject class's information storage features and the expected usage of these features to write a login module that works with the **`AuthenticationManager`**.

This section examines this requirement and introduces two abstract base **`LoginModule`** implementations that can help you implement custom login modules.

You can obtain security information associated with a **`Subject`** by using the following methods:

```
java.util.Set getPrincipals()
java.util.Set getPrincipals(java.lang.Class c)
java.util.Set getPrivateCredentials()
java.util.Set getPrivateCredentials(java.lang.Class c)
java.util.Set getPublicCredentials()
java.util.Set getPublicCredentials(java.lang.Class c)
```

For **`Subject`** identities and roles, PicketBox has selected the most logical choice: the principals sets obtained via **`getPrincipals()`** and **`getPrincipals(java.lang.Class)`**. The usage pattern is as follows:

- User identities (for example; user name, social security number, employee ID) are stored as **`java.security.Principal`** objects in the **`SubjectPrincipals`** set. The **`Principal`** implementation that represents the user identity must base comparisons and equality on the name of the principal. A suitable implementation is available as the **`org.jboss.security.SimplePrincipal`** class. Other **`Principal`** instances may be added to the **`SubjectPrincipals`** set as needed.

- Assigned user roles are also stored in the **`Principals`** set, and are grouped in named role sets using **`java.security.acl.Group`** instances. The **`Group`** interface defines a collection of **`Principal`**s and/or **`Group`**s, and is a subinterface of **`java.security.Principal`**.

- Any number of role sets can be assigned to a **`Subject`**.

- The PicketBox framework uses two well-known role sets with the names **`Roles`** and **`CallerPrincipal`**.

  - The **`Roles`** group is the collection of **`Principal`**s for the named roles as known in the application domain under which the **`Subject`** has been authenticated. This role set is used by methods like the **`EJBContext.isCallerInRole(String)`**, which EJBs can use to see if the current caller belongs to the named application domain role. The security interceptor logic that performs method permission checks also uses this role set.

  - The **`CallerPrincipal`** **`Group`** consists of the single **`Principal`** identity assigned to the user in the application domain. The **`EJBContext.getCallerPrincipal()`** method uses the **`CallerPrincipal`** to allow the application domain to map from the operation environment identity to a user identity suitable for the application. If a **`Subject`** does not have a **`CallerPrincipal`** **`Group`**, the application identity is the same as operational environment identity.

Report a bug

15.2.1. Subject Usage Pattern Support

To simplify correct implementation of the **`Subject`** usage patterns described in Section 15.2, “Custom Modules”, PicketBox includes login modules that populate the authenticated **`Subject`** with a template pattern that enforces correct **`Subject`** usage.

AbstractServerLoginModule

The most generic of the two is the **`org.jboss.security.auth.spi.AbstractServerLoginModule`** class.

It provides an implementation of the **`javax.security.auth.spi.LoginModule`** interface and offers abstract methods for the key tasks specific to an operation environment security infrastructure. The key details of the class are highlighted in Example 15.18, “AbstractServerLoginModule Class Fragment”. The JavaDoc comments detail the responsibilities of subclasses.

IMPORTANT

The **`loginOk`** instance variable is pivotal. This must be set to **`true`** if the log in succeeds, or **`false`** by any subclasses that override the log in method. If this variable is incorrectly set, the commit method will not correctly update the subject.

Tracking the log in phase outcomes allows login modules to be chained together with control flags. These control flags do not require the login modules to succeed as part of the authentication process.

Example 15.18. AbstractServerLoginModule Class Fragment

```
package org.jboss.security.auth.spi;
/**
 *  This class implements the common functionality required for a JAAS
 *  server-side LoginModule and implements the PicketBox standard
 *  Subject usage pattern of storing identities and roles. Subclass
 *  this module to create your own custom LoginModule and override the
 *  login(), getRoleSets(), and getIdentity() methods.
 */
public abstract class AbstractServerLoginModule
    implements javax.security.auth.spi.LoginModule
{
    protected Subject subject;
    protected CallbackHandler callbackHandler;
    protected Map sharedState;
    protected Map options;
    protected Logger log;

    /** Flag indicating if the shared credential should be used */
    protected boolean useFirstPass;
    /**
     * Flag indicating if the login phase succeeded. Subclasses that
     * override the login method must set this to true on successful
     * completion of login
     */
    protected boolean loginOk;

    // ...
    /**
     * Initialize the login module. This stores the subject,
     * callbackHandler and sharedState and options for the login
     * session. Subclasses should override if they need to process
     * their own options. A call to super.initialize(...)  must be
     * made in the case of an override.
     *
     * <p>
     * The options are checked for the  <em>password-stacking</em>
parameter.
     * If this is set to "useFirstPass", the login identity will be
taken from the
     * <code>javax.security.auth.login.name</code> value of the
sharedState map,
     * and the proof of identity from the
     * <code>javax.security.auth.login.password</code> value of the
sharedState map.
```

```
     *
     * @param subject the Subject to update after a successful login.
     * @param callbackHandler the CallbackHandler that will be used to
obtain the
     * the user identity and credentials.
     * @param sharedState a Map shared between all configured login
module instances
     * @param options the parameters passed to the login module.
     */
    public void initialize(Subject subject,
                           CallbackHandler callbackHandler,
                           Map sharedState,
                           Map options)
    {
        // ...
    }


    /**
     *  Looks for javax.security.auth.login.name and
     *  javax.security.auth.login.password values in the sharedState
     *  map if the useFirstPass option was true and returns true if
     *  they exist. If they do not or are null this method returns
     *  false.
     *  Note that subclasses that override the login method
     *  must set the loginOk var to true if the login succeeds in
     *  order for the commit phase to populate the Subject. This
     *  implementation sets loginOk to true if the login() method
     *  returns true, otherwise, it sets loginOk to false.
     */
    public boolean login()
        throws LoginException
    {
        // ...
    }

    /**
     *  Overridden by subclasses to return the Principal that
     *  corresponds to the user primary identity.
     */
    abstract protected Principal getIdentity();

    /**
     *  Overridden by subclasses to return the Groups that correspond
     *  to the role sets assigned to the user. Subclasses should
     *  create at least a Group named "Roles" that contains the roles
     *  assigned to the user.  A second common group is
     *  "CallerPrincipal," which provides the application identity of
     *  the user rather than the security domain identity.
     *
     *  @return Group[] containing the sets of roles
     */
    abstract protected Group[] getRoleSets() throws LoginException;
}
```

UsernamePasswordLoginModule

The second abstract base login module suitable for custom login modules is the **`org.jboss.security.auth.spi.UsernamePasswordLoginModule`**.

This login module further simplifies custom login module implementation by enforcing a string-based user name as the user identity and a **`char[]`** password as the authentication credentials. It also supports the mapping of anonymous users (indicated by a null user name and password) to a principal with no roles. The key details of the class are highlighted in the following class fragment. The JavaDoc comments detail the responsibilities of subclasses.

Example 15.19. UsernamePasswordLoginModule Class Fragment

```
package org.jboss.security.auth.spi;

/**
 *  An abstract subclass of AbstractServerLoginModule that imposes a
 *  an identity == String username, credentials == String password
 *  view on the login process. Subclasses override the
 *  getUsersPassword() and getUsersRoles() methods to return the
 *  expected password and roles for the user.
 */
public abstract class UsernamePasswordLoginModule
    extends AbstractServerLoginModule
{
    /** The login identity */
    private Principal identity;
    /** The proof of login identity */
    private char[] credential;
    /** The principal to use when a null username and password are seen
*/
    private Principal unauthenticatedIdentity;

    /**
     * The message digest algorithm used to hash passwords. If null then
     * plain passwords will be used. */
    private String hashAlgorithm = null;

    /**
     *  The name of the charset/encoding to use when converting the
     * password String to a byte array. Default is the platform's
     * default encoding.
     */
     private String hashCharset = null;

    /** The string encoding format to use. Defaults to base64. */
    private String hashEncoding = null;

    // ...

    /**
     *  Override the superclass method to look for an
     *  unauthenticatedIdentity property. This method first invokes
     *  the super version.
     *
     *  @param options,
     *  @option unauthenticatedIdentity: the name of the principal to
     *  assign and authenticate when a null username and password are
     *  seen.
     */
    public void initialize(Subject subject,
                           CallbackHandler callbackHandler,
                           Map sharedState,
                           Map options)
    {
        super.initialize(subject, callbackHandler, sharedState,
                         options);
        // Check for unauthenticatedIdentity option.
        Object option = options.get("unauthenticatedIdentity");
        String name = (String) option;
        if (name != null) {
            unauthenticatedIdentity = new SimplePrincipal(name);
        }
    }

    // ...

    /**
     *  A hook that allows subclasses to change the validation of the
     *  input password against the expected password. This version
     *  checks that neither inputPassword or expectedPassword are null
     *  and that inputPassword.equals(expectedPassword) is true;
     *
     *  @return true if the inputPassword is valid, false otherwise.
     */
    protected boolean validatePassword(String inputPassword,
                                       String expectedPassword)
    {
        if (inputPassword == null || expectedPassword == null) {
            return false;
        }
        return inputPassword.equals(expectedPassword);
    }

    /**
     *  Get the expected password for the current username available
     * via the getUsername() method. This is called from within the
     * login() method after the CallbackHandler has returned the
     * username and candidate password.
     *
     * @return the valid password String
     */
    abstract protected String getUsersPassword()
        throws LoginException;
}
```

Subclassing Login Modules

The choice of sub-classing the **`AbstractServerLoginModule`** versus **`UsernamePasswordLoginModule`** is based on whether a string-based user name and credentials are usable for the authentication technology you are writing the login module for. If the string-based semantic is valid, then subclass **`UsernamePasswordLoginModule`**, otherwise subclass **`AbstractServerLoginModule`**.

Subclassing Steps

The steps your custom login module must execute depend on which base login module class you choose. When writing a custom login module that integrates with your security infrastructure, you should start by sub-classing **`AbstractServerLoginModule`** or **`UsernamePasswordLoginModule`** to ensure that your login module provides the authenticated **`Principal`** information in the form expected by the PicketBox security manager.

When sub-classing the **`AbstractServerLoginModule`**, you must override the following:

- **`void initialize(Subject, CallbackHandler, Map, Map)`**: if you have custom options to parse.
- **`boolean login()`**: to perform the authentication activity. Be sure to set the **`loginOk`** instance variable to true if log in succeeds, false if it fails.
- **`Principal getIdentity()`**: to return the **`Principal`** object for the user authenticated by the **`log()`** step.
- **`Group[] getRoleSets()`**: to return at least one **`Group`** named **`Roles`** that contains the roles assigned to the **`Principal`** authenticated during **`login()`**. A second common **`Group`** is named **`CallerPrincipal`** and provides the user's application identity rather than the security domain identity.

When sub-classing the **`UsernamePasswordLoginModule`**, you must override the following:

- **`void initialize(Subject, CallbackHandler, Map, Map)`**: if you have custom options to parse.
- **`Group[] getRoleSets()`**: to return at least one **`Group`** named **`Roles`** that contains the roles assigned to the **`Principal`** authenticated during **`login()`**. A second common **`Group`** is named **`CallerPrincipal`** and provides the user's application identity rather than the security domain identity.
- **`String getUsersPassword()`**: to return the expected password for the current user name available via the **`getUsername()`** method. The **`getUsersPassword()`** method is called from within **`login()`** after the **`callbackhandler`** returns the user name and candidate password.

Report a bug

15.2.2. Custom LoginModule Example

The following information will help you to create a custom Login Module example that extends the **`UsernamePasswordLoginModule`** and obtains a user's password and role names from a JNDI lookup.

At the end of this section you will have created a custom JNDI context login module that will return a user's password if you perform a lookup on the context using a name of the form **`password/<username>`** (where **`<username>`** is the current user being authenticated). Similarly, a lookup of the form **`roles/<username>`** returns the requested user's roles. In Example 15.20, “JndiUserAndPassLoginModule Custom Login Module” is the source code for the **`JndiUserAndPassLoginModule`** custom login module.

Note that because this extends the JBoss **`UsernamePasswordLoginModule`**, the

`JndiUserAndPassLoginModule` obtains the user's password and roles from the JNDI store. The **`JndiUserAndPassLoginModule`** does not interact with the JAAS LoginModule operations.

Example 15.20. JndiUserAndPassLoginModule Custom Login Module

```
package org.jboss.book.security.ex2;

import java.security.acl.Group;
import java.util.Map;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;
import org.jboss.logging.Logger;
import org.jboss.security.SimpleGroup;
import org.jboss.security.SimplePrincipal;
import org.jboss.security.auth.spi.UsernamePasswordLoginModule;
/**
 * An example custom login module that obtains passwords and roles for a
user from a JNDI lookup.
 *
 * @author Scott.Stark@jboss.org
 */
public class JndiUserAndPassLoginModule extends
UsernamePasswordLoginModule {
  /** The JNDI name to the context that handles the password/username
lookup */
  private String userPathPrefix;
  /** The JNDI name to the context that handles the roles/username
lookup */
  private String rolesPathPrefix;
  private static Logger log =
Logger.getLogger(JndiUserAndPassLoginModule.class);
  /**
   * Override to obtain the userPathPrefix and rolesPathPrefix options.
   */
  @Override
  public void initialize(Subject subject, CallbackHandler
callbackHandler, Map sharedState, Map options) {
    super.initialize(subject, callbackHandler, sharedState, options);
    userPathPrefix = (String) options.get("userPathPrefix");
    rolesPathPrefix = (String) options.get("rolesPathPrefix");
  }
  /**
   * Get the roles the current user belongs to by querying the
rolesPathPrefix + '/' + super.getUsername() JNDI location.
   */
  @Override
  protected Group[] getRoleSets() throws LoginException {
    try {
      InitialContext ctx = new InitialContext();
      String rolesPath = rolesPathPrefix + '/' + super.getUsername();
      String[] roles = (String[]) ctx.lookup(rolesPath);
      Group[] groups = { new SimpleGroup("Roles") };
      log.info("Getting roles for user=" + super.getUsername());
```

```
        for (int r = 0; r < roles.length; r++) {
          SimplePrincipal role = new SimplePrincipal(roles[r]);
          log.info("Found role=" + roles[r]);
          groups[0].addMember(role);
        }
        return groups;
      } catch (NamingException e) {
        log.error("Failed to obtain groups for user=" +
super.getUsername(), e);
        throw new LoginException(e.toString(true));
      }
    }
    /**
     * Get the password of the current user by querying the userPathPrefix
+ '/' + super.getUsername() JNDI location.
     */
    @Override
    protected String getUsersPassword() throws LoginException {
      try {
        InitialContext ctx = new InitialContext();
        String userPath = userPathPrefix + '/' + super.getUsername();
        log.info("Getting password for user=" + super.getUsername());
        String passwd = (String) ctx.lookup(userPath);
        log.info("Found password=" + passwd);
        return passwd;
      } catch (NamingException e) {
        log.error("Failed to obtain password for user=" +
super.getUsername(), e);
        throw new LoginException(e.toString(true));
      }
    }
}
```

Example 15.21. Definition of security-ex2 security domain with the newly-created custom login module

```
/subsystem=security/security-domain=security-ex2/:add
/subsystem=security/security-domain=security-
ex2/authentication=classic:add
/subsystem=security/security-domain=security-
ex2/authentication=classic/login-module=ex2/:add(\
flag=required,\
code=org.jboss.book.security.ex2.JndiUserAndPassLoginModule,\
module-options=[("userPathPrefix"=>"/security/store/password"),\
("rolesPathPrefix"=>"/security/store/roles")]\
)
```

The choice of using the **`JndiUserAndPassLoginModule`** custom login module for the server side authentication of the user is determined by the login configuration for the example security domain. The EJB JAR **`META-INF/jboss-ejb3.xml`** descriptor sets the security domain. For a web application it is part of the **`WEB-INF/jboss-web.xml`** file.

Example 15.22. `jboss-ejb3.xml` Example

```
<?xml version="1.0"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:s="urn:security"
version="3.1" impl-version="2.0">
  <assembly-descriptor>
    <s:security>
      <ejb-name>*</ejb-name>
      <s:security-domain>security-ex2</s:security-domain>
    </s:security>
  </assembly-descriptor>
</jboss:ejb-jar>
```

Example 15.23. `jboss-web.xml example`

```
<?xml version="1.0"?>
<jboss-web>
    <security-domain>security-ex2</security-domain>
</jboss-web>
```

Report a bug

CHAPTER 16. SINGLE SIGN ON (SSO)

16.1. ABOUT SINGLE SIGN ON (SSO) FOR WEB APPLICATIONS

Overview

Single Sign On (SSO) allows authentication to one resource to implicitly authorize access to other resources.

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. Clustered SSO data can be shared between applications in multiple virtual hosts, and is resilient to failover. In addition, clustered SSO is able to receive requests from a load balancer.

How SSO Works

If a resource is unprotected, a user is not challenged to authenticate at all. If a user accesses a protected resource, the user is required to authenticate.

Upon successful authentication, the roles associated with the user are stored and used for authorization of all other associated resources.

If the user logs out of an application, or an application invalidates the session programmatically, all persisted authorization data is removed, and the process starts over.

A session timeout does not invalidate the SSO session if other sessions are still valid.

Limitations of SSO

No propagation across third-party boundaries.

SSO can only be used between applications deployed within JBoss EAP 6 containers.

Container-managed authentication only.

You must use container-managed authentication elements such as **`<login-config>`** in your application's **`web.xml`**.

Requires cookies.

SSO is maintained via browser cookies and URL rewriting is not supported.

Realm and security-domain limitations

Unless the **`requireReauthentication`** parameter is set to **`true`**, all web applications configured for the same SSO valve must share the same Realm configuration in **`web.xml`** and the same security domain.

You can nest the Realm element inside the Host element or the surrounding Engine element, but not inside a context.xml element for one of the involved web applications.

The **`<security-domain>`** configured in the **`jboss-web.xml`** must be consistent across all web applications.

All security integrations must accept the same credentials (for instance, username and password).

Report a bug

16.2. ABOUT CLUSTERED SINGLE SIGN ON (SSO) FOR WEB APPLICATIONS

Single Sign On (SSO) is the ability for users to authenticate to a single web application, and by means of a successful authentication, to be granted authorization to multiple other applications. Clustered SSO stores the authentication and authorization information in a clustered cache. This allows for applications on multiple different servers to share the information, and also makes the information resilient to a failure of one of the hosts.

A SSO configuration is called a valve. A valve is connected to a security domain, which is configured at the level of the server or server group. Each application which should share the same cached authentication information is configured to use the same valve. This configuration is done in the application's **`jboss-web.xml`**.

Some common SSO valves supported by the web subsystem of JBoss EAP 6 include:

- Apache Tomcat ClusteredSingleSignOn
- Apache Tomcat IDPWebBrowserSSOValve
- SPNEGO-based SSO provided by PicketLink

Depending on the specific type of valve, you may need to do some additional configuration in your security domain, in order for your valve to work properly.

Report a bug

16.3. CHOOSE THE RIGHT SSO IMPLEMENTATION

JBoss EAP 6 runs Java Enterprise Edition (EE) applications, which may be web applications, EJB applications, web services, or other types. Single Sign On (SSO) allows you to propagate security context and identity information between these applications. Several SSO solutions are available but choosing the right solution depends on your requirements.

Note that there is a distinct difference between clustered web application sessions and clustered web application SSO. Clustered web application sessions ensures that client sessions of distributable applications are distributed across the nodes in a cluster. Clustered web application SSO ensures that SSO sessions are distributed across the nodes in a cluster. Although they are separate, they are commonly used together.

Kerberos-Based Desktop SSO

If your organization already uses a Kerberos-based authentication and authorization system, such as Microsoft Active Directory, you can use the same systems to transparently authenticate to your enterprise applications running on JBoss EAP 6.

Non-Clustered Web Application SSO

If you are running multiple applications on a single instance and need to enable SSO session replication for those applications, non-clustered SSO will meet your requirements.

Clustered Web Application SSO

If you are running either a single application, or multiple applications, across a cluster and need to enable SSO session replication for those applications, clustered SSO will meet your requirements.

Report a bug

16.4. USE SINGLE SIGN ON (SSO) IN A WEB APPLICATION

Overview

Single Sign On (SSO) capabilities are provided by the web and Infinispan subsystems. Use this procedure to configure SSO in web applications.

Prerequisites

- You need to have a configured security domain which handles authentication and authorization.
- The **infinispan** subsystem needs to be present. It is present in the **full-ha** profile for a managed domain, or by using the **standalone-full-ha.xml** configuration in a standalone server.
- The **web cache-container** and SSO cache-container must each be present. The initial configuration files already contain the **web** cache-container, and some of the configurations already contain the SSO cache-container as well. Use the following commands to check for and enable the SSO cache container. Note that these commands modify the **ha** profile of a managed domain. You can change the commands to use a different profile, or remove the **/profile=ha** portion of the command, for a standalone server.

Example 16.1. Check for the web cache-container

The profiles and configurations mentioned above include the **web** cache-container by default. Use the following command to verify its presence. If you use a different profile, substitute its name instead of **ha**.

```
/profile=ha/subsystem=infinispan/cache-container=web/:read-
resource(recursive=false,proxies=false,include-
runtime=false,include-defaults=true)
```

If the result is **success** the subsystem is present. Otherwise, you need to add it.

Example 16.2. Add the web cache-container

Use the following three commands to enable the **web** cache-container to your configuration. Modify the name of the profile as appropriate, as well as the other parameters. The parameters here are the ones used in a default configuration.

```
/profile=ha/subsystem=infinispan/cache-container=web:add(aliases=
["standard-session-cache"],default-
cache="repl",module="org.jboss.as.clustering.web.infinispan")
```

```
/profile=ha/subsystem=infinispan/cache-
container=web/transport=TRANSPORT:add(lock-timeout=60000)
```

```
/profile=ha/subsystem=infinispan/cache-container=web/replicated-
cache=repl:add(mode="ASYNC",batching=true)
```

-

Example 16.3. Check for the `SSO` cache-container

Run the following Management CLI command:

```
/profile=ha/subsystem=infinispan/cache-container=web/:read-
resource(recursive=true,proxies=false,include-
runtime=false,include-defaults=true)
```

Look for output like the following: **`"sso" => {`**

If you do not find it, the SSO cache-container is not present in your configuration.

Example 16.4. Add the `SSO` cache-container

```
/profile=ha/subsystem=infinispan/cache-container=web/replicated-
cache=sso:add(mode="SYNC", batching=true)
```

- The **`web`** subsystem needs to be configured to use SSO. The following command enables SSO on the virtual server called **`default-host`**, and the cookie domain **`domain.com`**. The cache name is **`sso`**, and reauthentication is disabled.

```
/profile=ha/subsystem=web/virtual-server=default-
host/sso=configuration:add(cache-container="web",cache-
name="sso",reauthenticate="false",domain="domain.com")
```

- Each application which will share the SSO information needs to be configured to use the same <security-domain> in its **`jboss-web.xml`** deployment descriptor and the same Realm in its **`web.xml`** configuration file.

Differences Between Clustered and Non-Clustered SSO Valves

The SSO Valves are configured automatically when **`sso`** is configured under the web subsystem in a server profile. The **`ClusteredSingleSignOn`** version is used when attribute **`cache-container`** is present, otherwise standard **`SingleSignOn`** class is used.

Example 16.5. Example Clustered SSO Configuration

```
/subsystem=web/virtual-server=default-host/sso=configuration:add(cache-
container="web",cache-
name="sso",reauthenticate="false",domain="domain.com")
```

Example 16.6. Example Non-Clustered SSO Configuration

```
/subsystem=web/virtual-server=default-
host/sso=configuration:add(reauthenticate="false")
```

Invalidate a Session

An application can programmatically invalidate a session by invoking method **`javax.servlet.http.HttpSession.invalidate()`**.

Report a bug

16.5. ABOUT 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, you need to obtain a ticket from the Ticket Granting Service (TGS), which is a service running on a server on your network. After obtaining the ticket, you request a Service Ticket (ST) from an Authentication Service (AS), which is another service running on your network. You then use the ST to authenticate to the service you want to use. 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 environment, and is rarely 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.

Report a bug

16.6. ABOUT SPNEGO

Simple and Protected GSS_API Negotiation Mechanism (SPNEGO) provides a mechanism for extending a Kerberos-based Single Sign On (SSO) environment for use in Web applications.

When an application on a client computer, such as a web browser, attempts to access a protect page on the 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.

Report a bug

16.7. ABOUT MICROSOFT ACTIVE DIRECTORY

Microsoft Active Directory 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 in the Microsoft Windows Server is referred to as the domain controller. Red Hat Enterprise Linux servers running the Samba service can also act as the domain controller in this type of network.

Active Directory relies on three core technologies which work together:

- Lightweight Directory Access Protocol (LDAP), for storing information about users, computers, passwords, and other resources.
- Kerberos, for providing secure authentication over the network.

- Domain Name Service (DNS) for providing mappings between IP addresses and host names of computers and other devices on the network.

Report a bug

16.8. CONFIGURE KERBEROS OR MICROSOFT ACTIVE DIRECTORY DESKTOP SSO FOR WEB APPLICATIONS

Introduction

To authenticate your web or EJB applications using your organization's existing Kerberos-based authentication and authorization infrastructure, such as Microsoft Active Directory, you can use the JBoss Negotation capabilities built into JBoss EAP 6. If you configure your web application properly, a successful desktop or network login is sufficient to transparently authenticate against your web application, so no additional login prompt is required.

Difference from Previous Versions of the Platform

There are a few noticeable differences between JBoss EAP 6 and earlier versions:

- Security domains are configured centrally, for each profile of a managed domain, or for each standalone server. They are not part of the deployment itself. The security domain a deployment should use is named in the deployment's **`jboss-web.xml`** or **`jboss-ejb3.xml`** file.

- Security properties are configured as part of the security domain, as part of its central configuration. They are not part of the deployment.

- You can no longer override the authenticators as part of your deployment. However, you can add a NegotiationAuthenticator valve to your **`jboss-web.xml`** descriptor to achieve the same effect. The valve still requires the **`<security-constraint>`** and **`<login-config>`** elements to be defined in the **`web.xml`**. These are used to decide which resources are secured. However, the chosen auth-method will be overridden by the NegotiationAuthenticator valve in the **`jboss-web.xml`**.

- The **`CODE`** attributes in security domains now use a simple name instead of a fully-qualified class name. The following table shows the mappings between the classes used for JBoss Negotiation, and their classes.

Table 16.1. Login Module Codes and Class Names

Simple Name	Class Name	Purpose
Kerberos	com.sun.security.auth.module.Krb5LoginModule	Kerberos login module
SPNEGO	org.jboss.security.negotiation.spnego.SPNEGOLoginModule	The mechanism which enables your Web applications to authenticate to your Kerberos authentication server.
AdvancedLdap	org.jboss.security.negotiation.AdvancedLdapLoginModule	Used with LDAP servers other than Microsoft Active Directory.

Simple Name	Class Name	Purpose
AdvancedAdLdap	org.jboss.security.negotiation.AdvancedADLoginModule	Used with Microsoft Active Directory LDAP servers.

Jboss Negotiation Toolkit

The **`JBoss Negotiation Toolkit`** is a debugging tool which is available for download from https://community.jboss.org/servlet/JiveServlet/download/16876-2-34629/jboss-negotiation-toolkit.war. It is provided as an extra tool to help you to debug and test the authentication mechanisms before introducing your application into production. It is an unsupported tool, but is considered to be very helpful, as SPNEGO can be difficult to configure for web applications.

Procedure 16.1. Setup SSO Authentication for your Web or EJB Applications

1. **Configure one security domain to represent the identity of the server. Set system properties if necessary.**
   The first security domain authenticates the container itself to the directory service. It needs to use a login module which accepts some type of static login mechanism, because a real user is not involved. This example uses a static principal and references a keytab file which contains the credential.

   The XML code is given here for clarity, but you should use the Management Console or Management CLI to configure your security domains.

```
<security-domain name="host" cache-type="default">
   <authentication>
      <login-module code="Kerberos" flag="required">
         <module-option name="storeKey" value="true"/>
         <module-option name="useKeyTab" value="true"/>
         <module-option name="principal"
value="host/testserver@MY_REALM"/>
         <module-option name="keyTab"
value="/home/username/service.keytab"/>
         <module-option name="doNotPrompt" value="true"/>
         <module-option name="debug" value="false"/>
      </login-module>
   </authentication>
</security-domain>
```

2. **Configure a second security domain to secure the web application or applications. Set system properties if necessary.**
   The second security domain is used to authenticate the individual user to the Kerberos or SPNEGO authentication server. You need at least one login module to authenticate the user, and another to search for the roles to apply to the user. The following XML code shows an example SPNEGO security domain. It includes an authorization module to map roles to individual users. You can also use a module which searches for the roles on the authentication server itself.

```
<security-domain name="SPNEGO" cache-type="default">
   <authentication>
      <!-- Check the username and password -->
      <login-module code="SPNEGO"  flag="requisite">
```

```
         <module-option name="password-stacking"
value="useFirstPass"/>
         <module-option name="serverSecurityDomain" value="host"/>
      </login-module>
      <!-- Search for roles -->
      <login-module code="UsersRoles" flag="required">
         <module-option name="password-stacking"
value="useFirstPass" />
         <module-option name="usersProperties" value="spnego-
users.properties" />
         <module-option name="rolesProperties" value="spnego-
roles.properties" />
      </login-module>
   </authentication>
</security-domain>
```

3. **Specify the security-constraint and login-config in the `web.xml`**
   The **`web.xml`** descriptor contain information about security constraints and login configuration. The following are example values for each.

```
<security-constraint>
   <display-name>Security Constraint on Conversation</display-name>
   <web-resource-collection>
      <web-resource-name>examplesWebApp</web-resource-name>
      <url-pattern>/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
   <role-name>RequiredRole</role-name>
   </auth-constraint>
</security-constraint>

<login-config>
   <auth-method>SPNEGO</auth-method>
   <realm-name>SPNEGO</realm-name>
</login-config>

<security-role>
   <description> role required to log in to the
Application</description>
   <role-name>RequiredRole</role-name>
</security-role>
```

4. **Specify the security domain and other settings in the `jboss-web.xml` descriptor.**
   Specify the name of the client-side security domain (the second one in this example) in the **`jboss-web.xml`** descriptor of your deployment, to direct your application to use this security domain.

   You can no longer override authenticators directly. Instead, you can add the NegotiationAuthenticator as a valve to your **`jboss-web.xml`** descriptor, if you need to. The **`<jacc-star-role-allow>`** allows you to use the asterisk (*) character to match multiple role names, and is optional.

```
<jboss-web>
```

```
   <security-domain>java:/jaas/SPNEGO</security-domain>
   <valve>
      <class-
name>org.jboss.security.negotiation.NegotiationAuthenticator</class-
name>
   </valve>
   <jacc-star-role-allow>true</jacc-star-role-allow>
</jboss-web>
```

5. **Add a dependency to your application's `MANIFEST.MF`, to locate the Negotiation classes.**
   The web application needs a dependency on class **`org.jboss.security.negotiation`** to be added to the deployment's **`META-INF/MANIFEST.MF`** manifest, in order to locate the JBoss Negotiation classes. The following shows a properly-formatted entry.

```
Manifest-Version: 1.0
Build-Jdk: 1.6.0_24
Dependencies: org.jboss.security.negotiation
```

Result

Your web application accepts and authenticates credentials against your Kerberos, Microsoft Active Directory, or other SPNEGO-compatible directory service. If the user runs the application from a system which is already logged into the directory service, and where the required roles are already applied to the user, the web application does not prompt for authentication, and SSO capabilities are achieved.

Report a bug

CHAPTER 17. ROLE-BASED SECURITY IN APPLICATIONS

17.1. JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)

Java Authentication and Authorization Service (JAAS) is a 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 Enterprise Edition access control architecture to support user-based authorization.

In JBoss EAP 6, JAAS only provides declarative role-based security. For more information about declarative security, refer to Section 2.1, “About Declarative Security”.

JAAS is independent of any underlying authentication technologies, such as Kerberos or LDAP. You can change your underlying security structure without changing your application. You only need to change the JAAS configuration.

Report a bug

17.2. ABOUT JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)

The security architecture of JBoss EAP 6 is comprised of the security configuration subsystem, and application-specific security configurations which are included in several configuration files within the application.

Domain, Server Group, and Server Specific Configuration

Server groups (in a managed domain) and servers (in a standalone server) include the configuration for security domains. A security domain includes information about a combination of authentication, authorization, mapping, and auditing modules, with configuration details. An application specifies which security domain it requires, by name, in its **`jboss-web.xml`**.

Application-specific Configuration

Application-specific configuration takes place in one or more of the following four files.

Table 17.1. Application-Specific Configuration Files

File	Description
ejb-jar.xml	The deployment descriptor for an Enterprise JavaBean (EJB) application, located in the **`META-INF`** directory of the archive. Use the **`ejb-jar.xml`** to specify roles and map them to principals, at the application level. You can also limit specific methods and classes to certain roles. It is also used for other EJB-specific configuration not related to security.

File	Description
web.xml	The deployment descriptor for a Java Enterprise Edition (EE) web application. Use the **`web.xml`** to declare the security domain the application uses for authentication and authorization, as well as resource and transport constraints for the application, such as limiting which types of HTTP requests are allowed. You can also configure simple web-based authentication in this file. It is also used for other application-specific configuration not related to security.
jboss-ejb3.xml	Contains JBoss-specific extensions to the **`ejb-jar.xml`** descriptor.
jboss-web.xml	Contains JBoss-specific extensions to the **`web.xml`** descriptor.

NOTE

The **`ejb-jar.xml`** and **`web.xml`** are defined in the Java Enterprise Edition (Java EE) specification. The **`jboss-ejb3.xml`** provides JBoss-specific extensions for the **`ejb-jar.xml`**, and the **`jboss-web.xml`** provides JBoss-specific extensions for the **`web.xml`**.

Report a bug

17.3. USE A SECURITY DOMAIN IN YOUR APPLICATION

Overview

To use a security domain in your application, first you need to define the security domain in the server's configuration and then enable it for an application in the application's deployment descriptor. Then you must add the required annotations to the EJB that uses it. This topic covers the steps required to use a security domain in your application.

WARNING

If an application is part of a security domain that uses an authentication cache, user authentications for that application will also be available to other applications in that security domain.

Procedure 17.1. Configure Your Application to Use a Security Domain

1. **Define the Security Domain**

You need to define the security domain in the server's configuration file, and then enable it for an application in the application's descriptor file.

a. **Configure the security domain in the server's configuration file**
The security domain is configured in the **`security`** subsystem of the server's configuration file. If the JBoss EAP 6 instance is running in a managed domain, this is the **`domain/configuration/domain.xml`** file. If the JBoss EAP 6 instance is running as a standalone server, this is the **`standalone/configuration/standalone.xml`** file.

The **`other`**, **`jboss-web-policy`**, and **`jboss-ejb-policy`** security domains are provided by default in JBoss EAP 6. The following XML example was copied from the **`security`** subsystem in the server's configuration file.

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
    <security-domains>
        <security-domain name="other" cache-type="default">
            <authentication>
                <login-module code="Remoting" flag="optional">
                    <module-option name="password-stacking"
value="useFirstPass"/>
                </login-module>
                <login-module code="RealmDirect"
flag="required">
                    <module-option name="password-stacking"
value="useFirstPass"/>
                </login-module>
            </authentication>
        </security-domain>
        <security-domain name="jboss-web-policy" cache-
type="default">
            <authorization>
                <policy-module code="Delegating"
flag="required"/>
            </authorization>
        </security-domain>
        <security-domain name="jboss-ejb-policy" cache-
type="default">
            <authorization>
                <policy-module code="Delegating"
flag="required"/>
            </authorization>
        </security-domain>
    </security-domains>
</subsystem>
```

You can configure additional security domains as needed using the Management Console or CLI.

b. **Enable the security domain in the application's descriptor file**
The security domain is specified in the **`<security-domain>`** child element of the **`<jboss-web>`** element in the application's **`WEB-INF/jboss-web.xml`** file. The following example configures a security domain named **`my-domain`**.

```
<jboss-web>
    <security-domain>my-domain</security-domain>
</jboss-web>
```

This is only one of many settings which you can specify in the **WEB-INF/jboss-web.xml** descriptor.

2. **Add the Required Annotation to the EJB**
You configure security in the EJB using the **@SecurityDomain** and **@RolesAllowed** annotations. The following EJB code example limits access to the **other** security domain by users in the **guest** role.

```
package example.ejb3;

import java.security.Principal;

import javax.annotation.Resource;
import javax.annotation.security.RolesAllowed;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;

import org.jboss.ejb3.annotation.SecurityDomain;

/**
 * Simple secured EJB using EJB security annotations
 * Allow access to "other" security domain by users in a "guest"
role.
 */
@Stateless
@RolesAllowed({ "guest" })
@SecurityDomain("other")
public class SecuredEJB {

   // Inject the Session Context
   @Resource
   private SessionContext ctx;

   /**
    * Secured EJB method using security annotations
    */
   public String getSecurityInfo() {
      // Session context injected using the resource annotation
      Principal principal = ctx.getCallerPrincipal();
      return principal.toString();
   }
}
```

For more code examples, see the **ejb-security** quickstart in the JBoss EAP 6 Quickstarts bundle, which is available from the Red Hat Customer Portal.

Report a bug

17.4. USE ROLE-BASED SECURITY IN SERVLETS

To add security to a servlet, you map each servlet to a URL pattern, and create security constraints on the URL patterns which need to be secured. The security constraints limit access to the URLs to roles. The authentication and authorization are handled by the security domain specified in the WAR's **`jboss-web.xml`**.

Prerequisites

Before you use role-based security in a servlet, the security domain used to authenticate and authorize access needs to be configured in the JBoss EAP 6 container.

Procedure 17.2. Add Role-Based Security to Servlets

1. **Add mappings between servlets and URL patterns.**
   Use **`<servlet-mapping>`** elements in the **`web.xml`** to map individual servlets to URL patterns. The following example maps the servlet called **`DisplayOpResult`** to the URL pattern **`/DisplayOpResult`**.

   ```
   <servlet-mapping>
       <servlet-name>DisplayOpResult</servlet-name>
       <url-pattern>/DisplayOpResult</url-pattern>
   </servlet-mapping>
   ```

2. **Add security constraints to the URL patterns.**
   To map the URL pattern to a security constraint, use a **`<security-constraint>`**. The following example constrains access from the URL pattern **`/DisplayOpResult`** to be accessed by principals with the role **`eap_admin`**. The role needs to be present in the security domain.

   ```
   <security-constraint>
    <display-name>Restrict access to role eap_admin</display-name>
    <web-resource-collection>
     <web-resource-name>Restrict access to role eap_admin</web-
   resource-name>
     <url-pattern>/DisplayOpResult/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
     <role-name>eap_admin</role-name>
    </auth-constraint>
   </security-constraint>

   <security-role>
     <role-name>eap_admin</role-name>
   </security-role>


   <login-config>
       <auth-method>BASIC</auth-method>
   </login-config>
   ```

   You need to specify the authentication method, which can be any of the following: **`BASIC, FORM, DIGEST, CLIENT-CERT, SPNEGO.`** This example uses **`BASIC`** authentication.

3. **Specify the security domain in the WAR's `jboss-web.xml`**

Add the security domain to the WAR's **`jboss-web.xml`** in order to connect the servlets to the configured security domain, which knows how to authenticate and authorize principals against the security constraints. The following example uses the security domain called **`acme_domain`**.

```
<jboss-web>
 ...
 <security-domain>acme_domain</security-domain>
 ...
</jboss-web>
```

Example 17.1. Example `web.xml` with Role-Based Security Configured

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0">

<display-name>Use Role-Based Security In Servlets</display-name>

<welcome-file-list>
  <welcome-file>/index.jsp</welcome-file>
</welcome-file-list>

<servlet-mapping>
    <servlet-name>DisplayOpResult</servlet-name>
    <url-pattern>/DisplayOpResult</url-pattern>
</servlet-mapping>

<security-constraint>
  <display-name>Restrict access to role eap_admin</display-name>
    <web-resource-collection>
      <web-resource-name>Restrict access to role eap_admin</web-
resource-name>
      <url-pattern>/DisplayOpResult/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
        <role-name>eap_admin</role-name>
      </auth-constraint>
    </security-constraint>

    <security-role>
      <role-name>eap_admin</role-name>
    </security-role>

    <login-config>
        <auth-method>BASIC</auth-method>
    </login-config>

</web-app>
```

Report a bug

17.5. USE A THIRD-PARTY AUTHENTICATION SYSTEM IN YOUR APPLICATION

You can integrate third-party security systems with JBoss EAP 6. These types of systems are usually token-based. The external system performs the authentication and passes a token back to the Web application through the request headers. This is often referred to as *perimeter authentication*. To configure perimeter authentication in your application, add a custom authentication valve. If you have a valve from a third-party provider, be sure it is in your classpath and follow the examples below, along with the documentation for your third-party authentication module.

NOTE

The location for configuring valves has changed in JBoss EAP 6. There is no longer a **`context.xml`** deployment descriptor. Valves are configured directly in the **`jboss-web.xml`** descriptor instead. The **`context.xml`** is now ignored.

Example 17.2. Basic Authentication Valve

```
<jboss-web>
  <valve>
    <class-
name>org.jboss.security.negotiation.NegotiationAuthenticator</class-
name>
  </valve>
</jboss-web>
```

This valve is used for Kerberos-based SSO. It also shows the most simple pattern for specifying a third-party authenticator for your Web application.

Example 17.3. Custom Valve With Header Attributes Set

```
<jboss-web>
  <valve>
    <class-
name>org.jboss.web.tomcat.security.GenericHeaderAuthenticator</class-
name>
    <param>
      <param-name>httpHeaderForSSOAuth</param-name>
      <param-value>sm_ssoid,ct-remote-user,HTTP_OBLIX_UID</param-value>
    </param>
    <param>
      <param-name>sessionCookieForSSOAuth</param-name>
      <param-value>SMSESSION,CTSESSION,ObSSOCookie</param-value>
    </param>
  </valve>
</jboss-web>
```

This example shows how to set custom attributes on your valve. The authenticator checks for the presence of the header ID and the session key, and passes them into the JAAS framework which drives the security layer, as the username and password value. You need a custom JAAS login module which can process the username and password and populate the subject with the correct roles. If no header values match the configured values, regular form-based authentication semantics apply.

Writing a Custom Authenticator

Writing your own authenticator is out of scope of this document. However, the following Java code is provided as an example.

Example 17.4. GenericHeaderAuthenticator.java

```
/*
 * JBoss, Home of Professional Open Source.
 * Copyright 2006, Red Hat Middleware LLC, and individual contributors
 * as indicated by the @author tags. See the copyright.txt file in the
 * distribution for a full listing of individual contributors.
 *
 * This is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as
 * published by the Free Software Foundation; either version 2.1 of
 * the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this software; if not, write to the Free
 * Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
 * 02110-1301 USA, or see the FSF site: http://www.fsf.org.
 */

package org.jboss.web.tomcat.security;

import java.io.IOException;
import java.security.Principal;
import java.util.StringTokenizer;

import javax.management.JMException;
import javax.management.ObjectName;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.catalina.Realm;
import org.apache.catalina.Session;
import org.apache.catalina.authenticator.Constants;
import org.apache.catalina.connector.Request;
import org.apache.catalina.connector.Response;
import org.apache.catalina.deploy.LoginConfig;
import org.jboss.logging.Logger;

import org.jboss.as.web.security.ExtendedFormAuthenticator;
```

```
/**
 * JBAS-2283: Provide custom header based authentication support
 *
 * Header Authenticator that deals with userid from the request header
Requires
 * two attributes configured on the Tomcat Service - one for the http
header
 * denoting the authenticated identity and the other is the SESSION
cookie
 *
 * @author <a href="mailto:Anil.Saldhana@jboss.org">Anil Saldhana</a>
 * @author <a href="mailto:sguilhen@redhat.com">Stefan Guilhen</a>
 * @version $Revision$
 * @since Sep 11, 2006
 */
public class GenericHeaderAuthenticator extends
ExtendedFormAuthenticator {
  protected static Logger log = Logger
      .getLogger(GenericHeaderAuthenticator.class);

  protected boolean trace = log.isTraceEnabled();

  // JBAS-4804: GenericHeaderAuthenticator injection of ssoid and
  // sessioncookie name.
  private String httpHeaderForSSOAuth = null;

  private String sessionCookieForSSOAuth = null;

  /**
   * <p>
   * Obtain the value of the <code>httpHeaderForSSOAuth</code>
attribute. This
   * attribute is used to indicate the request header ids that have to
be
   * checked in order to retrieve the SSO identity set by a third party
   * security system.
   * </p>
   *
   * @return a <code>String</code> containing the value of the
   *         <code>httpHeaderForSSOAuth</code> attribute.
   */
  public String getHttpHeaderForSSOAuth() {
    return httpHeaderForSSOAuth;
  }

  /**
   * <p>
   * Set the value of the <code>httpHeaderForSSOAuth</code> attribute.
This
   * attribute is used to indicate the request header ids that have to
be
   * checked in order to retrieve the SSO identity set by a third party
   * security system.
   * </p>
   *
   * @param httpHeaderForSSOAuth
   *             a <code>String</code> containing the value of the
   *             <code>httpHeaderForSSOAuth</code> attribute.
   */
  public void setHttpHeaderForSSOAuth(String httpHeaderForSSOAuth) {
    this.httpHeaderForSSOAuth = httpHeaderForSSOAuth;
  }

  /**
   * <p>
   * Obtain the value of the <code>sessionCookieForSSOAuth</code>
attribute.
   * This attribute is used to indicate the names of the SSO cookies
that may
   * be present in the request object.
   * </p>
   *
   * @return a <code>String</code> containing the names (separated by a
   *         <code>','</code>) of the SSO cookies that may have been
set by a
   *         third party security system in the request.
   */
  public String getSessionCookieForSSOAuth() {
    return sessionCookieForSSOAuth;
  }

  /**
   * <p>
   * Set the value of the <code>sessionCookieForSSOAuth</code>
attribute. This
   * attribute is used to indicate the names of the SSO cookies that may
be
   * present in the request object.
   * </p>
   *
   * @param sessionCookieForSSOAuth
   *             a <code>String</code> containing the names (separated
by a
   *             <code>','</code>) of the SSO cookies that may have been
set by
   *             a third party security system in the request.
   */
  public void setSessionCookieForSSOAuth(String sessionCookieForSSOAuth)
{
    this.sessionCookieForSSOAuth = sessionCookieForSSOAuth;
  }

  /**
   * <p>
   * Creates an instance of <code>GenericHeaderAuthenticator</code>.
   * </p>
   */
  public GenericHeaderAuthenticator() {
    super();
  }

  public boolean authenticate(Request request, HttpServletResponse
response,
      LoginConfig config) throws IOException {
    log.trace("Authenticating user");

    Principal principal = request.getUserPrincipal();
    if (principal != null) {
      if (trace)
        log.trace("Already authenticated '" + principal.getName() +
"'");
      return true;
    }

    Realm realm = context.getRealm();
    Session session = request.getSessionInternal(true);

    String username = getUserId(request);
    String password = getSessionCookie(request);

    // Check if there is sso id as well as sessionkey
    if (username == null || password == null) {
      log.trace("Username is null or password(sessionkey) is
null:fallback to form auth");
      return super.authenticate(request, response, config);
    }
    principal = realm.authenticate(username, password);

    if (principal == null) {
      forwardToErrorPage(request, response, config);
      return false;
    }

    session.setNote(Constants.SESS_USERNAME_NOTE, username);
    session.setNote(Constants.SESS_PASSWORD_NOTE, password);
    request.setUserPrincipal(principal);

    register(request, response, principal, HttpServletRequest.FORM_AUTH,
        username, password);
    return true;
  }

  /**
   * Get the username from the request header
   *
   * @param request
   * @return
   */
  protected String getUserId(Request request) {
    String ssoid = null;
    // We can have a comma-separated ids
    String ids = "";
    try {
      ids = this.getIdentityHeaderId();
    } catch (JMException e) {
      if (trace)
        log.trace("getUserId exception", e);
    }
```

```
        if (ids == null || ids.length() == 0)
          throw new IllegalStateException(
              "Http headers configuration in tomcat service missing");

        StringTokenizer st = new StringTokenizer(ids, ",");
        while (st.hasMoreTokens()) {
          ssoid = request.getHeader(st.nextToken());
          if (ssoid != null)
            break;
        }
        if (trace)
          log.trace("SSOID-" + ssoid);
        return ssoid;
      }

      /**
       * Obtain the session cookie from the request
       *
       * @param request
       * @return
       */
      protected String getSessionCookie(Request request) {
        Cookie[] cookies = request.getCookies();
        log.trace("Cookies:" + cookies);
        int numCookies = cookies != null ? cookies.length : 0;

        // We can have comma-separated ids
        String ids = "";
        try {
          ids = this.getSessionCookieId();
          log.trace("Session Cookie Ids=" + ids);
        } catch (JMException e) {
          if (trace)
            log.trace("checkSessionCookie exception", e);
        }
        if (ids == null || ids.length() == 0)
          throw new IllegalStateException(
              "Session cookies configuration in tomcat service missing");

        StringTokenizer st = new StringTokenizer(ids, ",");
        while (st.hasMoreTokens()) {
          String cookieToken = st.nextToken();
          String val = getCookieValue(cookies, numCookies, cookieToken);
          if (val != null)
            return val;
        }
        if (trace)
          log.trace("Session Cookie not found");
        return null;
      }

      /**
       * Get the configured header identity id in the tomcat service
       *
       * @return
       * @throws JMException
```

```
     */
    protected String getIdentityHeaderId() throws JMException {
      if (this.httpHeaderForSSOAuth != null)
        return this.httpHeaderForSSOAuth;
      return (String) mserver.getAttribute(new ObjectName(
          "jboss.web:service=WebServer"), "HttpHeaderForSSOAuth");
    }

    /**
     * Get the configured session cookie id in the tomcat service
     *
     * @return
     * @throws JMException
     */
    protected String getSessionCookieId() throws JMException {
      if (this.sessionCookieForSSOAuth != null)
        return this.sessionCookieForSSOAuth;
      return (String) mserver.getAttribute(new ObjectName(
          "jboss.web:service=WebServer"), "SessionCookieForSSOAuth");
    }

    /**
     * Get the value of a cookie if the name matches the token
     *
     * @param cookies
     *            array of cookies
     * @param numCookies
     *            number of cookies in the array
     * @param token
     *            Key
     * @return value of cookie
     */
    protected String getCookieValue(Cookie[] cookies, int numCookies,
        String token) {
      for (int i = 0; i < numCookies; i++) {
        Cookie cookie = cookies[i];
        log.trace("Matching cookieToken:" + token + " with cookie name="
            + cookie.getName());
        if (token.equals(cookie.getName())) {
          if (trace)
            log.trace("Cookie-" + token + " value=" + cookie.getValue());
          return cookie.getValue();
        }
      }
      return null;
    }
  }
```

Report a bug

CHAPTER 18. MIGRATION

18.1. CONFIGURE APPLICATION SECURITY CHANGES

Configure security for basic authentication

In previous versions of JBoss EAP, properties files placed in the ***EAP_HOME*/server/*SERVER_NAME*/conf/** directory were on classpath and could be easily found by the **UsersRolesLoginModule**. In JBoss EAP 6, the directory structure has changed. Properties files must be packaged within the application to make them available in the classpath.

IMPORTANT

You must stop the server before editing the server configuration file for your change to be persisted on server restart.

To configure security for basic authentication, add a new security domain under **security-domains** to the **standalone/configuration/standalone.xml** or the **domain/configuration/domain.xml** server configuration file:

```
<security-domain name="example">
    <authentication>
        <login-module code="UsersRoles" flag="required">
            <module-option name="usersProperties"
                    value="${jboss.server.config.dir}/example-
users.properties"/>
            <module-option name="rolesProperties"
                    value="${jboss.server.config.dir}/example-
roles.properties"/>
        </login-module>
    </authentication>
</security-domain>
```

If the JBoss EAP 6 instance is running as a standalone server, **${jboss.server.config.dir}** refers to the ***EAP_HOME*/standalone/configuration/** directory. If the instance is running in a managed domain, **${jboss.server.config.dir}** refers to the ***EAP_HOME*/domain/configuration/** directory.

Modify security domain names

In JBoss EAP 6, security domains no longer use the prefix **java:/jaas/** in their names.

- For Web applications, you must remove this prefix from the security domain configurations in the **jboss-web.xml**.
- For Enterprise applications, you must remove this prefix from the security domain configurations in the **jboss-ejb3.xml** file. This file has replaced the **jboss.xml** in JBoss EAP 6.

Report a bug

APPENDIX A. REFERENCE

A.1. INCLUDED AUTHENTICATION MODULES

The following authentication modules are included in JBoss EAP 6. Some of these handle authorization as well as authentication. These usually include the word **`Role`** within the **`Code`** name.

When you configure these modules, use the **`Code`** value or the full (package qualified) name to refer to the module.

Authentication Modules

Table A.1. `RealmDirect`

Code	**`RealmDirect`**
Class	**`org.jboss.as.security.RealmDirectLoginModule`**
Description	A login module implementation to interface directly with the security realm. This login module allows all interactions with the backing store to be delegated to the realm removing the need for any duplicate and synchronized definitions. Used for remoting calls and management interface.

Table A.2. `RealmDirect` Module Options

Option	Type	Default	Description
`realm`	string	**`ApplicationRealm`**	Name of the desired realm.

Table A.3. `Client`

Code	**`Client`**
Class	**`org.jboss.security.ClientLoginModule`**
Description	This login module is designed to establish caller identity and credentials when JBoss EAP 6 is acting as a client. It should never be used as part of a security domain used for server authentication.

Table A.4. `Client` Module Options

Option	Type	Default	Description
`multi-threaded`	**`true`** or **`false`**	**`false`**	Set to true if each thread has its own principal and credential storage. Set to false to indicate that all threads in the VM share the same identity and credential.
`password-stacking`	**`useFirstPass`** or **`false`**	**`false`**	Set to **`useFirstPass`** to indicate that this login module should look for information stored in the **`LoginContext`** to use as the identity. This option can be used when stacking other login modules with this one.
`restore-login-identity`	**`true`** or **`false`**	**`false`**	Set to true if the identity and credential seen at the start of the **`login()`** method should be restored after the **`logout()`** method is invoked.

Table A.5. `Remoting`

Code	**`Remoting`**
Class	**`org.jboss.as.security.remoting.RemotingLoginModule`**

Description	This login module is used to check if the request currently being authenticated is a request received over a Remoting connection, and if so the identity that was created during the Remoting authentication process is used and associated with the current request. If the request did not arrive over a Remoting connection this module does nothing and allows the JAAS based login to continue to the next module.

Table A.6. Remoting Module Options

Option	Type	Default	Description
`password-stacking`	**`useFirstPass`** or **`false`**	**`false`**	A value of **`useFirstPass`** indicates that this login module should first look to the information stored in the **`LoginContext`** for the identity. This option can be used when stacking other login modules with this one.
`principalClass`	A fully-qualified classname.	none	A **`Principal`** implementation class which contains a constructor that takes String arguments for the principal name.
`unauthenticatedIdentity`	A principal name.	none	Defines the principal name assigned to requests which contain no authentication information. This can allow unprotected servlets to invoke methods on EJBs that do not require a specific role. Such a principal has no associated roles and can only access unsecured EJBs or EJB methods that are associated with the **`unchecked permission`** constraint.

Table A.7. `Certificate`

Code	**`Certificate`**
Class	**`org.jboss.security.auth.spi.BaseCertLoginModule`**

Description	This login module is designed to authenticate users based on **`X509 Certificates`**. A use case for this is **`CLIENT-CERT`** authentication of a web application.

Table A.8. `Certificate` Module Options

Option	Type	Default	Description
`securityDomain`	string	none	Name of the security domain that has the JSSE configuration for the truststore holding the trusted certificates.
`verifier`	class	none	The class name of the **`org.jboss.security.auth.certs.X509CertificateVerifier`** to use for verification of the login certificate.

Table A.9. `CertificateRoles`

Code	**`CertificateRoles`**
Class	**`org.jboss.security.auth.spi.CertRolesLoginModule`**
Description	This login module extends the Certificate login module to add role mapping capabilities from a properties file. It takes all of the same options as the Certificate login module, and adds the following options.

Table A.10. `CertificateRoles` Module Options

Option	Type	Default	Description

Option	Type	Default	Description
`rolesProperties`	string	**`roles.properties`**	The name of the resource or file containing the roles to assign to each user. The role properties file must be in the format **`username=role1,role2`** where the username is the DN of the certificate, escaping any **`=`** (equals) and space characters. The following example is in the correct format: `CN\=unit-tests-client,\ OU\=Red\ Hat\ Inc.,\ O\=Red\ Hat\ Inc.,\ ST\=North\ Carolina,\ C\=US`
`defaultRolesProperties`	string	**`defaultRoles.properties`**	Name of the resource or file to fall back to if the **`rolesProperties`** file cannot be found.
`roleGroupSeparator`	A single character.	**`.`** (a single period)	Which character to use as the role group separator in the **`rolesProperties`** file.

Table A.11. `Database`

Code	**`Database`**
Class	**`org.jboss.security.auth.spi.DatabaseServerLoginModule`**
Description	A JDBC-based login module that supports authentication and role mapping. It is based on two logical tables, with the following definitions. • **`Principals: PrincipalID (text), Password (text)`** • **`Roles: PrincipalID (text), Role (text), RoleGroup (text)`**

Table A.12. `Database` Module Options

Option	Type	Default	Description
`digestCallback`	A fully-qualified classname	none	The class name of the **`DigestCallback`** implementation that includes pre/post digest content like salts for hashing the input password. Only used if **`hashAlgorithm`** has been specified.
`dsJndiName`	A JNDI resource	none	The name of the JNDI resource storing the authentication information. This option is required.
`hashAlgorithm`	String	Use plain passwords	The message digest algorithm used to hash passwords. Supported algorithms depend on the Java Security Provider, but the following are supported: **`MD5`**, **`SHA-1`**, and **`SHA-256`**.
`hashCharset`	String	The platform's default encoding	The name of the charset/encoding to use when converting the password String to a byte array. This includes all supported Java charset names.
`hashEncoding`	String	Base64	The string encoding format to use.
`ignorePasswordCase`	boolean	false	A flag indicating if the password comparison should ignore case.
`inputValidator`	A fully-qualified classname	none	The instance of the InputValidator implementation used to validate the username and password supplied by the client.
`principalsQuery`	prepared SQL statement	**`select Password from Principals where PrincipalID=?`**	The prepared SQL query to obtain the information about the principal.
`rolesQuery`	prepared SQL statement	none	The prepared SQL query to obtain the information about the roles. It should be equivalent to **`select Role, RoleGroup from Roles where PrincipalID=?`**, where Role is the role name and the RoleGroup column value should always be either **`Roles`** with a capital **`R`** or **`CallerPrincipal`**.

Option	Type	Default	Description
`storeDigestCallback`	A fully-qualified classname	none	The class name of the **`DigestCallback`** implementation that includes pre/post digest content like salts for hashing the store/expected password. Only used if **`hashStorePassword`** or **`hashUserPassword`** is **`true`** and **`hashAlgorithm`** has been specified.
`suspendResume`	boolean	true	Whether any existing JTA transaction should be suspended during database operations.
`throwValidatorError`	boolean	false	A flag that indicates whether validation errors should be exposed to clients or not
`transactionManagerJndiName`	JNDI Resource	java:/TransactionManager	The JNDI name of the transaction manager used by the login module.

Table A.13. `DatabaseCertificate`

Code	**`DatabaseCertificate`**
Class	**`org.jboss.security.auth.spi.DatabaseCertLoginModule`**
Description	This login module extends the Certificate login module to add role mapping capabilities from a database table. It has the same options plus these additional options:

Table A.14. `DatabaseCertificate` Module Options

Option	Type	Default	Description
`dsJndiName`	A JNDI resource		The name of the JNDI resource storing the authentication information. This option is required.

Option	Type	Default	Description
`rolesQuery`	prepared SQL statement	`select Role,RoleGroup from Roles where PrincipalID=?`	SQL prepared statement to be executed in order to map roles. It should be an equivalent to `select Role, RoleGroup from Roles where PrincipalID=?`, where Role is the role name and the RoleGroup column value should always be either `Roles` with a capital `R` or `CallerPrincipal`.
`suspendResume`	`true` or `false`	`true`	Whether any existing JTA transaction should be suspended during database operations.

Table A.15. `Identity`

Code	`Identity`
Class	`org.jboss.security.auth.spi.IdentityLoginModule`
Description	Associates the principal specified in the module options with any subject authenticated against the module. The type of Principal class used is `org.jboss.security.SimplePrincipal.`. If no principal option is specified a principal with the name of `guest` is used.

Table A.16. `Identity` Module Options

Option	Type	Default	Description
`principal`	String	`guest`	The name to use for the principal.
`roles`	comma-separated list of strings	none	A comma-delimited list of roles which will be assigned to the subject.

Table A.17. `Ldap`

Code	`Ldap`

Class	**`org.jboss.security.auth.spi.LdapLoginModule`**
Description	Authenticates against an LDAP server, when the username and password are stored in an LDAP server that is accessible using a JNDI LDAP provider. Many of the options are not required, because they are determined by the LDAP provider or the environment.

Table A.18. `Ldap` Module Options

Option	Type	Default	Description
`java.naming.factory.initial`	class name	**`com.sun.jndi.ldap.LdapCtxFactory`**	**`InitialContextFactory`** implementation class name.
`java.naming.provider.url`	**`ldap://`** URL	none	URL for the LDAP server.
`java.naming.security.authentication`	**`none`**, **`simple`**, or the name of a SASL mechanism	**`simple`**	The security level to use to bind to the LDAP server.
`java.naming.security.protocol`	transport protocol	If unspecified, determined by the provider.	The transport protocol to use for secure access, such as SSL.
`java.naming.security.principal`	string	none	The name of the principal for authenticating the caller to the service. This is built from other properties described below.
`java.naming.security.credentials`	credential type	none	The type of credential used by the authentication scheme. Some examples include hashed password, clear-text password, key, or certificate. If this property is unspecified, the behavior is determined by the service provider.

Option	Type	Default	Description
`principalDNPrefix`	string		Prefix added to the username to form the user DN. You can prompt the user for a username and build the fully-qualified DN by using the **`principalDNPrefix`** and **`principalDNSuffix`**.
`principalDNSuffix`	string		Suffix added to the username to form the user DN. You can prompt the user for a username and build the fully-qualified DN by using the **`principalDNPrefix`** and **`principalDNSuffix`**.
`useObjectCredential`	**`true`** or **`false`**	false	Whether the credential should be obtained as an opaque Object using the **`org.jboss.security.auth.callback.ObjectCallback`** type of Callback rather than as a **`char[]`** password using a JAAS PasswordCallback. This allows for passing **`non-char[]`** credential information to the LDAP server.
`rolesCtxDN`	fully-qualified DN	none	The fully-qualified DN for the context to search for user roles.
`userRolesCtxDNAttributeName`	attribute	none	The attribute in the user object that contains the DN for the context to search for user roles. This differs from **`rolesCtxDN`** in that the context to search for a user's roles may be unique for each user.
`roleAttributeID`	attribute	**`roles`**	Name of the attribute containing the user roles.

Option	Type	Default	Description
`roleAttributeIsDN`	**`true`** or **`false`**	**`false`**	Whether or not the **`roleAttributeID`** contains the fully-qualified DN of a role object. If false, the role name is taken from the value of the **`roleNameAttributeId`** attribute of the context name. Certain directory schemas, such as Microsoft Active Directory, require this attribute to be set to **`true`**.
`roleNameAttributeID`	attribute	**`name`**	Name of the attribute within the **`roleCtxDN`** context which contains the role name. If the **`roleAttributeIsDN`** property is set to **`true`**, this property is used to find the role object's name attribute.
`uidAttributeID`	attribute	**`uid`**	Name of the attribute in the **`UserRolesAttributeDN`** that corresponds to the user ID. This is used to locate the user roles.
`matchOnUserDN`	**`true`** or **`false`**	**`false`**	Whether or not the search for user roles should match on the user's fully-distinguished DN or the username only. If **`true`**, the full user DN is used as the match value. If **`false`**, only the username is used as the match value against the **`uidAttributeName`** attribute.
`allowEmptyPasswords`	**`true`** or **`false`**	**`false`**	Whether to allow empty passwords. Most LDAP servers treat empty passwords as anonymous login attempts. To reject empty passwords, set this to **`false`**.

Table A.19. `LdapExtended`

Code	**`LdapExtended`**
Class	**`org.jboss.security.auth.spi.LdapExtLoginModule`**
Description	An alternate LDAP login module implementation that uses searches to locate the bind user and associated roles. The roles query recursively follows DNs to navigate a hierarchical role structure. It uses the same **`java.naming`** options as the Ldap module, and uses the following options instead of the other options of the Ldap module. The authentication happens in 2 steps: 1. An initial bind to the LDAP server is done using the bindDN and **`bindCredential`** options. The **`bindDN`** is a LDAP user with the ability to search both the baseCtxDN and **`rolesCtxDN`** trees for the user and roles. The user DN to authenticate against is queried using the filter specified by the **`baseFilter`** attribute. 2. The resulting user DN is authenticated by binding to the LDAP server using the user DN as the **`InitialLdapContext`** environment **`Context.SECURITY_PRINCIPAL`**. The **`Context.SECURITY_CREDENTIALS`** property is set to the String password obtained by the callback handler.

Table A.20. `LdapExtended` Module Options

Option	Type	Default	Description
`baseCtxDN`	fully-qualified DN	none	The fixed DN of the top-level context to begin the user search.
`bindDN`	fully-qualified DN	none	The DN used to bind against the LDAP server for the user and roles queries. This DN needs read and search permissions on the **`baseCtxDN`** and **`rolesCtxDN`** values.
`bindCredential`	string, optionally encrypted	none	Password stored as plain text for the **`bindDN`**, or loaded externally using **EXT** command. This password can be

Option	Type	Default	Description
			encrypted using the [illegible]anism. You can use the following formats: • **`{EXT}...`** where the **`...`** is the required external command. Example: **`{EXT}cat /mysecretpasswordfile`** • **`{EXTC[:expiration_in_millis]}...`** where the **`...`** is the command that will be passed to the **`Runtime.exec(String)`** method to execute a platform command. The first line of the command output is used as the password. **`EXTC`** variant caches the passwords for **`expiration_in_millis`** milliseconds. Default cache expiration is 0 = infinity. • **`{CLASS}classname[:ctorargs]`** where the **`[:ctorargs]`** is an optional string delimited by the **`:`** from the classname that will be passed to the classname **`ctor`**. The **`ctorargs`** itself is a comma delimited list of strings.

Option	Type	Default	Description
			The password is obtained from classname by invoking a **`char[] toCharArray()`** method if found, otherwise, the **`String toString()`** method is used. See the following topic to read more about encrypting sensitive strings: Section 10.11.2, "Create a Java Keystore to Store Sensitive Strings"
`baseFilter`	LDAP filter string	none	A search filter used to locate the context of the user to authenticate. The input username or **`userDN`** obtained from the login module callback is substituted into the filter anywhere a **`{0}`** expression is used. A common example for the search filter is **`(uid={0})`**.
`rolesCtxDN`	fully-qualified DN	none	The fixed DN of the context to search for user roles. This is not the DN where the actual roles are, but the DN where the objects containing the user roles are. For example, in a Microsoft Active Directory server, this is the DN where the user account is.

Option	Type	Default	Description
`roleFilter`	LDAP filter string	none	A search filter used to locate the roles associated with the authenticated user. The input username or **`userDN`** obtained from the login module callback is substituted into the filter anywhere a **`{0}`** expression is used. The authenticated **`userDN`** is substituted into the filter anywhere a **`{1}`** is used. An example search filter that matches on the input username is **`(member={0})`**. An alternative that matches on the authenticated **`userDN`** is **`(member={1})`**.
`roleAttributeIsDN`	**`true`** or **`false`**	**`false`**	Whether or not the **`roleAttributeID`** contains the fully-qualified DN of a role object. If false, the role name is taken from the value of the **`roleNameAttributeId`** attribute of the context name. Certain directory schemas, such as Microsoft Active Directory, require this attribute to be set to **`true`**.
`defaultRole`	Role name	none	A role included for all authenticated users
`parseRoleNameFromDN`	**`true`** or **`false`**	**`false`**	A flag indicating if the DN returned by a query contains the roleNameAttributeID. If set to **`true`**, the DN is checked for the roleNameATtributeID. If set to **`false`**, the DN is not checked for the roleNameAttributeID. This flag can improve the performance of LDAP queries.

Option	Type	Default	Description
`parseUsername`	**`true`** or **`false`**	**`false`**	A flag indicating if the DN is to be parsed for the username. If set to **`true`**, the DN is parsed for the username. If set to **`false`** the DN is not parsed for the username. This option is used together with usernameBeginString and usernameEndString.
`usernameBeginString`	string	none	Defines the string which is to be removed from the start of the DN to reveal the username. This option is used together with **`usernameEndString`**.
`usernameEndString`	string	none	Defines the string which is to be removed from the end of the DN to reveal the username. This option is used together with **`usernameBeginString`**.
`roleNameAttributeID`	attribute	**`name`**	Name of the attribute within the **`roleCtxDN`** context which contains the role name. If the **`roleAttributeIsDN`** property is set to **`true`**, this property is used to find the role object's name attribute.
`distinguishedNameAttribute`	attribute	**`distinguishedName`**	The name of the attribute in the user entry that contains the DN of the user. This may be necessary if the DN of the user itself contains special characters (backslash for example) that prevent correct user mapping. If the attribute does not exist, the entry's DN is used.

Option	Type	Default	Description
`roleRecursion`	integer	**`0`**	The numbers of levels of recursion the role search will go below a matching context. Disable recursion by setting this to **`0`**.
`searchTimeLimit`	integer	**`10000`** (10 seconds)	The timeout in milliseconds for user or role searches.
`searchScope`	One of: **`OBJECT_SCOPE, ONELEVEL_SCOPE, SUBTREE_SCOPE`**	**`SUBTREE_SCOPE`**	The search scope to use.
`allowEmptyPasswords`	**`true`** or **`false`**	**`false`**	Whether to allow empty passwords. Most LDAP servers treat empty passwords as anonymous login attempts. To reject empty passwords, set this to false.

Table A.21. `RoleMapping`

Code	**`RoleMapping`**
Class	**`org.jboss.security.auth.spi.RoleMappingLoginModule`**
Description	Maps a role which is the end result of the authentication process to a declarative role. This module must be flagged as **`optional`** when you add it to the security domain.

Table A.22. `RoleMapping` Module Options

Option	Type	Default	Description
`rolesProperties`	The fully-qualified file path and name of a properties file or resource	**`none`**	The fully-qualified file path and name of a properties file or resource which maps roles to replacement roles. The format is **`original_role=role1,role2,role3`**

Option	Type	Default	Description
`replaceRole`	**`true`** or **`false`**	**`false`**	Whether to add to the current roles, or replace the current roles with the mapped ones. Replaces if set to **`true`**.

Table A.23. `RunAs`

Code	**`RunAs`**
Class	**`org.jboss.security.auth.spi.RunAsLoginModule`**
Description	A helper module that pushes a **`run as`** role onto the stack for the duration of the login phase of authentication, and pops the **`run as`** role off the stack in either the commit or abort phase. This login module provides a role for other login modules that must access secured resources in order to perform their authentication, such as a login module which accesses a secured EJB. **`RunAsLoginModule`** must be configured before the login modules that require a **`run as`** role to be established.

Table A.24. `RunAs` Options

Option	Type	Default	Description
`roleName`	role name	**`nobody`**	The name of the role to use as the **`run as`** role during the login phase.
`principalName`	principal name	**`nobody`**	Name of the principal to use as the **`run as`** principal during login phase. If not specified a default of **`nobody`** is used.
`principalClass`	A fully-qualified classname.	none	A **`Principal`** implementation class which contains a constructor that takes String arguments for the principal name.

Table A.25. `Simple`

Code	**`Simple`**

Class	**`org.jboss.security.auth.spi.SimpleServerLoginModule`**
Description	A module for quick setup of security for testing purposes. It implements the following simple algorithm: • If the password is null, authenticate the user and assign an identity of **`guest`** and a role of **`guest`**. • Otherwise, if the password is equal to the user, assign an identity equal to the username and both **`admin`** and **`guest`** roles. • Otherwise, authentication fails.

`Simple` Module Options

The **`Simple`** module has no options.

Table A.26. `ConfiguredIdentity`

Code	**`ConfiguredIdentity`**
Class	**`org.picketbox.datasource.security.ConfiguredIdentityLoginModule`**
Description	Associates the principal specified in the module options with any subject authenticated against the module. The type of Principal class used is **`org.jboss.security.SimplePrincipal`**.

Table A.27. `ConfiguredIdentity` Module Options

Option	Type	Default	Description
`username`	string	none	The username for authentication.

Option	Type	Default	Description
`password`	encrypted string	none	The password to use for authentication. To encrypt the password, use the module directly at the command line. `java org.picketbox.datasource.security.SecureIdentityLoginModule` *`password_to_encrypt`* Paste the result of this command into the module option's value field.
`principal`	Name of a principal	**`none`**	The principal which will be associated with any subject authenticated against the module.

Table A.28. `SecureIdentity`

Code	**`SecureIdentity`**
Class	**`org.picketbox.datasource.security.SecureIdentityLoginModule`**
Description	This module is provided for legacy purposes. It allows you to encrypt a password and then use the encrypted password with a static principal. If your application uses **`SecureIdentity`**, consider using a password vault mechanism instead.

Table A.29. `SecureIdentity` Module Options

Option	Type	Default	Description
`username`	string	none	The username for authentication.

Option	Type	Default	Description
`password`	encrypted string	none	The password to use for authentication. To encrypt the password, use the module directly at the command line. `java org.picketbox.datasource.security.SecureIdentityLoginModule *password_to_encrypt*` Paste the result of this command into the module option's value field.
`managedConnectionFactoryName`	JCA resource	none	The name of the JCA connection factory for your datasource.

Table A.30. `PropertiesUsers`

Code	**`PropertiesUsers`**
Class	**`org.jboss.security.auth.spi.PropertiesUsersLoginModule`**
Description	Uses a properties file to store usernames and passwords for authentication. No authorization (role mapping) is provided. This module is only appropriate for testing.

Table A.31. `SimpleUsers`

Code	**`SimpleUsers`**
Class	**`org.jboss.security.auth.spi.SimpleUsersLoginModule`**
Description	This login module stores the username and clear-text password using ***`module-option`***. ***`module-option`***'s ***`name`*** and ***`value`*** attributes specify a username and password. It is included for testing only, and is not appropriate for a production environment.

Table A.32. `LdapUsers`

Code	**`LdapUsers`**
Class	**`org.jboss.security.auth.spi.LdapUsersLoginModule`**
Description	The **`LdapUsers`** module is superseded by the **`ExtendedLDAP`** and **`AdvancedLdap`** modules.

Table A.33. `Kerberos`

Code	**`Kerberos`**
Class	**`com.sun.security.auth.module.Krb5LoginModule`**
Description	Performs Kerberos login authentication, using GSSAPI. This module is part of the security framework from the API provided by Sun Microsystems. Details can be found at http://docs.oracle.com/javase/7/docs/jre/api/security/jaas/spec/com/sun/security/auth/module/Krb5LoginModule.html. This module needs to be paired with another module which handles the authentication and roles mapping.

Table A.34. `Kerberos` Module Options

Option	Type	Default	Description
`storekey`	**`true`** or **`false`**	false	Whether or not to add the **`KerberosKey`** to the subject's private credentials.
`doNotPrompt`	**`true`** or **`false`**	false	If set to **`true`**, the user is not prompted for the password.
`useTicketCache`	Boolean value of **`true`** or **`false`** .	false	If **`true`**, the TGT is obtained from the ticket cache. If **`false`**, the ticket cache is not used.

Option	Type	Default	Description
`ticketcache`	A file or resource representing a Kerberos ticket cache.	The default depends on which operating system you use. • Red Hat Enterprise Linux / Solaris: **`/tmp/krb5cc_`*`uid`***, using the numeric UID value of the operating system. • Microsoft Windows Server: uses the Local Security Authority (LSA) API to find the ticketcache.	The location of the ticket cache.
`useKeyTab`	**`true`** or **`false`**	false	Whether to obtain the principal's key from a key table file.
`keytab`	A file or resource representing a Kerberos keytab.	the location in the operating system's Kerberos configuration file, or **`/home/`*`user`*`/krb5.keytab`**	The location of the key table file.
`principal`	string	none	The name of the principal. This can either be a simple user name or a service name such as **`host/testserver.acme.com`**. Use this instead of obtaining the principal from the key table, or when the key table contains more than one principal.

Option	Type	Default	Description
`useFirstPass`	**`true`** or **`false`**	false	Whether to retrieve the username and password from the module's shared state, using **`javax.security.auth.login.name`** and **`javax.security.auth.login.password`** as the keys. If authentication fails, no retry attempt is made.
`tryFirstPass`	**`true`** or **`false`**	false	Same as **`useFirstPass`**, but if authentication fails, the module uses the **`CallbackHandler`** to retrieve a new username and password. If the second authentication fails, the failure is reported to the calling application.
`storePass`	**`true`** or **`false`**	false	Whether to store the username and password in the module's shared state. This does not happen if the keys already exist in the shared state, or if authentication fails.
`clearPass`	**`true`** or **`false`**	false	Set this to **`true`** to clear the username and password from the shared state after both phases of authentication complete.

Table A.35. `SPNEGO`

Code	**`SPNEGO`**
Class	**`org.jboss.security.negotiation.spnego.SPNEGOLoginModule`**
Description	Allows SPNEGO authentication to a Microsoft Active Directory server or other environment which supports SPNEGO. SPNEGO can also carry Kerberos credentials. This module needs to be paired with another module which handles authentication and role mapping.

Table A.36. `SPNEGO` Module Options

Option	Type	Default	Description
`serverSecurityDomain`	`string`	`null`.	Defines the domain that is used to retrieve the identity of the server service through the kerberos login module. This property must be set.
`removeRealmFromPrincipal`	`boolean`	`false`	Specifies that the Kerberos realm should be removed from the principal before further processing.
`usernamePasswordDomain`	`string`	`null`	Specifies another security domain within the configuration that should be used as a failover login when Kerberos fails.

Table A.37. `AdvancedLdap`

Code	`AdvancedLdap`
Class	`org.jboss.security.negotiation.AdvancedLdapLoginModule`
Description	A module which provides additional functionality, such as SASL and the use of a JAAS security domain.

Table A.38. `AdvancedLdap` Module Options

Option	Type	Default	Description
`bindAuthentication`	string	none	The type of SASL authentication to use for binding to the directory server.
`java.naming.provider.url`	`string`	none	The URI of the directory server.
`baseCtxDN`	fully-qualified DN	none	The distinguished name to use as the base for searches.

Option	Type	Default	Description
`baseFilter`	String representing a LDAP search filter.	none	The filter to use to narrow down search results.
`roleAttributeID`	String value representing an LDAP attribute.	none	The LDAP attribute which contains the names of authorization roles.
`roleAttributeIsDN`	**`true`** or **`false`**	**`false`**	Whether the role attribute is a Distinguished Name (DN).
`roleNameAttributeID`	String representing an LDAP attribute.	none	The attribute contained within the **`RoleAttributeId`** which contains the actual role attribute.
`recurseRoles`	**`true`** or **`false`**	**`false`**	Whether to recursively search the **`RoleAttributeId`** for roles.

Table A.39. `AdvancedADLdap`

Code	**`AdvancedADLdap`**
Class	**`org.jboss.security.negotiation.AdvancedADLoginModule`**
Description	This module extends the **`AdvancedLdap`** login module, and adds extra parameters that are relevant to Microsoft Active Directory.

Table A.40. `UsersRoles`

Code	**`UsersRoles`**
Class	**`org.jboss.security.auth.spi.UsersRolesLoginModul`**
Description	A simple login module that supports multiple users and user roles stored in two different properties files.

Table A.41. `UsersRoles` Module Options

Option	Type	Default	Description
usersProperties	Path to a file or resource.	**users.properties**	The file or resource which contains the user-to-password mappings. The format of the file is ***user=hashed-password***
rolesProperties	Path to a file or resource.	**roles.properties**	The file or resource which contains the user-to-role mappings. The format of the file is **username=role1,role2,role3**
password-stacking	**useFirstPass** or **false**	**false**	A value of **useFirstPass** indicates that this login module should first look to the information stored in the **LoginContext** for the identity. This option can be used when stacking other login modules with this one.
hashAlgorithm	String representing a password hashing algorithm.	**none**	The name of the **java.security.MessageDigest** algorithm to use to hash the password. There is no default so this option must be explicitly set to enable hashing. When **hashAlgorithm** is specified, the clear text password obtained from the **CallbackHandler** is hashed before it is passed to **UsernamePasswordLoginModule.validatePassword** as the **inputPassword** argument. The password stored in the **users.properties** file must be comparably hashed.
hashEncoding	**base64** or **hex**	**base64**	The string format for the hashed password, if hashAlgorithm is also set.

Option	Type	Default	Description
`hashCharset`	string	The default encoding set in the container's runtime environment	The encoding used to convert the clear-text password to a byte array.
`unauthenticatedIdentity`	principal name	none	Defines the principal name assigned to requests which contain no authentication information. This can allow unprotected servlets to invoke methods on EJBs that do not require a specific role. Such a principal has no associated roles and can only access unsecured EJBs or EJB methods that are associated with the **`unchecked permission`** constraint.

Custom Authentication Modules

Authentication modules are implementations of **`javax.security.auth.spi.LoginModule`**. Refer to the API documentation for more information about creating a custom authentication module.

Report a bug

A.2. INCLUDED AUTHORIZATION MODULES

The following modules provide authorization services.

Code	Class
DenyAll	org.jboss.security.authorization.modules.AllDenyAuthorizationModule
PermitAll	org.jboss.security.authorization.modules.AllPermitAuthorizationModule
Delegating	org.jboss.security.authorization.modules.DelegatingAuthorizationModule
Web	org.jboss.security.authorization.modules.WebAuthorizationModule
JACC	org.jboss.security.authorization.modules.JACCAuthorizationModule

Report a bug

A.3. INCLUDED SECURITY MAPPING MODULES

The following security mapping roles are provided in JBoss EAP 6.

Code	Class
PropertiesRoles	org.jboss.security.mapping.providers.role.PropertiesRolesMappingProvider
SimpleRoles	org.jboss.security.mapping.providers.role.SimpleRolesMappingProvider
DeploymentRoles	org.jboss.security.mapping.providers.DeploymentRolesMappingProvider
DatabaseRoles	org.jboss.security.mapping.providers.role.DatabaseRolesMappingProvider
LdapRoles	org.jboss.security.mapping.providers.role.LdapRolesMappingProvider

Report a bug

A.4. INCLUDED SECURITY AUDITING PROVIDER MODULES

JBoss EAP 6 provides one security auditing provider.

Code	Class
LogAuditProvider	org.jboss.security.audit.providers.LogAuditProvider

Report a bug

A.5. JBOSS-WEB.XML CONFIGURATION REFERENCE

Introduction

The **`jboss-web.xml`** is a file within your deployment's **`WEB-INF`** or **`META-INF`** directory. It contains configuration information about features the JBoss Web container adds to the Servlet 3.0 specification. Settings specific to the Servlet 3.0 specification are placed into **`web.xml`** in the same directory.

The top-level element in the **`jboss-web.xml`** file is the **`<jboss-web>`** element.

Mapping Global Resources to WAR Requirements

Many of the available settings map requirements set in the application's **`web.xml`** to local resources. The explanations of the **`web.xml`** settings can be found at http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html.

For instance, if the **`web.xml`** requires **`jdbc/MyDataSource`**, the **`jboss-web.xml`** may map the global datasource **`java:/DefaultDS`** to fulfill this need. The WAR uses the global datasource to fill its need for **`jdbc/MyDataSource`**.

Table A.42. Common Top-Level Attributes

Attribute	Description
env-entry	A mapping to an **`env-entry`** required by the **`web.xml`**.
ejb-ref	A mapping to an **`ejb-ref`** required by the **`web.xml`**.
ejb-local-ref	A mapping to an **`ejb-local-ref`** required by the **`web.xml`**.
service-ref	A mapping to a **`service-ref`** required by the **`web.xml`**.
resource-ref	A mapping to a **`resource-ref`** required by the **`web.xml`**.
resource-env-ref	A mapping to a **`resource-env-ref`** required by the **`web.xml`**.
message-destination-ref	A mapping to a **`message-destination-ref`** required by the **`web.xml`**.
persistence-context-ref	A mapping to a **`persistence-context-ref`** required by the **`web.xml`**.
persistence-unit-ref	A mapping to a **`persistence-unit-ref`** required by the **`web.xml`**.
post-construct	A mapping to a **`post-context`** required by the **`web.xml`**.
pre-destroy	A mapping to a **`pre-destroy`** required by the **`web.xml`**.
data-source	A mapping to a **`data-source`** required by the **`web.xml`**.
context-root	The root context of the application. The default value is the name of the deployment without the **`.war`** suffix.
virtual-host	The name of the HTTP virtual-host the application accepts requests from. It refers to the contents of the HTTP **`Host`** header.

Attribute	Description
annotation	Describes an annotation used by the application. Refer to <annotation> for more information.
listener	Describes a listener used by the application. Refer to <listener> for more information.
session-config	This element fills the same function as the **`<session-config>`** element of the **`web.xml`** and is included for compatibility only.
valve	Describes a valve used by the application. Refer to <valve> for more information.
overlay	The name of an overlay to add to the application.
security-domain	The name of the security domain used by the application. The security domain itself is configured in the web-based management console or the management CLI.
security-role	This element fills the same function as the **`<security-role>`** element of the **`web.xml`** and is included for compatibility only.
use-jboss-authorization	If this element is present and contains the case insensitive value "true", the JBoss web authorization stack is used. If it is not present or contains any value that is not "true", then only the authorization mechanisms specified in the Java Enterprise Edition specifications are used. This element is new to JBoss EAP 6.
disable-audit	If this empty element is present, web security auditing is disabled. Otherwise, it is enabled. Web security auditing is not part of the Java EE specification. This element is new to JBoss EAP 6.
disable-cross-context	If **`false`**, the application is able to call another application context. Defaults to **`true`**.

The following elements each have child elements.

<annotation>

Describes an annotation used by the application. The following table lists the child elements of an **`<annotation>`**.

Table A.43. Annotation Configuration Elements

Attribute	Description
class-name	Name of the class of the annotation
servlet-security	The element, such as **`@ServletSecurity`**, which represents servlet security.
run-as	The element, such as **`@RunAs`**, which represents the run-as information.
multi-part	The element, such as **`@MultiPart`**, which represents the multi-part information.

<listener>

Describes a listener. The following table lists the child eleents of a **`<listener>`**.

Table A.44. Listener Configuration Elements

Attribute	Description
class-name	Name of the class of the listener
listener-type	List of **`condition`** elements, which indicate what kind of listener to add to the Context of the application. Valid choices are: **CONTAINER** Adds a ContainerListener to the Context. **LIFECYCLE** Adds a LifecycleListener to the Context. **SERVLET_INSTANCE** Adds an InstanceListener to the Context. **SERVLET_CONTAINER** Adds a WrapperListener to the Context. **SERVLET_LIFECYCLE** Adds a WrapperLifecycle to the Context.
module	The name of the module containing the listener class.
param	A parameter. Contains two child elements, **`<param-name>`** and **`<param-value>`**.

<valve>

Describes a valve of the application. It contains the same configuration elements as <listener>.

Report a bug

A.6. EJB SECURITY PARAMETER REFERENCE

Table A.45. EJB security parameter elements

Element	Description
`<security-identity>`	Contains child elements pertaining to the security identity of an EJB.
`<use-caller-identity />`	Indicates that the EJB uses the same security identity as the caller.
`<run-as>`	Contains a **`<role-name>`** element.
`<run-as-principal>`	If present, indicates the principal assigned to outgoing calls. If not present, outgoing calls are assigned to a principal named **`anonymous`**.
`<role-name>`	Specifies the role the EJB should run as.
`<description>`	Describes the role named in **`<role-name>`** .

Example A.1. Security identity examples

This example shows each tag described in Table A.45, "EJB security parameter elements". They can also be used inside a **`<session>`**.

```
<ejb-jar>
    <enterprise-beans>
        <session>
            <ejb-name>ASessionBean</ejb-name>
            <security-identity>
                <use-caller-identity/>
            </security-identity>
        </session>
        <session>
            <ejb-name>RunAsBean</ejb-name>
            <security-identity>
                <run-as>
                    <description>A private internal role</description>
                    <role-name>InternalRole</role-name>
                </run-as>
            </security-identity>
        </session>
    <session>
    <ejb-name>RunAsBean</ejb-name>
    <security-identity>
    <run-as-principal>internal</run-as-principal>
    </security-identity>
```

```
        </session>
    </enterprise-beans>
</ejb-jar>
```

Report a bug

APPENDIX B. REVISION HISTORY

Revision 6.2.2-15	**Wed May 21 2014**	**Tom Wells, Russell Dickenson, Lucas Costi, Sande Gilda**

Red Hat JBoss Enterprise Application Platform Common Criteria Certification