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)  
1.2. ABOUT SECURITY  
1.3. SECURITY FOR THE SYSTEM ADMINISTRATOR  
1.4. SECURITY FOR THE J2EE DEVELOPER  

**PART II. SECURING THE PLATFORM**  

**CHAPTER 2. THE SECURITY SUBSYSTEM**  

2.1. ABOUT THE SECURITY SUBSYSTEM  
2.2. ABOUT THE STRUCTURE OF THE SECURITY SUBSYSTEM  
2.3. ABOUT ENCRYPTION  
2.4. ABOUT DECLARATIVE SECURITY  
2.5. SECURITY REFERENCES  
2.6. CONFIGURE THE SECURITY SUBSYSTEM  
2.7. JAVA EE DECLARATIVE SECURITY OVERVIEW  
2.8. SECURITY IDENTITY  
2.9. SECURITY ROLES  
2.10. EJB METHOD PERMISSIONS  
2.11. ENTERPRISE BEANS SECURITY ANNOTATIONS  
2.12. WEB CONTENT SECURITY CONSTRAINTS  
2.13. ENABLE FORM-BASED AUTHENTICATION  
2.14. ENABLE DECLARATIVE SECURITY  

**CHAPTER 3. MANAGEMENT INTERFACE SECURITY**  

3.1. SECURE THE MANAGEMENT INTERFACES  
3.2. DEFAULT USER SECURITY CONFIGURATION  
3.3. OVERVIEW OF ADVANCED MANAGEMENT INTERFACE CONFIGURATION  
3.4. DISABLE THE HTTP MANAGEMENT INTERFACE  
3.5. REMOVE SILENT AUTHENTICATION FROM THE DEFAULT SECURITY REALM  
3.6. DISABLE REMOTE ACCESS TO THE JMX SUBSYSTEM  
3.7. CONFIGURE SECURITY REALMS FOR THE MANAGEMENT INTERFACES  
3.8. PASSWORD VAULTS FOR SENSITIVE STRINGS  
3.8.1. About Securing Sensitive Strings in Clear-Text Files  
3.8.2. Create a Java Keystore to Store Sensitive Strings  
3.8.3. Mask the Keystore Password and Initialize the Password Vault  
3.8.4. Configure JBoss EAP 6 to Use the Password Vault  
3.8.5. Store and Retrieve Encrypted Sensitive Strings in the Java Keystore  
3.8.6. Store and Resolve Sensitive Strings In Your Applications  
3.9. LDAP  
3.9.1. About LDAP  
3.9.2. Use LDAP to Authenticate to the Management Interfaces  

**CHAPTER 4. JAVA SECURITY MANAGER**  

4.1. ABOUT THE JAVA SECURITY MANAGER  
4.2. ABOUT JAVA SECURITY MANAGER POLICIES  
4.3. RUN JBOSS EAP 6 WITHIN THE JAVA SECURITY MANAGER  
4.4. WRITE A JAVA SECURITY MANAGER POLICY  
4.5. DEBUG SECURITY MANAGER POLICIES  

**CHAPTER 5. PATCH INSTALLATION**  

4.6. PATCH INSTALLATION  
4.7. APPLYING A JAVA SECURITY MANAGER POLICY  
4.8. DEBUG SECURITY MANAGER POLICIES  
4.9. ADMINISTERING AND MANAGING PATCHES AND POLICIES  
4.10. MIGRATE APPLICATIONS TO NEW POLICIES  

---

**Table of Contents**  

6: PART I. SECURITY FOR RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6  
7: CHAPTER 1. INTRODUCTION  
7: 1.1. ABOUT RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6 (JBoss EAP 6)  
7: 1.2. ABOUT SECURITY  
7: 1.3. SECURITY FOR THE SYSTEM ADMINISTRATOR  
8: 1.4. SECURITY FOR THE J2EE DEVELOPER  
9: PART II. SECURING THE PLATFORM  
10: CHAPTER 2. THE SECURITY SUBSYSTEM  
10: 2.1. ABOUT THE SECURITY SUBSYSTEM  
10: 2.2. ABOUT THE STRUCTURE OF THE SECURITY SUBSYSTEM  
11: 2.3. ABOUT ENCRYPTION  
11: 2.4. ABOUT DECLARATIVE SECURITY  
12: 2.5. SECURITY REFERENCES  
13: 2.6. CONFIGURE THE SECURITY SUBSYSTEM  
14: 2.7. JAVA EE DECLARATIVE SECURITY OVERVIEW  
14: 2.8. SECURITY IDENTITY  
16: 2.9. SECURITY ROLES  
17: 2.10. EJB METHOD PERMISSIONS  
20: 2.11. ENTERPRISE BEANS SECURITY ANNOTATIONS  
21: 2.12. WEB CONTENT SECURITY CONSTRAINTS  
23: 2.13. ENABLE FORM-BASED AUTHENTICATION  
24: 2.14. ENABLE DECLARATIVE SECURITY  
25: CHAPTER 3. MANAGEMENT INTERFACE SECURITY  
25: 3.1. SECURE THE MANAGEMENT INTERFACES  
25: 3.2. DEFAULT USER SECURITY CONFIGURATION  
26: 3.3. OVERVIEW OF ADVANCED MANAGEMENT INTERFACE CONFIGURATION  
27: 3.4. DISABLE THE HTTP MANAGEMENT INTERFACE  
28: 3.5. REMOVE SILENT AUTHENTICATION FROM THE DEFAULT SECURITY REALM  
30: 3.6. DISABLE REMOTE ACCESS TO THE JMX SUBSYSTEM  
30: 3.7. CONFIGURE SECURITY REALMS FOR THE MANAGEMENT INTERFACES  
31: 3.8. PASSWORD VAULTS FOR SENSITIVE STRINGS  
31: 3.8.1. About Securing Sensitive Strings in Clear-Text Files  
31: 3.8.2. Create a Java Keystore to Store Sensitive Strings  
34: 3.8.3. Mask the Keystore Password and Initialize the Password Vault  
35: 3.8.4. Configure JBoss EAP 6 to Use the Password Vault  
36: 3.8.5. Store and Retrieve Encrypted Sensitive Strings in the Java Keystore  
39: 3.8.6. Store and Resolve Sensitive Strings In Your Applications  
41: 3.9. LDAP  
41: 3.9.1. About LDAP  
42: 3.9.2. Use LDAP to Authenticate to the Management Interfaces  
46: CHAPTER 4. JAVA SECURITY MANAGER  
46: 4.1. ABOUT THE JAVA SECURITY MANAGER  
46: 4.2. ABOUT JAVA SECURITY MANAGER POLICIES  
46: 4.3. RUN JBOSS EAP 6 WITHIN THE JAVA SECURITY MANAGER  
47: 4.4. WRITE A JAVA SECURITY MANAGER POLICY  
50: 4.5. DEBUG SECURITY MANAGER POLICIES  
52: CHAPTER 5. PATCH INSTALLATION  
52: 4.6. PATCH INSTALLATION  
52: 4.7. APPLYING A JAVA SECURITY MANAGER POLICY  
52: 4.8. DEBUG SECURITY MANAGER POLICIES  
52: 4.9. ADMINISTERING AND MANAGING PATCHES AND POLICIES  
52: 4.10. MIGRATE APPLICATIONS TO NEW POLICIES
12.3.2.1. About EJB Method Permissions
12.3.2.2. Use EJB Method Permissions
12.3.3. EJB Security Annotations
12.3.3.1. About EJB Security Annotations
12.3.3.2. Use EJB Security Annotations
12.3.4. Remote Access to EJBs
12.3.4.1. About Remote Method Access
12.3.4.2. About Remoting Callbacks
12.3.4.3. About Remoting Server Detection
12.3.4.4. Configure the Remoting Subsystem
12.3.4.5. Use Security Realms with Remote EJB Clients
12.3.4.6. Add a New Security Realm
12.3.4.7. Add a User to a Security Realm
12.3.4.8. About Remote EJB Access Using SSL Encryption
12.4. JAX-RS APPLICATION SECURITY
12.4.1. Enable Role-Based Security for a RESTEasy JAX-RS Web Service
12.4.2. Secure a JAX-RS Web Service using Annotations
12.5. SECURE REMOTE PASSWORD PROTOCOL
12.5.1. About Secure Remote Password Protocol (SRP)
12.5.2. Configure Secure Remote Password (SRP) Protocol

CHAPTER 13. SINGLE SIGN ON (SSO) ................................................. 118.
13.1. ABOUT SINGLE SIGN ON (SSO) FOR WEB APPLICATIONS
13.2. ABOUT CLUSTERED SINGLE SIGN ON (SSO) FOR WEB APPLICATIONS
13.3. CHOOSE THE RIGHT SSO IMPLEMENTATION
13.4. USE SINGLE SIGN ON (SSO) IN A WEB APPLICATION
13.5. ABOUT KERBEROS
13.6. ABOUT SPNEGO
13.7. ABOUT MICROSOFT ACTIVE DIRECTORY
13.8. CONFIGURE KERBEROS OR MICROSOFT ACTIVE DIRECTORY DESKTOP SSO FOR WEB APPLICATIONS

CHAPTER 14. ROLE-BASED SECURITY IN APPLICATIONS .......................... 128.
14.1. ABOUT APPLICATION SECURITY
14.2. ABOUT SECURITY AUDITING
14.3. ABOUT SECURITY MAPPING
14.4. ABOUT THE SECURITY EXTENSION ARCHITECTURE
14.5. ABOUT JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)
14.6. USE A SECURITY DOMAIN IN YOUR APPLICATION
14.7. USE ROLE-BASED SECURITY IN SERVLETS
14.8. USE A THIRD-PARTY AUTHENTICATION SYSTEM IN YOUR APPLICATION

CHAPTER 15. MIGRATION ........................................................................ 146.
15.1. CONFIGURE APPLICATION SECURITY CHANGES

CHAPTER 16. AUTHENTICATION AND AUTHORIZATION .......................... 147.
16.1. ABOUT AUTHENTICATION
16.2. ABOUT AUTHORIZATION
16.3. JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)
16.4. ABOUT JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)
16.5. JAVA AUTHORIZATION CONTRACT FOR CONTAINERS (JACC)
16.5.1. About Java Authorization Contract for Containers (JACC)
16.5.2. Configure Java Authorization Contract for Containers (JACC) Security
16.6. JAVA AUTHENTICATION SPI FOR CONTAINERS (JASPI)
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 SECURITY

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 following document: https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html-single/Security_Guide/index.html

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

1.3. SECURITY FOR THE SYSTEM ADMINISTRATOR

System Administrators, tasked with providing always on computer systems and networks, must be proficient in dealing with attacks on their networks and must also be proactive in thwarting such attacks by the use of planned security drills and audits.

For a successful system administrator, planning for security breaches is a combination of art and science. Security threats, whether they be physical, network or data based, are varying in nature and a successful security system administrator can prepare for outages.
1.4. SECURITY FOR THE J2EE DEVELOPER

Application level security falls in the hands of the J2EE Developer. Even this can be divided into three separate roles:

- Application Developer - responsible for security at the development level and for defining the roles, rules and business logic into the application logic.

- Application Assembler - responsible for ensuring that the packaging of EAR's and WAR's is done so that cross-application vulnerabilities are minimized.

- Application Deployer - responsible for securing the deployment of EAR's and assigning and maintaining access control lists.

It is not uncommon for all three roles to be played by the same set of developers.

JBoss EAP 6, as a component platform, provides declarative security. Rather than embed security logic into a business component, you describe the security roles and permissions in a standard XML descriptor. This way, business level code is isolated from the security code. Read more about declarative security in JBoss EAP 6 here Section 2.4, “About Declarative Security”.

Declarative security is bolstered by programmatic security. J2EE developers can use J2EE APIs in code to determine authorization and enforce enhanced security.
PART II. SECURING THE PLATFORM
CHAPTER 2. THE SECURITY SUBSYSTEM

2.1. 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

2.2. 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 2.1. Example Security Subsystem Configuration

```xml
<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-users.properties"/>
        </login-module>
      </authentication>
    </security-domain>
  </security-domains>
</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

2.3. 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

2.4. 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
2.5. 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 2.2. ejb-jar.xml descriptor fragment

```xml
<!-- 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.
2.6. 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 2.2, “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.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deep-copy-subject-mode</td>
<td>Specifies whether to copy or link to security tokens, for additional thread safety.</td>
</tr>
<tr>
<td>authentication-manager-class-name</td>
<td>Specifies an alternate AuthenticationManager implementation class name to use.</td>
</tr>
<tr>
<td>authorization-manager-class-name</td>
<td>Specifies an alternate AuthorizationManager implementation class name to use.</td>
</tr>
<tr>
<td>audit-manager-class-name</td>
<td>Specifies an alternate AuditManager implementation class name to use.</td>
</tr>
<tr>
<td>identity-trust-manager-class-name</td>
<td>Specifies an alternate IdentityTrustManager implementation class name to use.</td>
</tr>
<tr>
<td>mapping-manager-class-name</td>
<td>Specifies the MappingManager implementation class name to use.</td>
</tr>
</tbody>
</table>

<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.

2.7. 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.

2.8. 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 indicates 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:

```xml
<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.

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

### 2.9. 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 2.4. An `ejb-jar.xml` descriptor fragment that illustrates the security-role element usage.

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

---

- **Report a bug**
Example 2.5. An example web.xml descriptor fragment that illustrates the security-role element usage.

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

2.10. 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:

```xml
<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:

```xml
<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:

```xml
<method>
  <ejb-name>EJBNAME</ejb-name>
  <method-name>METHOD</method-name>
  <method-param>
    <java-type>TYPE</java-type>
  </method-param>
</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 2.6, "An ejb-jar.xml descriptor fragment that illustrates the method-permission element usage." provides complete examples of the method-permission element usage.

Example 2.6. An ejb-jar.xml descriptor fragment that illustrates the method-permission element usage.

```xml
<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>
      <method-params>
        <method-param>java.lang.String</method-param>
      </method-params>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```
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

### 2.12. 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](image)

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 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 2.7. web.xml Descriptor Fragment**

---
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.

2.14. 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. MANAGEMENT INTERFACE SECURITY

3.1. 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 3.5, “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 11.2, “Specify Which Network Interface JBoss EAP 6 Uses”
- Section 11.3, “Configure Network Firewalls to Work with JBoss EAP 6”

Report a bug

3.2. 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:

  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.

  NOTE

  HTTP access is considered to be remote, even if you connect to the localhost using HTTP.
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, refer to the Getting Started chapter of the Installation guide for JBoss EAP 6. For each user, the username, a hashed password, and the realm are stored in a file.

**Standalone server**

```text
JPP_HOME/standalone/configuration/mgmt-users.properties
```

Even though the contents of the `mgmt-users.properties` are masked, the file should 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

### 3.3. 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>`

**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.

**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.

**Report a bug**

### 3.4. 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 3.1. 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,
```
To remove the HTTP interface, issue the following command:

**Example 3.2. 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 3.3. Re-Create the HTTP Interface**

```
```

3.5. 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 3.4. Example of the `local` element in the `security-realm`**

```
<security-realms>
  <security-realm name="ManagementRealm">
    <authentication>
      <local default-user="$local"/>
    </authentication>
  </security-realm>
</security-realms>
```
Procedure 3.1. 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
3.6. 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 3.5. Remove the Remote Connector from the JMX Subsystem**

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

**Example 3.6. 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

3.7. 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 3.7. 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" => {
```
Write a Security Realm

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

Example 3.8. 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 its name as a reference to the Management Interface.

Example 3.9. Add a Security Realm to a Management Interface

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

Report a bug

3.8. PASSWORD VAULTS FOR SENSITIVE STRINGS

3.8.1. 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.

Report a bug

3.8.2. 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 3.2. 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 default is DSA. The example in this procedure uses RSA. Check 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 default size of keys is 1024. It must be between 512 and 2048, and a multiple of 64. The example in this procedure uses 2048.

   **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 3.10. Example input and output of keystore command

```
$ keytool -genkey -alias vault -keyalg RSA -keysize 2048 -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

3.8.3. Mask the Keystore Password and Initialize the Password Vault

Prerequisites

- Section 3.8.2, “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 3.8.2, “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 3.8.2, "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**

## 3.8.4. 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 3.8.2, “Create a Java Keystore to Store Sensitive Strings”
- Section 3.8.3, “Mask the Keystore Password and Initialize the Password Vault”

**Procedure 3.3. 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 3.8.2, “Create a Java Keystore to Store Sensitive Strings” and Section 3.8.3, "Mask the Keystore Password and Initialize the Password Vault".

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYSTORE_URL</td>
<td>The file system path or URI of the keystore file, usually called something like <code>vault.keystore</code></td>
</tr>
<tr>
<td>KEYSTORE_PASSWORD</td>
<td>The password used to access the keystore. This value should be masked.</td>
</tr>
<tr>
<td>KEYSTORE_ALIAS</td>
<td>The name of the keystore.</td>
</tr>
<tr>
<td>SALT</td>
<td>The salt used to encrypt and decrypt keystore values.</td>
</tr>
<tr>
<td>ITERATION_COUNT</td>
<td>The number of times the encryption algorithm is run.</td>
</tr>
</tbody>
</table>
The path to the directory from which the keystore commands are run. Typically the directory containing the password vault.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENC_FILE_DIR</td>
<td>The name of the host you are configuring</td>
</tr>
</tbody>
</table>

**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.

- **Managed Domain**

  ```bash
  /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**

  ```bash
  /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:

```bash
/core-service=vault:add(vault-options=[["KEYSTORE_URL" => "/home/user/vault/vault.keystore"], ("KEYSTORE_PASSWORD" => "MASK-3y28rCZ1cKR"), ("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 3.8.5, “Store and Retrieve Encrypted Sensitive Strings in the Java Keystore”.

**Report a bug**

**3.8.5. 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 3.8.2, “Create a Java Keystore to Store Sensitive Strings”
- Section 3.8.3, “Mask the Keystore Password and Initialize the Password Vault”
- Section 3.8.4, “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 3.4. 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 3.8.2, “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_ExmapleDS. 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
Shared
Key:N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMWN1MDMyNDdmNjI2TE10RV9CUkVBS3ZhdWx0
Configuration should be done as follows:
VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMWN1MDMyNDdmNjI2TE10RV9CUkVBS3ZhdWx0
********************************************
```

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.

```xml
...<subpackage 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/>
      <security>
        <user-name>sa</user-name>
        <password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMWN1MDMyNDdmNjI2TE10RV9CUkVBS3ZhdWx0}</password>
      </security>
    </datasource>
  </datasources>
...<password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMWN1MDMyNDdmNjI2TE10RV9CUkVBS3ZhdWx0}</password></subpackage>
```

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::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMWN1MDMyN0dNamI2TE10RV9CUkVBS3ZhdWx0}</password>
```

3.8.6. 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 3.11. 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
```
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
Shared
Key: OWY5M2I5NzctYzd0S000MmZhlWExZGYtNjczM2U5ZGUyOWIxTE10RV9CUkVBS3ZhdWx0
Configuration should be done as follows:
VAULT:: DS:: thePass:: OWY5M2I5NzctYzd0S000MmZhlWExZGYtNjczM2U5ZGUyOWIxTE10
RV9CUkVBS3ZhdWx0
********************************************************************************

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 3.12. Servlet Using a Vaulted Password

```java
package vaulterror.web;

import java.io.IOException;
```
Your servlet is now able to resolve the vaulted string.

Report a bug

3.9. LDAP

3.9.1. 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

3.9.2. 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 3.1. Attributes of an LDAP Outbound Connection

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>The name to identify this connection. This name is used in the security realm definition.</td>
</tr>
<tr>
<td>url</td>
<td>yes</td>
<td>The URL address of the directory server.</td>
</tr>
<tr>
<td>search-dn</td>
<td>yes</td>
<td>The fully distinguished name (DN) of the user authorized to perform searches.</td>
</tr>
<tr>
<td>search-credentials</td>
<td>yes</td>
<td>The password of the user authorized to perform searches.</td>
</tr>
<tr>
<td>initial-context-factory</td>
<td>no</td>
<td>The initial context factory to use when establishing the connection. Defaults to com.sun.jndi.ldap.LdapCtxFactory.</td>
</tr>
</tbody>
</table>

Example 3.13. 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

```
/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")
```

Example 3.14. XML Representing an LDAP Outbound Connection

```
<outbound-connections>
  <ldap name="ldap_connection" url="ldap://127.0.0.1:389" search-dn="cn=search,dc=acme,dc=com" search-credential="myPass" />
</outboundconnections>
```

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:

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

After escaping the ampersand character, the filter appears as:

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

### Example 3.15. 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"`

```xml
<security-realm name="TestRealm">
  <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 3.16. Add an LDAP Security Realm

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

```bash
/host=master/core-service=management/security-realm=ldap_security_realm/authentication=ldap:add(base-dn="DC=mycompany,DC=org", recursive=true, username-
```
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 3.17. 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=TestRealm)
```

Report a bug
CHAPTER 4. JAVA SECURITY MANAGER

4.1. 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

4.2. 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 or on the code’s signature.

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

Report a bug

4.3. 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.
- 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 4.1. Edit Configuration Files
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**
     \[ EAP_HOME/bin/domain.conf \]
   - **Standalone Server**
     \[ EAP_HOME/bin/standalone.conf \]

2. **Add the Java options at the end of the file.**
   Add the following line to a new line at the very end of the file. 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. You can modify the `-Djava.security.debug` to log more or less information, by specifying the debug level. The most verbose is `failure,access,policy`.
   ```bash
   JAVA_OPTS="$JAVA_OPTS -Djava.security.manager -Djboss.home.dir=$PWD/.. -Djava.security.policy==$PWD/server.policy -Djava.security.debug=failure"
   ```

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

**Report a bug**

### 4.4. 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/technology/tools/](http://docs.oracle.com/javase/6/docs/technology/tools/).

**Basic Information**

A security policy consists of the following configuration elements:

**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 "any principals".

**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.

**Procedure 4.2. 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 new policy.**
   To create a new 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.

**Permission Specific to JBoss EAP 6**

`org.jboss.security.SecurityAssociation.getPrincipalInfo`

Provides access to the `org.jboss.security.SecurityAssociation getPrincipal()` and `getCredential()` methods. The risk involved with using this runtime permission is the ability to see the current thread caller and credentials.

`org.jboss.security.SecurityAssociation.getSubject`

Provides access to the `org.jboss.security.SecurityAssociation getSubject()` method.

`org.jboss.security.SecurityAssociation.setPrincipalInfo`

Provides access to the `org.jboss.security.SecurityAssociation setPrincipal()`, `setCredential()`, `setSubject()`, `pushSubjectContext()`, and `popSubjectContext()` methods. The risk involved with using this runtime permission is the ability to set the current thread caller and credentials.

`org.jboss.security.SecurityAssociation.setServer`

Provides access to the `org.jboss.security.SecurityAssociation setServer()` method. The risk involved with using this runtime permission is the ability to enable or disable multi-thread storage of the caller principal and credential.

`org.jboss.security.SecurityAssociation.setRunAsRole`

Provides access to the `org.jboss.security.SecurityAssociation pushRunAsRole()`,...
popRunAsRole(), pushRunAsIdentity(), and popRunAsIdentity() methods. The risk involved with using this runtime permission is the ability to change the current caller run-as role principal.

**org.jboss.security.SecurityAssociation.accessContextInfo**

Provides access to the **org.jboss.security.SecurityAssociation accessContextInfo()** and accessContextInfo() getter and setter methods. This allows you to both set and get the current security context info.

**org.jboss.naming.JndiPermission**

Provides special permissions to files and directories in a specified JNDI tree path, or recursively to all files and subdirectories. A JndiPermission consists of a pathname and a set of valid permissions related to the file or directory.

The available permissions include:

- bind
- rebind
- unbind
- lookup
- list
- listBindings
- createSubcontext
- all

Pathnames ending in /* indicate that the specified permissions apply to all files and directories of the pathname. Pathnames ending in /- indicate recursive permissions to all files and subdirectories of the pathname. Pathnames consisting of the special token <<ALL BINDINGS>> matches any file in any directory.

**org.jboss.security.srp.SRPPermission**

A custom permission class for protecting access to sensitive SRP information like the private session key and private key. This permission does not have any actions defined. The getSessionFactory() target provides access to the private session key which results from the SRP negotiation. Access to this key allows you to encrypt and decrypt messages that have been encrypted with the session key.

**org.hibernate.secure.HibernatePermission**

This permission class provides basic permissions to secure Hibernate sessions. The target for this property is the entity name. The available actions include:

- insert
- delete
- update
- read
org.jboss.metadata.spi.stack.MetaDataStackPermission
Provides a custom permission class for controlling how callers interact with the metadata stack. The available permissions are:

- modify
- push (onto the stack)
- pop (off the stack)
- peek (onto the stack)
- * (all)

org.jboss.config.spi.ConfigurationPermission
Secures setting of configuration properties. Defines only permission target names, and no actions. The targets for this property include:

- <property name> (the property this code has permission to set)
- * (all properties)

org.jboss.kernel.KernelPermission
Secures access to the kernel configuration. Defines only permission target names and no actions. The targets for this property include:

- access (to the kernel configuration)
- configure (implies access)
- * (all)

org.jboss.kernel.plugins.util.KernelLocatorPermission
Secures access to the kernel. Defines only permission target names and no actions. The targets for this property include:

- kernel
- * (all)

Report a bug

4.5. 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 4.3. 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 5. PATCH INSTALLATION

5.1. ABOUT PATCHING MECHANISMS

JBoss security and bug patches are released in two forms.

- Planned updates: As part of a micro, minor or major upgrade of an existing product.
- Asynchronous updates: As a one off patch which is released outside the normal upgrade cycle of the existing product.

Deciding whether a patch is released as part of a planned update or an out-of-cycle one-off depends on the severity of the flaw being fixed. Flaws of low impact are typically deferred, to be resolved in the next minor release of the affected products. Flaws of moderate or higher impact are typically addressed in order of importance as an update to the product with an asynchronous release and contain only a resolution to the flaw at hand.

The severity of a security flaw is based on the assessment of the bug by the Security Response Team at Red Hat, combined with several consistent factors:

- How easily can a flaw be exploited?
- What kind of damage can be done if exploited?
- Are there typically other factors involved that lower the impact of the flaw (such as firewalls, Security-Enhanced Linux, compiler directives, and so forth)?

Red Hat maintains a mailing list for notifying subscribers about security related flaws. See Section 5.2, “Subscribe to Patch Mailing Lists”

For more information on how Red Hat rates JBoss security flaws, please click on the following link: http://securityblog.redhat.com/2012/09/19/how-red-hat-rates-jboss-security-flaws/

Report a bug

5.2. 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 5.1. 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 completely 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 account, you will be subscribed to receive security related announcements from the JBoss patch mailing list.

[Report a bug](#)

### 5.3. INSTALL PATCHES IN ZIP FORM

**Summary**

JBoss security patches are distributed in two forms: zip (for all products) and RPM (for a subset of products). JBoss bug fix patches are only distributed in zip format. This task describes the steps you need to take to install the patches (security or bug fixes) via the zip format.

**Prerequisites**

- Valid access and subscription to the Red Hat Customer Portal.
- A current subscription to a JBoss product installed in a zip format.

**Procedure 5.2. Apply a patch to a JBoss product via the zip method**

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 zip distributions of JBoss products, the errata includes a link to a URL on the Customer Portal where the patch zip can be downloaded. This download contains the patched versions of existing JBoss products and only includes the files that have been changed from the previous install.

**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.

   **NOTE**

   Only security patches are announced on the JBoss watch mailing list.

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 patch from the Red Hat Customer Portal.

4. The downloadable zip file from the customer portal will contain all the files required to fix the security issue or bug. Download this patch zip file in the same location as your JBoss product.

5. Unzip the patch file in the same location where the JBoss product is installed. The patched versions overwrite the existing files.

Result
The JBoss product is patched with the latest update using the zip format.

Report a bug

5.4. 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. This RPM update method is used to ship security asynchronous patches and macro/minor/major product updates only.

Prerequisites
- A valid subscription to the Red Hat Network.
- A current subscription to a JBoss product installed via an RPM package.

Procedure 5.3. Apply a patch to a JBoss product via the RPM method

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.

For RPM distributions of JBoss products, the errata include references to the updated RPM packages. The patch can be installed by using yum or another RPM tool to update the relevant packages.

**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
```

or a similar command to install the patch.

Result

The JBoss product is patched with the latest update using the RPM format.

Report a bug

5.5. 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 5.1. Severity Ratings of JBoss Security Patches

<table>
<thead>
<tr>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>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.</td>
</tr>
<tr>
<td>Important</td>
<td>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.</td>
</tr>
<tr>
<td>Moderate</td>
<td>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.</td>
</tr>
<tr>
<td>Low</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 5.1. 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
CHAPTER 6. SECURITY DOMAINS

6.1. 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, which contains information about actions which are allowed or prohibited. 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.

6.2. 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 6.3, “About Authentication”
- Section 6.5, “About Authorization” and access control
- Section 6.7, “About Security Auditing”
- Section 6.9, “About Security Mapping” of principals, roles, and attributes

6.3. 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

6.4. CONFIGURE AUTHENTICATION IN A SECURITY DOMAIN

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

Procedure 6.1. 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.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.</td>
</tr>
<tr>
<td>requisite</td>
<td>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).</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>sufficient</td>
<td>The LoginModule is not required to succeed. If it succeeds, control immediately returns to the application (authentication does not proceed down the LoginModule list). If it fails, authentication continues down the LoginModule list.</td>
</tr>
<tr>
<td>optional</td>
<td>The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.</td>
</tr>
</tbody>
</table>

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

6.5. 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.

6.6. CONFIGURE AUTHORIZATION IN A SECURITY DOMAIN

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

Procedure 6.2. 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.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required</td>
<td>The LoginModule is required to succeed. If it succeeds or fails, authorization still continues to proceed down the LoginModule list.</td>
</tr>
<tr>
<td>Requisite</td>
<td>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).</td>
</tr>
<tr>
<td>Sufficient</td>
<td>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).</td>
</tr>
</tbody>
</table>

Report a bug
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

### 6.7. 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

### 6.8. CONFIGURE SECURITY AUDITING

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

**Procedure 6.3. 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.

6.9. **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.

6.10. **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.


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 7. SSL ENCRYPTION

7.1. 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

7.2. 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 7.3, “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 7.1. 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 7.1. 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 7.2. Management CLI Command
   /profile=default/subsystem=web/connector=HTTPS/ssl=configuration:add(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 7.4, “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

7.3. 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 7.2. 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.

   ```bash
   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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-genkeypair</code></td>
<td>The <code>keytool</code> command to generate a key pair containing a public and private key.</td>
</tr>
<tr>
<td><code>-alias</code></td>
<td>The alias for the keystore. This value is arbitrary, but the alias <code>jboss</code> is the default used by the JBoss Web server.</td>
</tr>
<tr>
<td><code>-keyalg</code></td>
<td>The key pair generation algorithm. In this case it is <code>RSA</code>.</td>
</tr>
<tr>
<td><code>-keystore</code></td>
<td>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 <code>server.keystore</code>.</td>
</tr>
<tr>
<td><code>-storepass</code></td>
<td>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 <code>mykeystorepass</code>. If you omit this parameter, you will be prompted to enter it when you execute the command.</td>
</tr>
</tbody>
</table>
**-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.com,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", then **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.

   ```bash
   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.**
   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 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

### 7.4. 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>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The display name of the SSL connector.</td>
<td><code>/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=name, value=https)</code></td>
</tr>
<tr>
<td>verify-client</td>
<td>Set to <code>true</code> to require a valid certificate chain from the client before accepting a connection. Set to <code>want</code> if you want the SSL stack to request a client Certificate, but not fail if one is not presented. Set to <code>false</code> (the default) to not require a certificate chain unless the client requests a resource protected by a security constraint that uses <code>CLIENT-CERT</code> authentication.</td>
<td><code>/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=verify-client, value=want)</code></td>
</tr>
<tr>
<td>verify-depth</td>
<td>The maximum number of intermediate certificate issuers checked before deciding that the clients do not have a valid certificate. The default value is 10.</td>
<td><code>/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=verify-depth, value=10)</code></td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>CLI Command</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>certificate-key-file</td>
<td>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 <code>.keystore</code> file in the home directory of the user running JBoss EAP 6. If your <code>keystoreType</code> does not use a file, set the parameter to an empty string.</td>
<td>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=certificate-key-file,value=../domain/configuration/server.keystore)</td>
</tr>
<tr>
<td>certificate-file</td>
<td>If you use OpenSSL encryption, set the value of this parameter to the path to the file containing the server certificate.</td>
<td>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=certificate-file,value=server.crt)</td>
</tr>
<tr>
<td>password</td>
<td>The password for both the trustore and keystore. In the following example, replace <code>PASSWORD</code> with your own password.</td>
<td>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=password,value=PASSWORD)</td>
</tr>
<tr>
<td>protocol</td>
<td>The version of the SSL protocol to use. Supported values include <code>SSLv2</code>, <code>SSLv3</code>, <code>TLSv1</code>, <code>SSLv2+SSLv3</code>, and <code>ALL</code>. The default is <code>ALL</code>.</td>
<td>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=protocol,value=ALL)</td>
</tr>
<tr>
<td>cipher-suite</td>
<td>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.</td>
<td>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=cipher-suite,value=&quot;TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA&quot;)</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>CLI Command</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>key.alias</td>
<td>The alias used to for the server certificate in the keystore. In the following example, replace <code>KEY_ALIAS</code> with your certificate's alias.</td>
<td><code>/profile=default/subsystem=web/connector=https/ssl=configuration/:write-attribute(name=key-alias,value=KEY_ALIAS)</code></td>
</tr>
<tr>
<td>truststore-type</td>
<td>The type of the truststore. Various types of keystores are available, including PKCS12 and Java's standard JKS.</td>
<td><code>/profile=default/subsystem=web/connector=https/ssl=configuration/:write-attribute(name=truststore-type,value=jks)</code></td>
</tr>
<tr>
<td>keystore-type</td>
<td>The type of the keystore. Various types of keystores are available, including PKCS12 and Java's standard JKS.</td>
<td><code>/profile=default/subsystem=web/connector=https/ssl=configuration/:write-attribute(name=keystore-type,value=jks)</code></td>
</tr>
<tr>
<td>ca-certificate-file</td>
<td>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.</td>
<td><code>/profile=default/subsystem=web/connector=https/ssl=configuration/:write-attribute(name=certificate-file,value=ca.crt)</code></td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>CLI Command</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ca-certificate-password</td>
<td>The Certificate password for the <strong>ca-certificate-file</strong>. In the following example, replace the MASKED_PASSWORD with your own masked password.</td>
<td><code>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=ca-certificate-password,value=MASKED_PASSWORD)</code></td>
</tr>
<tr>
<td>ca-revocation-url</td>
<td>A file or URL which contains the revocation list. It refers to the <strong>crlFile</strong> for JSSE or the <strong>SSLCARevocationFile</strong> for SSL.</td>
<td><code>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=ca-revocation-url,value=ca.crl)</code></td>
</tr>
<tr>
<td>session-cache-size</td>
<td>The size of the SSLSession cache. This attribute applies only to JSSE connectors. The default is 0, which specifies an unlimited cache size.</td>
<td><code>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=session-cache-size,value=100)</code></td>
</tr>
<tr>
<td>session-timeout</td>
<td>The number of seconds before a cached SSLSession expires. This attribute applies only to JSSE connectors. The default is <strong>86400</strong> seconds, which is 24 hours.</td>
<td><code>/profile=default/sub system=web/connector =HTTPS/ssl=configuration/:write-attribute(name=session-timeout,value=43200)</code></td>
</tr>
</tbody>
</table>

**Report a bug**
8.1. 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 user, password, and role 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.roles** 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.

8.2. 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.
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.

8.3. 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 9. SUBSYSTEM CONFIGURATION

9.1. TRANSACTION SUBSYSTEM CONFIGURATION

9.1.1. Configure the ORB for JTS Transactions

In a default installation of JBoss EAP 6, the ORB is disabled. You can enable the ORB using the command-line Management CLI.

**NOTE**

In a managed domain, the JacORB subsystem is available in **full** and **full-ha** profiles only. In a standalone server, it is available when you use the **standalone-full.xml** or **standalone-full-ha.xml** configurations.

Procedure 9.1. Configure the ORB using the Management Console

1. **View the profile settings.**
   Select **Profiles** (managed domain) or **Profile** (standalone server) from the top right of the management console. If you use a managed domain, select either the **full** or **full-ha** profile from the selection box at the top left.

2. **Modify the Initializers Settings**
   Expand the **Subsystems** menu at the left, if necessary. Expand the **Container** sub-menu and click **JacORB**.

   In the form that appears in the main screen, select the **Initializers** tab and click the **Edit** button.

   Enable the security interceptors by setting the value of **Security** to **on**.

   To enable the ORB for JTS, set the **Transaction Interceptors** value to **on**, rather than the default **spec**.

   Refer to the **Need Help?** link in the form for detailed explanations about these values. Click **Save** when you have finished editing the values.

3. **Advanced ORB Configuration**
   Refer to the other sections of the form for advanced configuration options. Each section includes a **Need Help?** link with detailed information about the parameters.

Configure the ORB using the Management CLI

You can configure each aspect of the ORB using the Management CLI. The following commands configure the initializers to the same values as the procedure above, for the Management Console. This is the minimum configuration for the ORB to be used with JTS.

These commands are configured for a managed domain using the **full** profile. If necessary, change the profile to suit the one you need to configure. If you use a standalone server, omit the **/profile=full** portion of the commands.

**Example 9.1. Enable the Security Interceptors**
Example 9.2. Enable the ORB for JTS

/profile=full/subsystem=jacorb/:write-attribute(name=security,value=on)
/profile=full/subsystem=jacorb/:write-attribute(name=transactions,value=on)

Report a bug

9.2. JMS CONFIGURATION

9.2.1. Reference for HornetQ Configuration Attributes

The JBoss EAP 6 implementation of HornetQ exposes the following attributes for configuration. You can use the Management CLI in particular to exposure the configurable or viewable attributes with the read-resource operation.

Example 9.3. Example

[standalone@localhost:9999 /] /subsystem=messaging/hornetq-server=default:read-resource

Table 9.1. HornetQ Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Example Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow-failback</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>async-connection-execution-enabled</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>backup</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>cluster-password</td>
<td>somethingsecure</td>
<td>STRING</td>
</tr>
<tr>
<td>cluster-user</td>
<td>HORNETQ.CLUSTER.ADMIN.USER</td>
<td>STRING</td>
</tr>
<tr>
<td>clustered</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>connection-ttl-override</td>
<td>-1</td>
<td>LONG</td>
</tr>
<tr>
<td>Attribute</td>
<td>Example Value</td>
<td>Type</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>create-bindings-dir</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>create-journal-dir</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>failback-delay</td>
<td>5000</td>
<td>LONG</td>
</tr>
<tr>
<td>failover-on-shutdown</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>id-cache-size</td>
<td>2000</td>
<td>INT</td>
</tr>
<tr>
<td>jmx-domain</td>
<td>org.hornetq</td>
<td>STRING</td>
</tr>
<tr>
<td>jmx-management-enabled</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>journal-buffer-size</td>
<td>100</td>
<td>LONG</td>
</tr>
<tr>
<td>journal-buffer-timeout</td>
<td>100</td>
<td>LONG</td>
</tr>
<tr>
<td>journal-compact-min-files</td>
<td>10</td>
<td>INT</td>
</tr>
<tr>
<td>journal-compact-percentage</td>
<td>30</td>
<td>INT</td>
</tr>
<tr>
<td>journal-file-size</td>
<td>102400</td>
<td>LONG</td>
</tr>
<tr>
<td>journal-max-io</td>
<td>1</td>
<td>INT</td>
</tr>
<tr>
<td>journal-min-files</td>
<td>2</td>
<td>INT</td>
</tr>
<tr>
<td>journal-sync-non-transactional</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>journal-sync-transactional</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>journal-type</td>
<td>ASYNCIO</td>
<td>STRING</td>
</tr>
<tr>
<td>Attribute</td>
<td>Example Value</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>live-connector-ref</td>
<td>reference</td>
<td>STRING</td>
</tr>
<tr>
<td>log-journal-write-rate</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>management-address</td>
<td>jms.queue.hornetq.management</td>
<td>STRING</td>
</tr>
<tr>
<td>management-notification-address</td>
<td>hornetq.notifications</td>
<td>STRING</td>
</tr>
<tr>
<td>memory-measure-interval</td>
<td>-1</td>
<td>LONG</td>
</tr>
<tr>
<td>memory-warning-threshold</td>
<td>25</td>
<td>INT</td>
</tr>
<tr>
<td>message-counter-enabled</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>message-counter-max-day-history</td>
<td>10</td>
<td>INT</td>
</tr>
<tr>
<td>message-counter-sample-period</td>
<td>10000</td>
<td>LONG</td>
</tr>
<tr>
<td>message-expiry-scan-period</td>
<td>30000</td>
<td>LONG</td>
</tr>
<tr>
<td>message-expiry-thread-priority</td>
<td>3</td>
<td>INT</td>
</tr>
<tr>
<td>page-max-concurrent-io</td>
<td>5</td>
<td>INT</td>
</tr>
<tr>
<td>perf-blast-pages</td>
<td>-1</td>
<td>INT</td>
</tr>
<tr>
<td>persist-delivery-count-before-delivery</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>persist-id-cache</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>persistence-enabled</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>Attribute</td>
<td>Example Value</td>
<td>Type</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>remoting-interceptors</td>
<td>undefined</td>
<td>LIST</td>
</tr>
<tr>
<td>run-sync-speed-test</td>
<td>false</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>scheduled-thread-pool-max-size</td>
<td>5</td>
<td>INT</td>
</tr>
<tr>
<td>security-domain</td>
<td>other</td>
<td>STRING</td>
</tr>
<tr>
<td>security-enabled</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>security-invalidation-interval</td>
<td>10000</td>
<td>LONG</td>
</tr>
<tr>
<td>server-dump-interval</td>
<td>-1</td>
<td>LONG</td>
</tr>
<tr>
<td>shared-store</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>started</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>thread-pool-max-size</td>
<td>30</td>
<td>INT</td>
</tr>
<tr>
<td>transaction-timeout</td>
<td>300000</td>
<td>LONG</td>
</tr>
<tr>
<td>transaction-timeout-scan-period</td>
<td>1000</td>
<td>LONG</td>
</tr>
<tr>
<td>version</td>
<td>2.2.16.Final (HQ_2_2_16_FINAL, 122)</td>
<td>STRING</td>
</tr>
<tr>
<td>wild-card-routing-enabled</td>
<td>true</td>
<td>BOOLEAN</td>
</tr>
</tbody>
</table>
WARNING

The value of `journal-file-size` must be higher than the size of message sent to server, or the server won't be able to store the message.

Report a bug
CHAPTER 10. WEB, HTTP CONNECTORS, AND HTTP CLUSTERING

10.1. CONFIGURE A MOD_CLUSTER WORKER NODE

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 10.1. 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/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,

       ```xml
       <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::ODVmYmJjNGMtZDU2ZC00YmNlWE4ODMtZjQ1NWNmNU4ZDc1TE10RV9CUkVBS3ZhdWx0`.

       You can find more information on the vault in the Password Vaults for Sensitive Strings section of this guide starting here: Section 3.8.1, “About Securing Sensitive Strings in Clear-Text Files”.
     - ii. Add the following block of XML code directly below the `<security-realm name="ManagementRealm">` line.

       ```xml
       <server-identities>
       <secret value="${VAULT::secret::password::ODVmYmJjNGMtZDU2ZC00YmNlWE4ODMtZjQ1NWNmNU4ZDc1TE10RV9CUkVBS3ZhdWx0}"/>
       </server-identities>
       ```
     - Be sure to replace the secret value with the masked password generated in the previous step.

   - **Specify the password as a system property.**
     - i. Add the following block of XML code directly below the `<security-realm name="ManagementRealm">` line

       ```xml
       <server-identities>
       ```

   **NOTE**

   When creating a password in the vault, it must be specified in plain text, not Base64-encoded.
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.

Report a bug
CHAPTER 11. NETWORK SECURITY

11.1. 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 3.5, “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 11.2, “Specify Which Network Interface JBoss EAP 6 Uses”
- Section 11.3, “Configure Network Firewalls to Work with JBoss EAP 6”

Report a bug

11.2. 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.


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 11.1. Specify the public interface.**
   
   ```
   EAP_HOME/bin/domain.sh -b 10.1.1.1
   ```

   **Example 11.2. Specify the management interface.**
   
   ```
   EAP_HOME/bin/domain.sh -bmanagement=10.1.1.1
   ```

   **Example 11.3. Specify different addresses for each interface.**
   
   ```
   EAP_HOME/bin/domain.sh -bmanagement=127.0.0.1 -b 10.1.1.1
   ```

   **Example 11.4. 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](https://issues.jboss.org/)

**11.3. 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 needs to 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. Refer to [Section 11.4, “Network Ports Used By JBoss EAP 6”](#) to determine the list of ports for your situation.

- 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
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 11.1. 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-hand side of the screen, a series of menus is shown. The bottom menu heading is General Configuration. Click the Socket Binding Groups 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, six 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 needs of your environment, some of the ports may need to be accessible across your firewall. If you are unsure of the purpose of a socket binding, refer to Section 11.4, “Network Ports Used By JBoss EAP 6” for a list of the default socket bindings and their purposes.

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. Consult [Section 11.4, “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, make sure 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**

### 11.4. 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 11.1. Reference of the default socket bindings

<table>
<thead>
<tr>
<th>Name</th>
<th>Port</th>
<th>Multicast Port</th>
<th>Description</th>
<th>full-ha-sockets</th>
<th>full-sockets</th>
<th>ha-socket</th>
<th>standard-socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>ajp</td>
<td>8009</td>
<td></td>
<td>Apache JServ Protocol. Used for HTTP clustering and load balancing.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>http</td>
<td>8080</td>
<td></td>
<td>The default port for deployed web applications.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>https</td>
<td>8443</td>
<td></td>
<td>SSL-encrypted connection between deployed web applications and clients.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>jacorb</td>
<td>3528</td>
<td></td>
<td>CORBA services for JTS transactions and other ORB-dependent services.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>jacorb-ssl</td>
<td>3529</td>
<td></td>
<td>SSL-encrypted CORBA services.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Name</td>
<td>Port</td>
<td>Multicast Port</td>
<td>Description</td>
<td>full-ha-sockets</td>
<td>full-sockets</td>
<td>ha-socket</td>
<td>standard-socket</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>jgroups-diagnostics</td>
<td>7500</td>
<td></td>
<td>Multicast. Used for peer discovery in HA clusters.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>jgroups-mping</td>
<td>45700</td>
<td></td>
<td>Multicast. Used to discover initial membership in a HA cluster.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>jgroups-tcp</td>
<td>7600</td>
<td></td>
<td>Unicast peer discovery in HA clusters using TCP.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>jgroups-tcp-fd</td>
<td>57600</td>
<td></td>
<td>Used for HA failure detection over TCP.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>jgroups-udp</td>
<td>55200</td>
<td>45688</td>
<td>Unicast peer discovery in HA clusters using UDP.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>jgroups-udp-fd</td>
<td>54200</td>
<td></td>
<td>Used for HA failure detection over UDP.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>messaging</td>
<td>5445</td>
<td></td>
<td>JMS service.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>messaging-group</td>
<td></td>
<td></td>
<td>Referenced by HornetQ JMS broadcast and discovery groups.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>messaging-throughput</td>
<td>5455</td>
<td></td>
<td>Used by JMS Remoting.</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>mod_cluster</td>
<td>23364</td>
<td></td>
<td>Multicast port for communication between JBoss EAP 6 and the HTTP load balancer.</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Name</td>
<td>Port</td>
<td>Multicast Port</td>
<td>Description</td>
<td>full-sockets</td>
<td>full-sockets</td>
<td>ha-socket</td>
<td>standard-socket</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>osgi-http</td>
<td>8090</td>
<td></td>
<td>Used by internal components which use the OSGi subsystem.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>remoting</td>
<td>4447</td>
<td></td>
<td>Used for remote EJB invocation.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>txn-recovery-environment</td>
<td>4712</td>
<td></td>
<td>The JTA transaction recovery manager.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>txn-status-manager</td>
<td>4713</td>
<td></td>
<td>The JTA / JTS transaction manager.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**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
PART III. SECURING APPLICATIONS
CHAPTER 12. APPLICATION SECURITY

12.1. 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.

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 12.1. 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 12.2. 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

12.2. DATASOURCE SECURITY

12.2.1. 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:

- Password vaults: Section 3.8.1, "About Securing Sensitive Strings in Clear-Text Files".

**Example 12.1. Security Domain Example**

```
<security>
  <security-domain>mySecurityDomain</security-domain>
</security>
```

**Example 12.2. Password Vault Example**

```
<security>
  <user-name>admin</user-name>
  <password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4MmEtMwN1MDMyNDdmNmI2TEl0RV9CUkVBS3ZhdWx0}</password>
</security>
```

Report a bug

12.3. EJB APPLICATION SECURITY
12.3.1. Security Identity

12.3.1.1. 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 12.3.1.2, “Set the Security Identity of an EJB”.

Report a bug

12.3.1.2. Set the Security Identity of an EJB

Example 12.3. 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.

```xml
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>ASessionBean</ejb-name>
      <!-- ... -->
      <security-identity>
        <use-caller-identity/>
      </security-identity>
    </session>
    <!-- ... -->
  </enterprise-beans>
</ejb-jar>
```

Example 12.4. 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.

```xml
<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>`.

```xml
<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 12.3.1.1, “About EJB Security Identity”
- Section A.6, “EJB Security Parameter Reference”

Report a bug

12.3.2. EJB Method Permissions

12.3.2.1. About EJB Method Permissions

EJB provides a `<method-permission>` 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 12.3.2.2, “Use EJB Method Permissions”.

Report a bug

12.3.2.2. 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 12.5. 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>
</method-permission>
```

**Example 12.6. 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>
  <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-permission>
```

**Example 12.7. 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>
</method-permission>
```
Example 12.8. Completely exclude specific EJB methods from being used

```xml
<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 12.9. A complete <assembly-descriptor> containing several <method-permission> blocks

```xml
<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>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```
12.3.3. EJB Security Annotations

12.3.3.1. About EJB Security Annotations

EJBs use security annotations to pass information about security to the deployer. These include:

@DeclareRoles
Declares which roles are available.

@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
Specifies which method permissions are allowed. For information about method permissions, refer to Section 12.3.2.1, “About EJB Method Permissions”.

@RolesAllowed, @PermitAll, @DenyAll

Specifies which method permissions are allowed. For information about method permissions, refer to Section 12.3.2.1, “About EJB Method Permissions”.

@RunAs

Configures the propagated security identity of a component.

For more information, refer to Section 12.3.3.2, “Use EJB Security Annotations”.

12.3.3.2. 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 12.3.2.2, “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 12.10. Security Annotations Example

```java
@Stateless
@RolesAllowed(\{"admin"\})
@SecurityDomain("other")
public class WelcomeEJB implements Welcome {
    @PermitAll
    public String WelcomeEveryone(String msg) {
        return "Welcome to " + msg;
    }
    @RunAs("tempemployee")
```
In this code, all roles can access method `WelcomeEveryone`. The `GoodBye` method runs as the `tempemployee` role. Only the `admin` role can access method `GoodbyeAdmin`, and any other methods with no security annotation.

```java
public String GoodBye(String msg) {
    return "Goodbye, " + msg;
}
public String GoodbyeAdmin(String msg) {
    return "See you later, " + msg;
}
```

12.3.4. Remote Access to EJBs

12.3.4.1. 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.

**12.3.4.2. 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
12.3.4.3. 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

12.3.4.4. 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.

```xml
<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:

```xml
<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 12.1. Worker Thread Pool Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-threads</td>
<td>The number of read threads to create for the remoting worker. Defaults to 1.</td>
<td><code>/profile=default/subsystem=remoting/:write-attribute(name=worker-read-threads,value=1)</code></td>
</tr>
<tr>
<td>write-threads</td>
<td>The number of write threads to create for the remoting worker. Defaults to 1.</td>
<td><code>/profile=default/subsystem=remoting/:write-attribute(name=worker-write-threads,value=1)</code></td>
</tr>
<tr>
<td>task-keepalive</td>
<td>The number of milliseconds to keep non-core remoting worker task threads alive. Defaults to 60.</td>
<td><code>/profile=default/subsystem=remoting/:write-attribute(name=worker-task-keepalive,value=60)</code></td>
</tr>
<tr>
<td>task-max-threads</td>
<td>The maximum number of threads for the remoting worker task thread pool. Defaults to 16.</td>
<td><code>/profile=default/subsystem=remoting/:write-attribute(name=worker-task-max-threads,value=16)</code></td>
</tr>
<tr>
<td>task-core-threads</td>
<td>The number of core threads for the remoting worker task thread pool. Defaults to 4.</td>
<td><code>/profile=default/subsystem=remoting/:write-attribute(name=worker-task-core-threads,value=4)</code></td>
</tr>
<tr>
<td>task-limit</td>
<td>The maximum number of remoting worker tasks to allow before rejecting. Defaults to 16384.</td>
<td><code>/profile=default/subsystem=remoting/:write-attribute(name=worker-task-limit,value=16384)</code></td>
</tr>
</tbody>
</table>

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 12.2. Connector Attributes
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-binding</td>
<td>The name of the socket binding to use for this connector.</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=socket-binding,value=remoting)</code></td>
</tr>
<tr>
<td>authentication-provider</td>
<td>The Java Authentication Service Provider Interface for Containers (JASPI</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=authentication-provider,value=myProvider)</code></td>
</tr>
<tr>
<td>security-realm</td>
<td>Optional. The security realm which contains your application’s users,</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=security-realm,value=ApplicationRealm)</code></td>
</tr>
<tr>
<td></td>
<td>passwords, and roles. An EJB or Web Application can authenticate against a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security realm. ApplicationRealm is available in a default JBoss EAP 6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>installation.</td>
<td></td>
</tr>
</tbody>
</table>

Table 12.3. Connector Elements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>sasl</td>
<td>Enclosing element for Simple Authentication and Security Layer (SASL)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>authentication mechanisms</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>Contains one or more <code>&lt;property&gt;</code> elements, each with a <code>name</code> attribute</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/property=myProp/:add(value=myPropValue)</code></td>
</tr>
<tr>
<td></td>
<td>and an optional <code>value</code> attribute.</td>
<td></td>
</tr>
</tbody>
</table>

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 12.4. Outbound Connection Elements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>outbound-connection</td>
<td>Generic outbound connection.</td>
<td>/profile=default/subsystem=remoting/outbound-connection=my-connection/:add(uri=<a href="http://my-connection">http://my-connection</a>)</td>
</tr>
<tr>
<td>local-outbound-connection</td>
<td>Outbound connection with a implicit local:// URI scheme.</td>
<td>/profile=default/subsystem=remoting/local-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting2)</td>
</tr>
<tr>
<td>remote-outbound-connection</td>
<td>Outbound connections for remote:// URI scheme, using basic/digest authentication with a security realm.</td>
<td>/profile=default/subsystem=remoting/remote-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting,username=myUser,security-realm=ApplicationRealm)</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>include-mechanisms</td>
<td>Contains a <strong>value</strong> attribute, which is a space-separated list of SASL mechanisms.</td>
<td>/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=include-mechanisms,value=[&quot;DIGEST&quot;,&quot;PLAIN&quot;,&quot;GSSAPI&quot;])</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
<td>CLI Command</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>qop</td>
<td>Contains a <strong>value</strong> attribute, which is a space-separated list of SASL Quality of protection values, in decreasing order of preference.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=qop,value=['&quot;auth&quot;'])</td>
</tr>
<tr>
<td>strength</td>
<td>Contains a <strong>value</strong> attribute, which is a space-separated list of SASL cipher strength values, in decreasing order of preference.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=strength,value=['&quot;medium&quot;'])</td>
</tr>
<tr>
<td>reuse-session</td>
<td>Contains a <strong>value</strong> attribute which is a boolean value. If true, attempt to reuse sessions.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=reuse-session,value=false)</td>
</tr>
<tr>
<td>server-auth</td>
<td>Contains a <strong>value</strong> attribute which is a boolean value. If true, the server authenticates to the client.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=server-auth,value=false)</td>
</tr>
<tr>
<td>policy</td>
<td>An enclosing element which contains zero or more of the following elements, which each take a single <strong>value</strong>.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:add</td>
</tr>
<tr>
<td></td>
<td>• forward-secrecy – whether mechanisms are required to implement forward secrecy (breaking into one session will not automatically provide information for breaking into future sessions)</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:add</td>
</tr>
</tbody>
</table>

---

**Attribute**

<table>
<thead>
<tr>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains a <strong>value</strong> attribute, which is a space-separated list of SASL Quality of protection values, in decreasing order of preference.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=qop,value=['&quot;auth&quot;'])</td>
</tr>
<tr>
<td>Contains a <strong>value</strong> attribute, which is a space-separated list of SASL cipher strength values, in decreasing order of preference.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=strength,value=['&quot;medium&quot;'])</td>
</tr>
<tr>
<td>Contains a <strong>value</strong> attribute which is a boolean value. If true, attempt to reuse sessions.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=reuse-session,value=false)</td>
</tr>
<tr>
<td>Contains a <strong>value</strong> attribute which is a boolean value. If true, the server authenticates to the client.</td>
<td>/profile=default/sub system=remoting/connector=remoting-connector/security=sasl:write-attribute(name=server-auth,value=false)</td>
</tr>
</tbody>
</table>

---

**Security Guide**

108
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>CLI Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-active</td>
<td>Whether mechanisms susceptible to non-dictionary attacks are permitted. A value of false permits, and true denies.</td>
<td><code>connector/security=asl/sasl-policy=policy:write-attribute(name=forward-secrecy,value=true)</code></td>
</tr>
<tr>
<td>no-anonymous</td>
<td>Whether mechanisms that accept anonymous login are permitted. A value of false permits, and true denies.</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/security=asl/sasl-policy=policy:write-attribute(name=no-active,value=false)</code></td>
</tr>
<tr>
<td>no-dictionary</td>
<td>Whether mechanisms susceptible to passive dictionary attacks are allowed. A value of false permits, and true denies.</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/security=asl/sasl-policy=policy:write-attribute(name=no-dictionary,value=true)</code></td>
</tr>
<tr>
<td>no-plain-text</td>
<td>Whether mechanisms which are susceptible to simple plain passive attacks are allowed. A value of false permits, and true denies.</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/security=asl/sasl-policy=policy:write-attribute(name=no-plain-text,value=false)</code></td>
</tr>
<tr>
<td>pass-credentials</td>
<td>Whether mechanisms which pass client credentials are allowed.</td>
<td><code>/profile=default/subsystem=remoting/connector=remoting-connector/security=asl/sasl-policy=policy:write-attribute(name=pass-credentials,value=true)</code></td>
</tr>
</tbody>
</table>
### Example 12.11. Example Configurations

This example shows the default remoting subsystem that ships with JBoss EAP 6.

```xml
<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.

```xml
<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>
    </sasl>
    <properties>
      <property name="myprop1" value="1"/>
      <property name="myprop2" value="2"/>
    </properties>
  </connector>
</subsystem>
```
Configuration Aspects Not Yet Documented

- JNDI and Multicast Automatic Detection

Report a bug

12.3.4.5. 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.

  ```properties
  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

12.3.4.6. Add a New Security Realm

1. Run the Management CLI.

```xml
<subsystem xmlns="urn:jboss:domain:server:5.0">
  <sasl>
    <authentication-provider name="myprovider" />
    <properties>
      <property name="myprop3" value="propValue" />
    </properties>
  </authentication-provider>
  <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>
  </connector>
</subsystem>
```
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.
   
   ```
   /host=master/core-service=management/security-realm=MyDomainRealm/authentication=properties:add(path=myfile.properties)
   ```

**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.

12.3.4.7. 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.

12.3.4.8. 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.

12.4. JAX-RS APPLICATION SECURITY

12.4.1. 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 12.3. 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:

   ```xml
   <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:

   ```xml
   <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:

   ```xml
   <security-constraint>
     <web-resource-collection>
       <web-resource-name>Resteasy</web-resource-name>
       <url-pattern>/PATH</url-pattern>
     </web-resource-collection>
   </security-constraint>
   ```
Result
Role-based security has been enabled within the application, with a set of defined roles.

Example 12.12. Example Role-Based Security Configuration

```xml
<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>
```

12.4.2. 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 12.4. Secure a JAX-RS Web Service using Supported Security Annotations
1. Enable role-based security. For more information, refer to: Section 12.4.1, "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.

12.5. SECURE REMOTE PASSWORD PROTOCOL

12.5.1. About Secure Remote Password Protocol (SRP)

The Secure Remote Password (SRP) protocol is an implementation of a public key exchange handshake described in the Internet Standards Working Group Request For Comments 2945 (RFC2945). The RFC2945 abstract states:

This document describes a cryptographically strong network authentication mechanism known as the Secure Remote Password (SRP) protocol. This mechanism is suitable for negotiating secure connections using a user-supplied password, while eliminating the security problems traditionally associated with reusable passwords. This system also performs a secure key exchange in the process of authentication, allowing security layers (privacy and/or integrity protection) to be enabled during the session. Trusted key servers and certificate infrastructures are not required, and clients are not required to store or manage any long-term keys. SRP offers both security and deployment advantages over existing challenge-response techniques, making it an ideal drop-in replacement where secure password authentication is needed.

The complete RFC2945 specification can be obtained from http://www.rfc-editor.org/rfc.html. Additional information on the SRP algorithm and its history can be found at http://srp.stanford.edu/.

Algorithms like Diffie-Hellman and RSA are known as public key exchange algorithms. The concept of public key algorithms is that you have two keys, one public that is available to everyone, and one that is private and known only to you. When someone wants to send encrypted information to you, they encrypt the information using your public key. Only you are able to decrypt the information using your private key. Contrast this with the more traditional shared password based encryption schemes that require the sender and receiver to know the shared password. Public key algorithms eliminate the need to share passwords.

12.5.2. Configure Secure Remote Password (SRP) Protocol
To use Secure Remote Password (SRP) Protocol in your application, you first create an MBean which implements the `SRPVerifierStore` interface. Information about the implementation is provided in The SRPVerifierStore Implementation.

**Procedure 12.5. Integrate the Existing Password Store**

1. **Create the hashed password information store.**
   - If your passwords are already stored in an irreversible hashed form, you need to do this on a per-user basis.
   
   You can implement `setUserVerifier(String, VerifierInfo)` as a noOp method, or a method that throws an exception stating that the store is read-only.

2. **Create the SRPVerifierStore interface.**
   - Create a custom `SRPVerifierStore` interface implementation that can obtain the `VerifierInfo` from the store you created.
   
   The `verifyUserChallenge(String, Object)` can be used to integrate existing hardware token based schemes like SafeWord or Radius into the SRP algorithm. This interface method is called only when the client SPRLookinModule configuration specifies the hasAuxChallenge option.

3. **Create the JNDI MBean.**
   - Create a MBean that exposes the `SRPVerifierStore` interface available to JNDI, and exposes any configurable parameters required.
   
   The default `org.jboss.security.srp.SRPVerifierStoreService` allows you to implement this. You can also implement the MBean using a Java properties file implementation of `SRPVerifierStore`.

**The SRPVerifierStore Implementation**

The default implementation of the `SRPVerifierStore` interface is not recommended for production systems, because it requires all password hash information to be available as a file of serialized objects.

The `SRPVerifierStore` implementation provides access to the `SRPVerifierStore.VerifierInfo` object for a given username. The `getUserVerifier(String)` method is called by the SRPService at the start of a user SRP session to obtain the parameters needed by the SRP algorithm.

**Elements of a VerifierInfo Object**

**username**
- The username or user ID used to authenticate

**verifier**
- A one-way hash of the password the user enters as proof of identity. The `org.jboss.security.Util` class includes a `calculateVerifier` method which performs the password hashing algorithm. The output password takes the form `H(salt | H(username | ':') | password)`, where `H` is the SHA secure hash function as defined by RFC2945. The username is converted from a string to a byte[] using UTF-8 encoding.

**salt**
A random number used to increase the difficulty of a brute force dictionary attack on the verifier password database in the event that the database is compromised. The value should be generated from a cryptographically strong random number algorithm when the user’s existing clear-text password is hashed.

\textbf{g}

The SRP algorithm primitive generator. This can be a well known fixed parameter rather than a per-user setting. The \texttt{org.jboss.security.srp.SRPConf} utility class provides several settings for \texttt{g}, including a suitable default obtained via \texttt{SRPConf.getDefaultParams().g()}.

\textbf{N}

The SRP algorithm safe-prime modulus. This can be a well-known fixed parameter rather than a per-user setting. The \texttt{org.jboss.security.srp.SRPConf} utility class provides several settings for \texttt{N} including a good default obtained via \texttt{SRPConf.getDefaultParams().N()}.

\textbf{Example 12.13. The SRPVeriferStore Interface}

```
package org.jboss.security.srp;

import java.io.IOException;
import java.io.Serializable;
import java.security.KeyException;

public interface SRPVeriferStore
{
    public static class VerifierInfo implements Serializable
    {
        public String username;

        public byte[] salt;
        public byte[] g;
        public byte[] N;
    }

    public VerifierInfo getUserVerifier(String username)
        throws KeyException, IOException;

    public void setUserVerifier(String username, VerifierInfo info)
        throws IOException;

    public void verifyUserChallenge(String username, Object auxChallenge)
        throws SecurityException;
}
```

Report a bug
CHAPTER 13. SINGLE SIGN ON (SSO)

13.1. 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).
13.2. 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.

13.3. 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. Depending on your organization's needs, a few different SSO solutions are available. The solution you use depends on whether you use web applications, EJB applications, or web services; whether your applications run on the same server, multiple non-clustered servers, or multiple clustered servers; and whether you need to integrate into a desktop-based authentication system or you only need authentication between your applications themselves.

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 in JBoss EAP 6.

Non-Clustered and Web Application SSO

If you need to propagate security information among applications which run within the same server group or instance, you can use non-clustered SSO. This only involves configuring the valve in your application's `jboss-web.xml` descriptor.

Clustered Web Application SSO

If you need to propagate security information among applications running in a clustered environment across multiple JBoss EAP 6 instances, you can use the clustered SSO valve. This is configured in your application's `jboss-web.xml`. 
13.4. 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=full` portion of the command, for a standalone server.

  Example 13.1. 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 13.2. 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 13.3. 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 13.4. 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

Clustered SSO allows sharing of authentication between separate hosts, while non-clustered SSO does not. The clustered and non-clustered SSO valves are configured the same way, but the clustered SSO includes the **cacheConfig**, **processExpiresInterval** and **maxEmptyLife** parameters, which control the clustering replication of the persisted data.

Example 13.5. Example Clustered SSO Configuration

Because clustered and non-clustered SSO configurations are so similar, only a clustered configuration is shown. This example uses a security domain called **tomcat**.

```
<jboss-web>
  <security-domain>tomcat</security-domain>
  <valve>
    <class-name>org.jboss.web.tomcat.service.sso.ClusteredSingleSignOn</class-name>
    <param>
      <param-name>maxEmptyLife</param-name>
      <param-value>900</param-value>
    </param>
  </valve>
</jboss-web>
```
Table 13.1. SSO Configuration Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookieDomain</td>
<td>The host domain to be used for SSO cookies. The default is / . To allow app1.xyz.com and app2.xyz.com to share SSO cookies, you could set the cookieDomain to xyz.com.</td>
</tr>
<tr>
<td>maxEmptyLife</td>
<td>Clustered SSO only. The maximum number of seconds an SSO valve with no active sessions will be usable by a request, before expiring. A positive value allows proper handling of shutdown of a node if it is the only one with active sessions attached to the valve. If maxEmptyLife is set to 0, the valve terminates at the same time as the local session copies, but backup copies of the sessions, from clustered applications, are available to other cluster nodes. Allowing the valve to live beyond the life of its managed sessions gives the user time to make another request which can then fail over to a different node, where it activates the backup copy of the session. Defaults to 1800 seconds (30 minutes).</td>
</tr>
<tr>
<td>processExpiresInterval</td>
<td>Clustered SSO only. The minimum number of seconds between efforts by the valve to find and invalidate SSO instances which have expired the MaxEmptyLife timeout. Defaults to 60 (1 minute).</td>
</tr>
<tr>
<td>requiresReauthentication</td>
<td>If true, each request uses cached credentials to reauthenticate to the security realm. If false (the default), a valid SSO cookie is sufficient for the valve to authenticate each new request.</td>
</tr>
</tbody>
</table>

Invalidate a Session

An application can programmatically invalidate a session by invoking method javax.servlet.http.HttpSession.invalidate().

Report a bug

13.5. 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

13.6. 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 protected 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

13.7. 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

13.8. 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 Negotiation 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>
<thead>
<tr>
<th>Simple Name</th>
<th>Class Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kerberos</td>
<td>com.sun.security.auth.module.Krb5LoginModule</td>
<td>Kerberos login module</td>
</tr>
<tr>
<td>SPNEGO</td>
<td>org.jboss.security.negotiation.spnego.SPNEGOLoginModule</td>
<td>The mechanism which enables your Web applications to authenticate to your Kerberos authentication server.</td>
</tr>
<tr>
<td>AdvancedLdap</td>
<td>org.jboss.security.negotiation.AdvancedLdapLoginModule</td>
<td>Used with LDAP servers other than Microsoft Active Directory.</td>
</tr>
<tr>
<td>AdvancedAdLdap</td>
<td>org.jboss.security.negotiation.AdvancedADLoginModule</td>
<td>Used with Microsoft Active Directory LDAP servers.</td>
</tr>
</tbody>
</table>

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 13.1. 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>
    <!-- Search for roles -->
    <login-module code="UserRoles" flag="required">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="serverSecurityDomain" value="host"/>
    </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="UserRoles" 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.
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.

```xml
<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>

<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.

```xml
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 14. ROLE-BASED SECURITY IN APPLICATIONS

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 16.1, “About Authentication”
- Section 16.2, “About Authorization”
- Section 14.2, “About Security Auditing”
- Section 2.4, “About Declarative Security”
- Section 12.3.2.1, “About EJB Method Permissions”
- Section 12.3.3.1, “About EJB Security Annotations”

See also Section 14.6, “Use a Security Domain in Your Application”.

14.2. 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.

14.3. 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
14.4. ABOUT THE SECURITY EXTENSION ARCHITECTURE

The architecture of the JBoss EAP 6 security extensions consists of three parts. These three parts connect your application to your underlying security infrastructure, whether it is LDAP, Kerberos, or another external system.

JAAS

The first part of the infrastructure is the JAAS API. JAAS is a pluggable framework which provides a layer of abstraction between your security infrastructure and your application.

The main implementation in JAAS is `org.jboss.security.plugins.JaasSecurityManager`, which implements the `AuthenticationManager` and `RealmMapping` interfaces. `JaasSecurityManager` integrates into the EJB and web container layers, based on the `<security-domain>` element of the corresponding component deployment descriptor.

For more information about JAAS, refer to Section 16.3, “Java Authentication and Authorization Service (JAAS)

The `JaasSecurityManagerService` MBean

The `JaasSecurityManagerService` MBean service manages security managers. Although its name begins with Jaas, the security managers it handles need not use JAAS in their implementation. The name reflects the fact that the default security manager implementation is the `JaasSecurityManager`.

The primary role of the `JaasSecurityManagerService` is to externalize the security manager implementation. You can change the security manager implementation by providing an alternate implementation of the `AuthenticationManager` and `RealmMapping` interfaces.

The second fundamental role of the `JaasSecurityManagerService` is to provide a JNDI `javax.naming.spi.ObjectFactory` implementation to allow for simple code-free management of the binding between the JNDI name and the security manager implementation. To enable security, specify the JNDI name of the security manager implementation via the `<security-domain>` deployment descriptor element.

When you specify a JNDI name, an object-binding needs to already exist. To simplify the setup of the binding between the JNDI name and security manager, the `JaasSecurityManagerService` binds a next naming system reference, nominating itself as the JNDI `ObjectFactory` under the name `java:/jaas`. This permits a naming convention of the form `java:/jaas/XYZ` as the value for the `<security-domain>` element, and the security manager instance for the `XYZ` security domain is created as needed, by creating an instance of the class specified by the `SecurityManagerClassName` attribute, using a constructor that takes the name of the security domain.

NOTE

You do not need to include the `java:/jaas` prefix in your deployment descriptor. You may do so, for backward compatibility, but it is ignored.

The `JaasSecurityDomain` MBean

The `org.jboss.security.plugins.JaasSecurityDomain` is an extension of `JaasSecurityManager` which adds the notion of a `KeyStore`, a `KeyManagerFactory`, and a `TrustManagerFactory` for supporting SSL and other cryptographic use cases.
Further information
For more information, and practical examples of the security architecture in action, refer to Section 14.5, “About Java Authentication and Authorization Service (JAAS).”

Report a bug

14.5. ABOUT JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)

The security architecture of JBoss EAP 6 is comprised of the security configuration subsystem, application-specific security configurations which are included in several configuration files within the application, and the JAAS Security Manager, which is implemented as an MBean.

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 14.1. Application-Specific Configuration Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ejb-jar.xml</td>
<td>The deployment descriptor for an Enterprise JavaBean (EJB) application, located in the META-INF directory of the EJB. 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.</td>
</tr>
<tr>
<td>web.xml</td>
<td>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.</td>
</tr>
<tr>
<td>jboss-ejb3.xml</td>
<td>Contains JBoss-specific extensions to the ejb-jar.xml descriptor.</td>
</tr>
<tr>
<td>jboss-web.xml</td>
<td>Contains JBoss-specific extensions to the web.xml descriptor.</td>
</tr>
</tbody>
</table>
NOTE


The JAAS Security Manager MBean

The Java Authentication and Authorization Service (JAAS) is a framework for user-level security in Java applications, using pluggable authentication modules (PAM). It is integrated into the Java Runtime Environment (JRE). In JBoss EAP 6, the container-side component is the `org.jboss.security.plugins.JaasSecurityManager` MBean. It provides the default implementations of the `AuthenticationManager` and `RealmMapping` interfaces.

The JaasSecurityManager MBean integrates into the EJB and web container layers based on the security domain specified in the EJB or web deployment descriptor files in the application. When an application deploys, the container associates the security domain specified in the deployment descriptor with the security manager instance of the container. The security manager enforces the configuration of the security domain as configured on the server group or standalone server.

Flow of Interaction between the Client and the Container with JAAS

The JaasSecurityManager uses the JAAS packages to implement the `AuthenticationManager` and `RealmMapping` interface behavior. In particular, its behavior derives from the execution of the login module instances that are configured in the security domain to which the JaasSecurityManager has been assigned. The login modules implement the security domain's principal authentication and role-mapping behavior. You can use the JaasSecurityManager across different security domains by plugging in different login module configurations for the domains.

To illustrate how the JaasSecurityManager uses the JAAS authentication process, the following steps outline a client invocation of method which implements method `EJBHome`. The EJB has already been deployed in the server and its `EJBHome` interface methods have been secured using `<method-permission>` elements in the `ejb-jar.xml` descriptor. It uses the `jwdomain` security domain, which is specified in the `<security-domain>` element of the `jboss-ejb3.xml` file. The image below shows the steps, which are explained afterward.
Figure 14.1. Steps of a Secured EJB Method Invocation

1. The client performs a JAAS login to establish the principal and credentials for authentication. This is labeled **Client Side Login** in the figure. This could also be performed via JNDI.

To perform a JAAS login, you create a LoginContext instance and pass in the name of the configuration to use. Here, the configuration name is **other**. This one-time login associates the login principal and credentials with all subsequent EJB method invocations. The process does not necessarily authenticate the user. The nature of the client-side login depends on the login
module configuration that the client uses. In this example, the other client-side login configuration entry uses the ClientLoginModule login module. This module binds the user name and password to the EJB invocation layer for later authentication on the server. The identity of the client is not authenticated on the client.

2. The client obtains the EJBHome method and invokes it on the server. The invocation includes the method arguments passed by the client, along with the user identity and credentials from the client-side JAAS login.

3. On the server, the security interceptor authenticates the user who invoked the method. This involves another JAAS login.

4. The security domain under determines the choice of login modules. The name of the security domain is passed to the LoginContext constructor as the login configuration entry name. The EJB security domain is jwdomain. If the JAAS authentication is successful, a JAAS Subject is created. A JAAS subject includes a PrincipalSet, which includes the following details:

   - A java.security.Principal instance that corresponds to the client identity from the deployment security environment.
   - A java.security.acl.Group called Roles, which contains the role names from the user's application domain. Objects of type org.jboss.security.SimplePrincipal objects represent the role names. These roles validate access to EJB methods according to constraints in ejb-jar.xml and the EJBContext.isCallerInRole(String) method implementation.
   - An optional java.security.acl.Group named CallerPrincipal, which contains a single org.jboss.security.SimplePrincipal that corresponds to the identity of the application domain's caller. The CallerPrincipal group member is the value returned by the EJBContext.getCallerPrincipal() method. This mapping allows a Principal in the operational security environment to map to a Principal known to the application. In the absence of a CallerPrincipal mapping, the operational principal is the same as the application domain principal.

5. The server verifies that the user calling the EJB method has the permission to do so. Performing this authorization involves the following steps:

   - Obtain the names of the roles allowed to access the EJB method from the EJB container. The role names are determined by ejb-jar.xml descriptor <role-name> elements of all <method-permission> elements containing the invoked method.

   - If no roles have been assigned, or the method is specified in an exclude-list element, access to the method is denied. Otherwise, the doesUserHaveRole method is invoked on the security manager by the security interceptor to check if the caller has one of the assigned role names. This method iterates through the role names and checks if the authenticated user's Subject Roles group contains a SimplePrincipal with the assigned role name. Access is allowed if any role name is a member of the Roles group. Access is denied if none of the role names are members.

   - If the EJB uses a custom security proxy, the method invocation is delegated to the proxy. If the security proxy denies access to the caller, it throws a java.lang.SecurityException. Otherwise, access to the EJB method is allowed and the method invocation passes to the next container interceptor. The SecurityProxyInterceptor handles this check and this interceptor is not shown.
For web connection requests, the web server checks the security constraints defined in web.xml that match the requested resource and the accessed HTTP method.

If a constraint exists for the request, the web server calls the JaaSSecurityManager to perform the principal authentication, which in turn ensures the user roles are associated with that principal object.

Report a bug

14.6. USE A SECURITY DOMAIN IN YOUR APPLICATION

Overview

To use a security domain in your application, first you must configure the domain in either the server's configuration file or the application's descriptor file. 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.

Procedure 14.1. Configure Your Application to Use a Security Domain

1. Define the Security Domain
   You can define the security domain either in the server's configuration file or the application's descriptor file.

   o 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.

```xml
<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>
```
configuration additional security domains as needed using the Management Console or CLI.

Configure 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`.

```xml
<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.

```java
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

14.7. 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 14.2. 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>
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**.

```xml
<jboss-web>
  ...
  <security-domain>acme_domain</security-domain>
  ...
</jboss-web>
```

**Example 14.1. Example web.xml with Role-Based Security Configured**

```xml
<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>
```

CHAPTER 14. ROLE-BASED SECURITY IN APPLICATIONS
14.8. 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 14.2. Basic Authentication Valve**

```xml
<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 14.3. Custom Valve With Header Attributes Set**

```xml
<jboss-web>
  <valve>
    <class-name>
```
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 14.4. GenericHeaderAuthenticator.java

```java
package org.jboss.web.tomcat.security;

import java.io.IOException;
import java.security.Principal;
import java.util.StringTokenizer;

public class GenericHeaderAuthenticator

    <param name="httpHeaderForSSOAuth">
        <param-name>sm_ssoid,ct-remote-user,HTTP_OBLIX_UID</param-name>
        <param-value>
    </param>

    <param name="sessionCookieForSSOAuth">
        <param-name>SMSESSION,CTSESSION,ObSSOCookie</param-name>
        <param-value>
    </param>

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 14.4. GenericHeaderAuthenticator.java

```
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;

    /**
     * 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.
     * @return a <code>String</code> containing the value of the
     * <code>httpHeaderForSSOAuth</code> attribute.
public String getHttpHeaderForSSOAuth() {
    return httpHeaderForSSOAuth;
}

/**
 * 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;
}

public String getSessionCookieForSSOAuth() {
    return sessionCookieForSSOAuth;
}

/**
 * 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;
}
public void setSessionCookieForSSOAuth(String sessionCookieForSSOAuth) {
    this.sessionCookieForSSOAuth = sessionCookieForSSOAuth;
}

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);
    if (numCookies = cookies != null ? cookies.length : 0;
    // We can have comma-separated ids
    String ids = ""
    try {
        ids = this.getSessionCookieId();
    } 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("Session Cookie not found");
            return null;
        } 
    }
    return null;
}
log.trace("Cookie-" + token + " value=" + cookie.getValue());
return cookie.getValue();
}

return null;
15.1. CONFIGURE APPLICATION SECURITY CHANGES

Configure security for basic authentication

In previous versions of JBoss EAP, properties files placed in the 
\texttt{EAP\_HOME/server/\texttt{SERVER\_NAME}/conf/} directory were on classpath and could be easily found by the \texttt{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.

\textbf{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 \texttt{security-domains} to the \texttt{standalone/configuration/standalone.xml} or the \texttt{domain/configuration/domain.xml} server configuration file:

\begin{verbatim}
<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>
\end{verbatim}

If the JBoss EAP 6 instance is running as a standalone server, \texttt{"${jboss.server.config.dir}"} refers to the \texttt{EAP\_HOME/standalone/configuration/} directory. If the instance is running in a managed domain, \texttt{"${jboss.server.config.dir}"} refers to the \texttt{EAP\_HOME/domain/configuration/} directory.

Modify security domain names

In JBoss EAP 6, security domains no longer use the prefix \texttt{java:/jaas/} in their names.

- For Web applications, you must remove this prefix from the security domain configurations in the \texttt{jboss-web.xml}.

- For Enterprise applications, you must remove this prefix from the security domain configurations in the \texttt{jboss-ejb3.xml} file. This file has replaced the \texttt{jboss.xml} in JBoss EAP 6.

Report a bug
CHAPTER 16. AUTHENTICATION AND AUTHORIZATION

16.1. 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

16.2. 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.

Report a bug

16.3. 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 2.4, “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

16.4. ABOUT JAVA AUTHENTICATION AND AUTHORIZATION SERVICE (JAAS)
The security architecture of JBoss EAP 6 is comprised of the security configuration subsystem, application-specific security configurations which are included in several configuration files within the application, and the JAAS Security Manager, which is implemented as an MBean.

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 16.1. Application-Specific Configuration Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ejb-jar.xml</td>
<td>The deployment descriptor for an Enterprise JavaBean (EJB) application, located in the <code>META-INF</code> directory of the EJB. Use the <code>ejb-jar.xml</code> 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.</td>
</tr>
<tr>
<td>web.xml</td>
<td>The deployment descriptor for a Java Enterprise Edition (EE) web application. Use the <code>web.xml</code> 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.</td>
</tr>
<tr>
<td>jboss-ejb3.xml</td>
<td>Contains JBoss-specific extensions to the <code>ejb-jar.xml</code> descriptor.</td>
</tr>
<tr>
<td>jboss-web.xml</td>
<td>Contains JBoss-specific extensions to the <code>web.xml</code> descriptor.</td>
</tr>
</tbody>
</table>

NOTE


The JAAS Security Manager MBean

The Java Authentication and Authorization Service (JAAS) is a framework for user-level security in Java applications, using pluggable authentication modules (PAM). It is integrated into the Java Runtime Environment (JRE). In JBoss EAP 6, the container-side component is the

The JaasSecurityManager MBean integrates into the EJB and web container layers based on the security domain specified in the EJB or web deployment descriptor files in the application. When an application deploys, the container associates the security domain specified in the deployment descriptor with the security manager instance of the container. The security manager enforces the configuration of the security domain as configured on the server group or standalone server.

Flow of Interaction between the Client and the Container with JAAS

The JaasSecurityManager uses the JAAS packages to implement the AuthenticationManager and RealmMapping interface behavior. In particular, its behavior derives from the execution of the login module instances that are configured in the security domain to which the JaasSecurityManager has been assigned. The login modules implement the security domain's principal authentication and role-mapping behavior. You can use the JaasSecurityManager across different security domains by plugging in different login module configurations for the domains.

To illustrate how the JaasSecurityManager uses the JAAS authentication process, the following steps outline a client invocation of method which implements method EJBHome. The EJB has already been deployed in the server and its EJBHome interface methods have been secured using <method-permission> elements in the ejb-jar.xml descriptor. It uses the jwdomain security domain, which is specified in the <security-domain> element of the jboss-ejb3.xml file. The image below shows the steps, which are explained afterward.
Figure 16.1. Steps of a Secured EJB Method Invocation

1. The client performs a JAAS login to establish the principal and credentials for authentication. This is labeled **Client Side Login** in the figure. This could also be performed via JNDI.

To perform a JAAS login, you create a LoginContext instance and pass in the name of the configuration to use. Here, the configuration name is `other`. This one-time login associates the login principal and credentials with all subsequent EJB method invocations. The process does not necessarily authenticate the user. The nature of the client-side login depends on the login
module configuration that the client uses. In this example, the other client-side login configuration entry uses the ClientLoginModule login module. This module binds the user name and password to the EJB invocation layer for later authentication on the server. The identity of the client is not authenticated on the client.

2. The client obtains the EJBHome method and invokes it on the server. The invocation includes the method arguments passed by the client, along with the user identity and credentials from the client-side JAAS login.

3. On the server, the security interceptor authenticates the user who invoked the method. This involves another JAAS login.

4. The security domain under determines the choice of login modules. The name of the security domain is passed to the LoginContext constructor as the login configuration entry name. The EJB security domain is jwdomain. If the JAAS authentication is successful, a JAAS Subject is created. A JAAS subject includes a PrincipalSet, which includes the following details:

   - A java.security.Principal instance that corresponds to the client identity from the deployment security environment.
   - A java.security.acl.Group called Roles, which contains the role names from the user's application domain. Objects of type org.jboss.security.SimplePrincipal objects represent the role names. These roles validate access to EJB methods according to constraints in ejb-jar.xml and the EJBCallContext.isCallerInRole(String) method implementation.
   - An optional java.security.acl.Group named CallerPrincipal, which contains a single org.jboss.security.SimplePrincipal that corresponds to the identity of the application domain's caller. The CallerPrincipal group member is the value returned by the EJBCallContext.getCallerPrincipal() method. This mapping allows a Principal in the operational security environment to map to a Principal known to the application. In the absence of a CallerPrincipal mapping, the operational principal is the same as the application domain principal.

5. The server verifies that the user calling the EJB method has the permission to do so. Performing this authorization involves the following steps:

   - Obtain the names of the roles allowed to access the EJB method from the EJB container. The role names are determined by ejb-jar.xml descriptor <role-name> elements of all <method-permission> elements containing the invoked method.
   - If no roles have been assigned, or the method is specified in an exclude-list element, access to the method is denied. Otherwise, the doesUserHaveRole method is invoked on the security manager by the security interceptor to check if the caller has one of the assigned role names. This method iterates through the role names and checks if the authenticated user's Subject Roles group contains a SimplePrincipal with the assigned role name. Access is allowed if any role name is a member of the Roles group. Access is denied if none of the role names are members.
   - If the EJB uses a custom security proxy, the method invocation is delegated to the proxy. If the security proxy denies access to the caller, it throws a java.lang.SecurityException. Otherwise, access to the EJB method is allowed and the method invocation passes to the next container interceptor. The SecurityProxyInterceptor handles this check and this interceptor is not shown.
For web connection requests, the web server checks the security constraints defined in `web.xml` that match the requested resource and the accessed HTTP method.

If a constraint exists for the request, the web server calls the JaasSecurityManager to perform the principal authentication, which in turn ensures the user roles are associated with that principal object.

**16.5. JAVA AUTHORIZATION CONTRACT FOR CONTAINERS (JACC)**

**16.5.1. 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](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.

**16.5.2. 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"/>
  </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>
```
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 16.1. 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 16.2. 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

16.6. JAVA AUTHENTICATION SPI FOR CONTAINERS (JASPI)

16.6.1. 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

16.6.2. 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 16.3. 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
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, "Client"
- Table A.3, "Certificate"
- Table A.5, "CertificateUsers"
- Table A.7, "CertificateRoles"
- Table A.9, "Database"
- Table A.11, "DatabaseCertificate"
- Table A.13, "Identity"
- Table A.15, "Ldap"
- Table A.17, "LdapExtended"
- Table A.19, "RoleMapping"
- Table A.21, "RunAs"
- Table A.23, "Simple"
- Table A.24, "ConfiguredIdentity"
- Table A.26, "SecureIdentity"
- Table A.28, "PropertiesUsers"
- Table A.30, "SimpleUsers"
- Table A.32, "LdapUsers"
- Table A.33, "Kerberos"
- Table A.35, "SPNEGOUsers"
- Table A.37, "AdvancedLdap"
- Table A.39, "AdvancedADLdap"
- Table A.40, "UsersRoles"
### Table A.1. Client

<table>
<thead>
<tr>
<th>Code</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.ClientLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>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 actual server authentication.</td>
</tr>
</tbody>
</table>

### Table A.2. Client Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multi-threaded</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>password-stacking</td>
<td>useFirstPass or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>restore-login-identity</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table A.3. Certificate

<table>
<thead>
<tr>
<th>Code</th>
<th>Certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.BaseCertLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table A.4. Certificate Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>securityDomain</td>
<td>string</td>
<td>none</td>
<td>Name of the security domain that has the JSSE configuration for the truststore holding the trusted certificates.</td>
</tr>
<tr>
<td>verifier</td>
<td>Class</td>
<td>none</td>
<td>The class name of the org.jboss.security.auth.certs.X509CertificateVerifier to use for verification of the login certificate.</td>
</tr>
</tbody>
</table>

### Table A.5. CertificateUsers

<table>
<thead>
<tr>
<th>Code</th>
<th>CertificateUsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.UsersRolesLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>Uses a properties resource. The first maps usernames to passwords, and the second maps usernames to roles.</td>
</tr>
</tbody>
</table>

### Table A.6. CertificateUsers Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unauthenticatedIdentity</td>
<td>A string</td>
<td>none</td>
<td>Defines the principal name that should be 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 either unsecured EJBs or EJB methods that are associated with the unchecked permission constraint.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>password-stacking</td>
<td>useFirstPass or</td>
<td>false</td>
<td>Set to <code>useFirstPass</code> to indicate that this login module should look for information stored in the <code>LoginContext</code> to use as the identity. This option can be used when stacking other login modules with this one.</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hashAlgorithm</td>
<td>A string</td>
<td>none</td>
<td>The name of the <code>java.security.MessageDigest</code> algorithm to use to hash the password. There is no default so this option must be explicitly set to enable hashing. When <code>hashAlgorithm</code> is specified, the clear text password obtained from the <code>CallbackHandler</code> is hashed before it is passed to <code>UsernamePasswordLoginModule.validatePassword</code> as the <code>inputPassword</code> argument. The <code>expectedPassword</code> stored in the <code>users.properties</code> file must be comparably hashed. Refer to <a href="http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html">http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html</a> for information on <code>java.security.MessageDigest</code> and the algorithms this class supports.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hashEncoding</td>
<td>base64 or hex</td>
<td>base64</td>
<td>The string format for the hashed password, if <code>hashAlgorithm</code> is also set.</td>
</tr>
<tr>
<td>hashCharset</td>
<td>A string</td>
<td></td>
<td>The default encoding set in the container's environment.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The encoding used to convert the clear-text password to a byte array.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------</td>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>usersProperties</td>
<td>The fully-qualified file path and name of a properties file or resource</td>
<td>users.properties</td>
<td>The file containing the mappings between users and passwords. Each property in the file has the format <code>username=password</code>.</td>
</tr>
<tr>
<td>rolesProperties</td>
<td>The fully-qualified file path and name of a properties file or resource</td>
<td>roles.properties</td>
<td>The file containing the mappings between users and roles. Each property in the file has the format <code>username=role1,role2,...,roleN</code>.</td>
</tr>
<tr>
<td>ignorePasswordCase</td>
<td>true or false</td>
<td>false</td>
<td>Whether the password comparison should ignore case. This is useful for hashed password encoding where the case of the hashed password is not significant.</td>
</tr>
<tr>
<td>principalClass</td>
<td>A fully-qualified classname.</td>
<td>none</td>
<td>A <code>Principal</code> implementation class which contains a constructor that takes a String argument for the principal name.</td>
</tr>
<tr>
<td>roleGroupSeparator</td>
<td>A single character</td>
<td>. (a single period)</td>
<td>The character used to separate the username from the role group name in the <code>rolesGroup</code> file.</td>
</tr>
<tr>
<td>defaultUsersProperties</td>
<td>string</td>
<td>defaultUsers.properties</td>
<td>Name of the resource or file to fall back to if the <code>usersProperties</code> file can't be found.</td>
</tr>
<tr>
<td>defaultRolesProperties</td>
<td>string</td>
<td>defaultRoles.properties</td>
<td>Name of the resource or file to fall back to if the <code>rolesProperties</code> file cannot be found.</td>
</tr>
<tr>
<td>hashUserPassword</td>
<td>true or false</td>
<td>true</td>
<td>Whether to hash the password entered by the user, when <code>hashAlgorithm</code> is specified. Defaults to <code>true</code>.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hashStorePassword</td>
<td>true or false</td>
<td>true</td>
<td>Whether the store password returned from <code>getUsersPassword()</code> should be hashed, when <code>hashAlgorithm</code> is specified.</td>
</tr>
<tr>
<td>digestCallback</td>
<td>A fully-qualified classname.</td>
<td>none</td>
<td>The class name of the <code>org.jboss.crypto.digest.DigestCallback</code> implementation that includes pre or post digest content such as salt values. Only used if <code>hashAlgorithm</code> is specified.</td>
</tr>
<tr>
<td>storeDigestCallback</td>
<td>A fully-qualified classname.</td>
<td>none</td>
<td>The class name of the <code>org.jboss.crypto.digest.DigestCallback</code> implementation that includes pre/post digest content like salts for hashing the store password. Only used if <code>hashStorePassword</code> is true and <code>hashAlgorithm</code> is specified.</td>
</tr>
<tr>
<td>callback.option.STRING</td>
<td>Various</td>
<td>none</td>
<td>All options prefixed with <code>callback.option.</code> are passed to the <code>DigestCallback.init(Map)</code> method. The input username is always passed in via the <code>javax.security.auth.login.name</code> option, and the input/store password is passed in via the <code>javax.security.auth.login.password</code> option to the <code>digestCallback</code> or <code>storeDigestCallback</code>.</td>
</tr>
</tbody>
</table>

Table A.7. CertificateRoles

<table>
<thead>
<tr>
<th>Code</th>
<th>CertificateRoles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>rolesProperties</td>
<td>A string</td>
</tr>
<tr>
<td>defaultRolesProperties</td>
<td>A string</td>
</tr>
<tr>
<td>roleGroupSeparator</td>
<td>A single character</td>
</tr>
</tbody>
</table>

Table A.9. Database

<table>
<thead>
<tr>
<th>Code</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.DatabaseServerLoginModule</td>
</tr>
</tbody>
</table>
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.10. Database Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsJndiName</td>
<td>A JNDI resource</td>
<td>none</td>
<td>The name of the JNDI resource storing the authentication information. This option is required.</td>
</tr>
<tr>
<td>principalsQuery</td>
<td>A prepared SQL statement</td>
<td>select Password from Principals where PrincipalID=?</td>
<td>The prepared SQL query to obtain the information about the principal.</td>
</tr>
<tr>
<td>rolesQuery</td>
<td>A prepared SQL statement</td>
<td>select Role, RoleGroup from Roles where PrincipalID=?</td>
<td>The prepared SQL query to obtain the information about the roles. It should be equivalent to <code>select Role, RoleGroup from Roles where PrincipalID=?</code>, where Role is the role name and the RoleGroup column value should always be either Roles with a capital R or CallerPrincipal.</td>
</tr>
</tbody>
</table>

### Table A.11. DatabaseCertificate

<table>
<thead>
<tr>
<th>Code</th>
<th>DatabaseCertificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.DatabaseCertLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>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:</td>
</tr>
</tbody>
</table>
### Table A.12. DatabaseCertificate Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsJndiName</td>
<td>A JNDI resource</td>
<td></td>
<td>The name of the JNDI resource storing the authentication information. This option is required.</td>
</tr>
<tr>
<td>rolesQuery</td>
<td>A prepared SQL statement</td>
<td>select Role,RoleGroup from Roles where PrincipalID=?</td>
<td>SQL prepared statement to be executed in order to map 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.</td>
</tr>
<tr>
<td>suspendResume</td>
<td>true or false</td>
<td>true</td>
<td>Whether any existing JTA transaction should be suspended during database operations.</td>
</tr>
</tbody>
</table>

### Table A.13. Identity

<table>
<thead>
<tr>
<th>Code</th>
<th>Identity</th>
<th>org.jboss.security.auth.spi.Identity LoginModule</th>
<th>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.</th>
</tr>
</thead>
</table>

### Table A.14. Identity Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>A string</td>
<td>guest</td>
<td>The name to use for the principal.</td>
</tr>
<tr>
<td>roles</td>
<td>A comma-separated list of strings</td>
<td>none</td>
<td>A comma-delimited list of roles which will be assigned to the subject.</td>
</tr>
</tbody>
</table>
### Table A.15. Ldap

<table>
<thead>
<tr>
<th>Code</th>
<th>Ldap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.LdapLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Table A.16. Ldap Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.naming.factory.initial</td>
<td>class name</td>
<td>com.sun.jndi.ldap.p.LdapCtxFactory</td>
<td>InitialContextFactory implementation class name.</td>
</tr>
<tr>
<td>java.naming.provider.url</td>
<td>ldap:// URL</td>
<td>none</td>
<td>URL for the LDAP server.</td>
</tr>
<tr>
<td>java.naming.security.authentication</td>
<td>none, simple, or the name of a SASL mechanism</td>
<td>simple</td>
<td>The security level to use to bind to the LDAP server.</td>
</tr>
<tr>
<td>java.naming.security.protocol</td>
<td>A transport protocol</td>
<td>If unspecified, determined by the provider.</td>
<td>The transport protocol to use for secure access, such as SSL.</td>
</tr>
<tr>
<td>java.naming.security.principal</td>
<td>A string</td>
<td>none</td>
<td>The name of the principal for authenticating the caller to the service. This is built from other properties described below.</td>
</tr>
<tr>
<td>java.naming.security.credentials</td>
<td>A credential type</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>principalDNPrefix</td>
<td>A string</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>principalDNSuffix</td>
<td>string</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>useObjectCredential</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>rolesCtxDN</td>
<td>A fully-qualified DN</td>
<td>none</td>
<td>The fully-qualified DN for the context to search for user roles.</td>
</tr>
<tr>
<td>userRolesCtxDNAtributeName</td>
<td>An attribute</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>roleAttributeID</td>
<td>An attribute</td>
<td>roles</td>
<td>Name of the attribute containing the user roles.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>roleAttributeIsDN</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>roleNameAttributeNameID</td>
<td>An attribute</td>
<td>group</td>
<td>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.</td>
</tr>
<tr>
<td>uidAttributeID</td>
<td>An attribute</td>
<td>uid</td>
<td>Name of the attribute in the UserRolesAttributeDN that corresponds to the user ID. This is used to locate the user roles.</td>
</tr>
<tr>
<td>matchOnUserDN</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>allowEmptyPasswords</td>
<td>true or false</td>
<td>true</td>
<td>Whether to allow empty passwords. Most LDAP servers treat empty passwords as anonymous login attempts. To reject empty passwords, set this to false.</td>
</tr>
</tbody>
</table>
Table A.17. LdapExtended

<table>
<thead>
<tr>
<th>Code</th>
<th>LdapExtended</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.LdapExtLoginModule</td>
</tr>
</tbody>
</table>
| 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>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseCtxDN</td>
<td>A fully-qualified DN</td>
<td>none</td>
<td>The fixed DN of the top-level context to begin the user search.</td>
</tr>
<tr>
<td>bindDN</td>
<td>A fully-qualified DN</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>bindCredential</td>
<td>A string, optionally encrypted</td>
<td>none</td>
<td>The password for the bindDN. This can be encrypted if the jaasSecurityDomain is specified.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jaasSecurityDomain</td>
<td>A JMX ObjectName</td>
<td>none</td>
<td>The JMX <code>ObjectName</code> of the <code>JaasSecurityDomain</code> to use to decrypt the <code>bindCredential</code>. The encrypted form of the password is the format returned by the <code>JaasSecurityDomain.encrypt64(byte[])</code> method.</td>
</tr>
<tr>
<td>baseFilter</td>
<td>LDAP filter string</td>
<td>none</td>
<td>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 <code>{0}</code> expression is used. A common example for the search filter is <code>(uid={0})</code>.</td>
</tr>
<tr>
<td>rolesCtxDN</td>
<td>fully-qualified DN</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>roleFilter</td>
<td>LDAP filter string</td>
<td></td>
<td>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}).</td>
</tr>
<tr>
<td>roleAttributeIsDN</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>defaultRole</td>
<td>Role name</td>
<td>none</td>
<td>A role included for all authenticated users</td>
</tr>
<tr>
<td>parseRoleNameFromDN</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parseUsername</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>usernameBeginString</td>
<td>a string</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>usernameEndString</td>
<td>a string</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>roleNameAttributeID</td>
<td>An attribute</td>
<td>group</td>
<td>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.</td>
</tr>
<tr>
<td>distinguishedNameAttribute</td>
<td>An attribute</td>
<td>distinguishedName</td>
<td>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.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>roleRecursion</td>
<td>An integer</td>
<td>0</td>
<td>The numbers of levels of recursion the role search will go below a matching context. Disable recursion by setting this to 0.</td>
</tr>
<tr>
<td>searchTimeLimit</td>
<td>An integer</td>
<td>10000 (10 seconds)</td>
<td>The timeout in milliseconds for user or role searches.</td>
</tr>
<tr>
<td>searchScope</td>
<td>One of: OBJECT_SCOPE, ONELEVEL_SCOPE, SUBTREE_SCOPE</td>
<td>SUBTREE_SCOPE</td>
<td>The search scope to use.</td>
</tr>
<tr>
<td>allowEmptyPasswords</td>
<td>true or false</td>
<td>true</td>
<td>Whether to allow empty passwords. Most LDAP servers treat empty passwords as anonymous login attempts. To reject empty passwords, set this to false.</td>
</tr>
</tbody>
</table>

**Table A.19. RoleMapping**

<table>
<thead>
<tr>
<th>Code</th>
<th>RoleMapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.RoleMappingLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Table A.20. RoleMapping Module Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolesProperties</td>
<td>The fully-qualified file path and name of a properties file or resource</td>
<td>roles.properties</td>
<td>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</td>
</tr>
</tbody>
</table>
Whether to add to the current roles, or replace the current roles with the mapped ones. Replaces if set to true.

Table A.21. RunAs

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replaceRole</td>
<td>true or false</td>
<td>false</td>
<td>Whether to add to the current roles, or replace the current roles with the mapped ones. Replaces if set to true.</td>
</tr>
</tbody>
</table>

Table A.22. RunAs Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roleName</td>
<td>A role name.</td>
<td>nobody</td>
<td>The name of the role to use as the run as role during the login phase.</td>
</tr>
</tbody>
</table>

Table A.23. Simple

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Simple</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.SimpleServerLoginModule</td>
<td></td>
</tr>
</tbody>
</table>
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.24. ConfiguredIdentity

<table>
<thead>
<tr>
<th>Code</th>
<th>ConfiguredIdentity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td><code>org.picketbox.datasource.security.ConfiguredIdentityLoginModule</code></td>
</tr>
<tr>
<td>Description</td>
<td>Associates the principal specified in the module options with any subject authenticated against the module. The type of Principal class used is <code>org.jboss.security.SimplePrincipal</code>.</td>
</tr>
</tbody>
</table>

Table A.25. ConfiguredIdentity Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principal</td>
<td>Name of a principal.</td>
<td>none</td>
<td>The principal which will be associated with any subject authenticated against the module.</td>
</tr>
</tbody>
</table>

Table A.26. SecureIdentity

<table>
<thead>
<tr>
<th>Code</th>
<th>SecureIdentity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td><code>org.picketbox.datasource.security.SecureIdentityLoginModule</code></td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table A.27. SecureIdentity Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>string</td>
<td>none</td>
<td>The username for authentication.</td>
</tr>
<tr>
<td>password</td>
<td>encrypted string</td>
<td>none</td>
<td>The password to use for authentication. To encrypt the password, use the module directly at the command line.</td>
</tr>
</tbody>
</table>

```java
java org.picketbox.datasource.security.SecureIdentityLoginModule
password_to_encrypt
```

Paste the result of this command into the module option's value field.

| managedConnectionFactoryName | A JCA resource | none    | The name of the JCA connection factory for your datasource.                 |

### Table A.28. PropertiesUsers

<table>
<thead>
<tr>
<th>Code</th>
<th>PropertiesUsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.PropertiesUsersLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>Uses a properties file to store usernames and passwords for authentication. No authorization (role mapping) is provided. This module is only appropriate for testing.</td>
</tr>
</tbody>
</table>

### Table A.29. PropertiesUsers Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>The fully-qualified file path and name of a Java properties file or resource.</td>
<td>none</td>
<td>The properties file containing the usernames and cleartext passwords to be used for authentication.</td>
</tr>
</tbody>
</table>
### Table A.30. SimpleUsers

<table>
<thead>
<tr>
<th>Code</th>
<th>SimpleUsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.SimpleUsersLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>This login module stores the username and clear-text password in a Java properties file. It is included for testing only, and is not appropriate for a production environment.</td>
</tr>
</tbody>
</table>

### Table A.31. SimpleUsers Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>string</td>
<td>none</td>
<td>The username to use for authentication.</td>
</tr>
<tr>
<td>password</td>
<td>string</td>
<td>none</td>
<td>The clear-text password to use for authentication.</td>
</tr>
</tbody>
</table>

### Table A.32. LdapUsers

<table>
<thead>
<tr>
<th>Code</th>
<th>LdapUsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.LdapUsersLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>The LdapUsers module is superseded by the ExtendedLDAP and AdvancedLDAP modules.</td>
</tr>
</tbody>
</table>

### Table A.33. Kerberos

<table>
<thead>
<tr>
<th>Code</th>
<th>Kerberos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>com.sun.security.auth.module.Krb5LoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>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 <a href="http://docs.oracle.com/javase/1.4.2/docs/guide/security/jaas/spec/com/sun/security/auth/module/Krb5LoginModule.html">http://docs.oracle.com/javase/1.4.2/docs/guide/security/jaas/spec/com/sun/security/auth/module/Krb5LoginModule.html</a>. This module needs to be paired with another module which handles the authentication and roles mapping.</td>
</tr>
</tbody>
</table>

### Table A.34. Kerberos Module Options
<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storekey</td>
<td>true or false</td>
<td>false</td>
<td>Whether or not to add the KerberosKey to the subject's private credentials.</td>
</tr>
<tr>
<td>doNotPrompt</td>
<td>true or false</td>
<td>false</td>
<td>If set to true, the user is not prompted for the password.</td>
</tr>
<tr>
<td>useTicketCache</td>
<td>Boolean value of true or false</td>
<td>false</td>
<td>If true, the GTG is obtained from the ticket cache. If false, the ticket cache is not used.</td>
</tr>
<tr>
<td>ticketcache</td>
<td>A file or resource representing a Kerberos ticket cache.</td>
<td>The default depends on which operating system you use.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Red Hat Enterprise Linux / Solaris: /tmp/krb5cache_uid, using the numeric UID value of the operating system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Microsoft Windows Server: uses the Local Security Authority (LSA) API to find the ticket cache.</td>
</tr>
<tr>
<td>useKeyTab</td>
<td>true or false</td>
<td>false</td>
<td>Whether to obtain the principal's key from a key table file.</td>
</tr>
<tr>
<td>keytab</td>
<td>A file or resource representing a Kerberos keytab.</td>
<td>the location in the operating system's Kerberos configuration file, or /home/user/krb5.keytab</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The location of the key table file.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>principal</td>
<td>A string</td>
<td>none</td>
<td>The name of the principal. This can either be a simple user name or a service name such as <code>host/testserver.acme.com</code>. Use this instead of obtaining the principal from the key table, or when the key table contains more than one principal.</td>
</tr>
<tr>
<td>useFirstPass</td>
<td>true or false</td>
<td>false</td>
<td>Whether to retrieve the username and password from the module's shared state, using <code>javax.security.auth.login.name</code> and <code>javax.security.auth.login.password</code> as the keys. If authentication fails, no retry attempt is made.</td>
</tr>
<tr>
<td>tryFirstPass</td>
<td>true or false</td>
<td>false</td>
<td>Same as <code>useFirstPass</code>, but if authentication fails, the module uses the <code>CallbackHandler</code> to retrieve a new username and password. If the second authentication fails, the failure is reported to the calling application.</td>
</tr>
<tr>
<td>storePass</td>
<td>true or false</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>clearPass</td>
<td>true or false</td>
<td>false</td>
<td>Set this to <code>true</code> to clear the username and password from the shared state after both phases of authentication complete.</td>
</tr>
</tbody>
</table>

Table A.35. SPNEGOUsers
Table A.36. SPNEGO Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storeKey</td>
<td>true or false</td>
<td>false</td>
<td>Whether or not to store the key.</td>
</tr>
<tr>
<td>useKeyTab</td>
<td>true or false</td>
<td>false</td>
<td>Whether to use a key table.</td>
</tr>
<tr>
<td>principal</td>
<td>String representing a principal for Kerberos authentication.</td>
<td>none</td>
<td>The name of the principal for authentication.</td>
</tr>
<tr>
<td>keyTab</td>
<td>A file or resource representing a keytab.</td>
<td>none</td>
<td>The location of a key table.</td>
</tr>
<tr>
<td>doNotPrompt</td>
<td>true or false</td>
<td>false</td>
<td>Whether to prompt for a password.</td>
</tr>
<tr>
<td>debug</td>
<td>true or false</td>
<td>false</td>
<td>Whether to record more verbose messages for debugging purposes.</td>
</tr>
</tbody>
</table>

Table A.37. AdvancedLdap

<table>
<thead>
<tr>
<th>Code</th>
<th>AdvancedLdap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.negotiation.AdvancedLdapLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>A module which provides additional functionality, such as SASL and the use of a JAAS security domain.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>bindAuthentication</td>
<td>string</td>
</tr>
<tr>
<td>jassSecurityDomain</td>
<td>string</td>
</tr>
<tr>
<td>java.naming.provider.url</td>
<td>string</td>
</tr>
<tr>
<td>baseCtxDN</td>
<td></td>
</tr>
<tr>
<td>baseFilter</td>
<td></td>
</tr>
<tr>
<td>roleAttributeID</td>
<td></td>
</tr>
<tr>
<td>roleAttributeIsDN</td>
<td>true or false</td>
</tr>
<tr>
<td>roleNameAttributeID</td>
<td></td>
</tr>
<tr>
<td>recurseRoles</td>
<td>true or false</td>
</tr>
</tbody>
</table>

Table A.39. AdvancedADLdap

<table>
<thead>
<tr>
<th>Code</th>
<th>AdvancedADLdap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.negotiation.AdvancedADLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>This module extends the AdvancedLdap login module, and adds extra parameters that are relevant to Microsoft Active Directory.</td>
</tr>
</tbody>
</table>
### Table A.40. UsersRoles

<table>
<thead>
<tr>
<th>Code</th>
<th>UsersRoles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>org.jboss.security.auth.spi.UsersRolesLoginModule</td>
</tr>
<tr>
<td>Description</td>
<td>A simple login module that supports multiple users and user roles stored in two different properties files.</td>
</tr>
</tbody>
</table>

### Table A.41. UsersRoles Module Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>usersProperties</td>
<td>Path to a file or resource.</td>
<td>users.properties</td>
<td>The file or resource which contains the user-to-password mappings. The format of the file is <code>user=hashed-password</code></td>
</tr>
<tr>
<td>rolesProperties</td>
<td>Path to a file or resource.</td>
<td>roles.properties</td>
<td>The file or resource which contains the user-to-role mappings. The format of the file is <code>username=role1,role2,role3</code></td>
</tr>
<tr>
<td>passwordStacking</td>
<td>useFirstPass or false</td>
<td>false</td>
<td>A value of <code>useFirstPass</code> indicates that this login module should first look to the information stored in the <code>LoginContext</code> for the identity. This option can be used when stacking other login modules with this one.</td>
</tr>
<tr>
<td>Option</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hashAlgorithm</td>
<td>A string representing a password hashing algorithm.</td>
<td>none</td>
<td>The name of the <em>java.security.MessageDigest</em> algorithm to use to hash the password. There is no default so this option must be explicitly set to enable hashing. When <code>hashAlgorithm</code> is specified, the clear text password obtained from the <code>CallbackHandler</code> is hashed before it is passed to <code>UsernamePasswordLoginModule.validatePassword</code> as the <code>inputPassword</code> argument. The password stored in the <code>users.properties</code> file must be comparably hashed.</td>
</tr>
<tr>
<td>hashEncoding</td>
<td>base64 or hex</td>
<td>base64</td>
<td>The string format for the hashed password, if <code>hashAlgorithm</code> is also set.</td>
</tr>
<tr>
<td>hashCharset</td>
<td>A string</td>
<td></td>
<td>The default encoding set in the container's runtime environment</td>
</tr>
<tr>
<td>unauthenticatedIdentity</td>
<td>A principal name</td>
<td>none</td>
<td>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 <code>unchecked permission</code> constraint.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenyAll</td>
<td>org.jboss.security.authorization.modules.AllDenyAuthorizationModule</td>
</tr>
<tr>
<td>PermitAll</td>
<td>org.jboss.security.authorization.modules.AllPermitAuthorizationModule</td>
</tr>
<tr>
<td>Delegating</td>
<td>org.jboss.security.authorization.modules.DelegatingAuthorizationModule</td>
</tr>
<tr>
<td>Web</td>
<td>org.jboss.security.authorization.modules.WebAuthorizationModule</td>
</tr>
<tr>
<td>JACC</td>
<td>org.jboss.security.authorization.modules.JACCAuthorizationModule</td>
</tr>
</tbody>
</table>

Report a bug

### A.3. INCLUDED SECURITY MAPPING MODULES

The following security mapping roles are provided in JBoss EAP 6.

<table>
<thead>
<tr>
<th>Code</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropertiesRoles</td>
<td>org.jboss.security.mapping.providers.role.PropertiesRolesMappingProvider</td>
</tr>
<tr>
<td>SimpleRoles</td>
<td>org.jboss.security.mapping.providers.role.SimpleRolesMappingProvider</td>
</tr>
<tr>
<td>DeploymentRoles</td>
<td>org.jboss.security.mapping.providersDeploymentRolesMappingProvider</td>
</tr>
<tr>
<td>DatabaseRoles</td>
<td>org.jboss.security.mapping.providers.role.DatabaseRolesMappingProvider</td>
</tr>
<tr>
<td>LdapRoles</td>
<td>org.jboss.security.mapping.providers.role.LdapRolesMappingProvider</td>
</tr>
</tbody>
</table>

Report a bug
A.4. INCLUDED SECURITY AUDITING PROVIDER MODULES

JBoss EAP 6 provides one security auditing provider.

<table>
<thead>
<tr>
<th>Code</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogAuditProvider</td>
<td>org.jboss.security.audit.providers.LogAuditProvider</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>env-entry</td>
<td>A mapping to an env-entry required by the web.xml.</td>
</tr>
<tr>
<td>ejb-ref</td>
<td>A mapping to an ejb-ref required by the web.xml.</td>
</tr>
<tr>
<td>ejb-local-ref</td>
<td>A mapping to an ejb-local-ref required by the web.xml.</td>
</tr>
<tr>
<td>service-ref</td>
<td>A mapping to a service-ref required by the web.xml.</td>
</tr>
<tr>
<td>resource-ref</td>
<td>A mapping to a resource-ref required by the web.xml.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resource-env-ref</td>
<td>A mapping to a resource-env-ref required by the web.xml.</td>
</tr>
<tr>
<td>message-destination-ref</td>
<td>A mapping to a message-destination-ref required by the web.xml.</td>
</tr>
<tr>
<td>persistence-context-ref</td>
<td>A mapping to a persistence-context-ref required by the web.xml.</td>
</tr>
<tr>
<td>persistence-unit-ref</td>
<td>A mapping to a persistence-unit-ref required by the web.xml.</td>
</tr>
<tr>
<td>post-construct</td>
<td>A mapping to a post-context required by the web.xml.</td>
</tr>
<tr>
<td>pre-destroy</td>
<td>A mapping to a pre-destroy required by the web.xml.</td>
</tr>
<tr>
<td>data-source</td>
<td>A mapping to a data-source required by the web.xml.</td>
</tr>
<tr>
<td>context-root</td>
<td>The root context of the application. The default value is the name of the deployment without the .war suffix.</td>
</tr>
<tr>
<td>virtual-host</td>
<td>The name of the HTTP virtual-host the application accepts requests from. It refers to the contents of the HTTP Host header.</td>
</tr>
<tr>
<td>annotation</td>
<td>Describes an annotation used by the application. Refer to &lt;annotation&gt; for more information.</td>
</tr>
<tr>
<td>listener</td>
<td>Describes a listener used by the application. Refer to &lt;listener&gt; for more information.</td>
</tr>
<tr>
<td>session-config</td>
<td>This element fills the same function as the &lt;session-config&gt; element of the web.xml and is included for compatibility only.</td>
</tr>
<tr>
<td>valve</td>
<td>Describes a valve used by the application. Refer to &lt;valve&gt; for more information.</td>
</tr>
<tr>
<td>overlay</td>
<td>The name of an overlay to add to the application.</td>
</tr>
<tr>
<td>security-domain</td>
<td>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.</td>
</tr>
</tbody>
</table>
This element fills the same function as the \texttt{<security-role>} element of the \texttt{web.xml} and is included for compatibility only.

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.

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.

If \texttt{false}, the application is able to call another application context. Defaults to \texttt{true}.

The following elements each have child elements.

\texttt{<annotation>}

Describes an annotation used by the application. The following table lists the child elements of an \texttt{<annotation>}.

### Table A.43. Annotation Configuration Elements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class-name</td>
<td>Name of the class of the annotation</td>
</tr>
<tr>
<td>servlet-security</td>
<td>The element, such as \texttt{@ServletSecurity}, which represents servlet security.</td>
</tr>
<tr>
<td>run-as</td>
<td>The element, such as \texttt{@RunAs}, which represents the run-as information.</td>
</tr>
<tr>
<td>multi-part</td>
<td>The element, such as \texttt{@Multipart}, which represents the multi-part information.</td>
</tr>
</tbody>
</table>

\texttt{<listener>}

Describes a listener. The following table lists the child elements of a \texttt{<listener>}.

### Table A.44. Listener Configuration Elements

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class-name</td>
<td>Name of the class of the listener</td>
</tr>
<tr>
<td>listener-type</td>
<td>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.</td>
</tr>
<tr>
<td>module</td>
<td>The name of the module containing the listener class.</td>
</tr>
<tr>
<td>param</td>
<td>A parameter. Contains two child elements, param-name&gt; and param-value&gt;.</td>
</tr>
</tbody>
</table>

<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

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;security-identity&gt;</td>
<td>Contains child elements pertaining to the security identity of an EJB.</td>
</tr>
<tr>
<td>&lt;use-caller-identity /&gt;</td>
<td>Indicates that the EJB uses the same security identity as the caller.</td>
</tr>
<tr>
<td>&lt;run-as&gt;</td>
<td>Contains a &lt;role-name&gt; element.</td>
</tr>
</tbody>
</table>
### 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 `<servlet>`.

```xml
<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>
```

---

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;run-as-principal&gt;</code></td>
<td>If present, indicates the principal assigned to outgoing calls. If not present, outgoing calls are assigned to a principal named <code>anonymous</code>.</td>
</tr>
<tr>
<td><code>&lt;role-name&gt;</code></td>
<td>Specifies the role the EJB should run as.</td>
</tr>
<tr>
<td><code>&lt;description&gt;</code></td>
<td>Describes the role named in <code>&lt;role-name&gt;</code></td>
</tr>
</tbody>
</table>
## APPENDIX B. REVISION HISTORY

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0.0-3</td>
<td>Fri Jul 17 2015</td>
<td>Scott Mumford</td>
</tr>
</tbody>
</table>

Built from Content Specification: 13944, Revision: 765278 by smumford