Red Hat JBoss Enterprise Application Platform 7.0

How to Configure Server Security

For Use with Red Hat JBoss Enterprise Application Platform 7.0

Last Updated: 2018-02-08

Red Hat JBoss Enterprise Application Platform 7.0 How to Configure Server Security

For Use with Red Hat JBoss Enterprise Application Platform 7.0

Legal Notice

Abstract

The purpose of this document is to provide a practical guide to securing Red Hat JBoss Enterprise Application Platform. More specifically, this guide details how to secure all of the management interfaces on JBoss EAP. Before reading this guide, users should read through the Security Architecture document for Red Hat JBoss Enterprise Application Platform 7.0 and have a solid understanding of how JBoss EAP handles security. This document also makes use of the JBoss EAP CLI interface for performing configuration changes. When completing this document, readers should have a solid, working understanding of how to secure JBoss EAP.

Table of Contents

CHAPTER 1. OVERVIEW OF SECURITY

The basics of JBoss EAP security as well as general security concepts are covered in the Red Hat JBoss Enterprise Application Platform Security Architecture Guide. Prior to reading this guide, it is important to understand the basic information covered in the Security Architecture Guide around authentication, authorization, security realms, encryption, and SSL/TLS.

CHAPTER 2. SECURING THE SERVER AND ITS INTERFACES

2.1. BUILDING BLOCKS

2.1.1. Interfaces and Socket Bindings

JBoss EAP utilizes its host's interfaces (e.g. inet-address, nic, etc) and ports for communication for both its web applications as well as its management interfaces. These interfaces and ports are defined and configured through the *interfaces* and *socket-binding-groups* settings in the JBoss EAP configuration files (e.g. **`standalone.xml`**, **`domain.xml`**, **`host.xml`**, etc).

For more information on how to define and configure *interfaces* and *socket-binding-groups*, consult the Socket Bindings section of the Configuration Guide.

Example Interfaces

```
<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>
</interfaces>
```

Example Socket Binding Group

```
<socket-binding-group name="standard-sockets" default-interface="public"
port-offset="${jboss.socket.binding.port-offset:0}">
    <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
    <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}"/>
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
        <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
</socket-binding-group>
```

2.1.2. Security Realms

JBoss EAP uses security realms to define authentication and authorization mechanisms (e.g. local, LDAP, properties, etc) which can then be used by the management interfaces. For more background information on security realms, consult the *Security Realms* section of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide.

Example Security Realms

```
<security-realms>
```

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

NOTE

In addition to updating the existing security realms, JBoss EAP also allows you to create new security realms. You can create new security realms via the management console as well as invoking the following command from the management CLI:

```
/core-service=management/security-realm=NEW-REALM-NAME:add()
```

If you create a new security realm and want to use a properties file for authentication or authorization, you must create new properties files specifically for the new security domain. JBoss EAP does not reuse existing files used by other security domains nor does it automatically create new files specified in the configuration if they do not exist.

2.1.3. Using Security Realms and Socket Bindings for Securing the Management Interfaces

By default, JBoss EAP defines an http-interface to connect to the management interfaces. This interface is defined in the *<management-interfaces>* section of the JBoss EAP configuration:

```
<management-interfaces>
  <http-interface security-realm="ManagementRealm" http-upgrade-
enabled="true">
    <socket-binding http="management-http"/>
  </http-interface>
</management-interfaces>
```

Notice that the interface specifies a *security-realm* and *socket-binding*. Updating the configuration for the specified security realm and socket binding allows for the management interfaces to be secured in different ways. In addition to being able to secure each of these interfaces via security realms and socket

bindings, both of these interfaces also may be completely disabled, and users of these interfaces may be configured to have various roles and access rights. There are also a few topics in this guide, such as security auditing, secure passwords and JMX that overlap with other subsystems within JBoss EAP , but still relate to securing JBoss EAP.

2.2. HOW TO SECURE THE MANAGEMENT INTERFACES

The following sections show how to perform various operations related to securing the JBoss EAP management interfaces and related subsystems.

NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, please see the JBoss EAP *Management CLI Guide*.

2.2.1. Configuring the Networking and Ports Used by JBoss EAP

Depending on the configuration of the host, JBoss EAP may be configured to use various network interfaces and ports. This allows JBoss EAP to work with different host, networking, and firewall requirements.

For more information on the Networking and Ports used by JBoss EAP as well as how to configure those settings, please see the *Network and Port Configuration* section of the Configuration Guide.

2.2.2. Configure the Management Interfaces for HTTPS

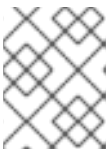
Configuring the JBoss EAP management interfaces for communication only using HTTPS provides increased security. All network traffic between the client and management interfaces is encrypted, which reduces the risk of security attacks such as a man-in-the-middle attack.

In this procedure unencrypted communication with the JBoss EAP instance is disabled. This procedure applies to both standalone server and managed domain configurations. For a managed domain, prefix the management CLI commands with the name of the host, for example: **`/host=master`**.

IMPORTANT

While performing the steps for enabling HTTPS on the management interfaces, do not reload the configuration unless explicitly instructed. Doing so may cause you to be locked out of the management interfaces.

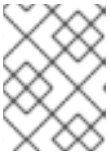
1. Create a keystore to secure the management interfaces.



NOTE

This keystore must be in JKS format as the management interfaces are not compatible with keystores in JCEKS format.

Use the following to generate a keystore, replacing the example values for the parameters, for example **`alias`**, **`keypass`**, **`keystore`**, **`storepass`** and **`dname`**, with the correct values for the environment.



NOTE

The parameter **validity** specifies for how many days the key is valid. A value of **730** equals two years.

Using the keytool command to generate a keystore from the terminal

```
$ keytool -genkeypair -alias appserver -storetype jks -keyalg RSA -
keysize 2048 -keypass password1 -keystore
EAP_HOME/standalone/configuration/identity.jks -storepass password1
-dname "CN=appserver,OU=Sales,O=Systems Inc,L=Raleigh,ST=NC,C=US" -
validity 730 -v
```

2. Ensure the management interfaces bind to HTTPS.

   a. Running a Standalone Server.
   To ensure the management interfaces bind to HTTPS, you must add the **management-https** configuration and remove the **management-http** configuration.

   Use the following CLI commands to bind the management interfaces to HTTPS:

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

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

   b. Running a Managed Domain
   Change the socket element within the **management-interface** section by adding **secure-port** and removing port configuration.

   Use the following commands to bind the management interfaces to HTTPS:

   ```
   /host=master/core-service=management/management-interface=http-
   interface:write-attribute(name=secure-port,value=9993)

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

3. **Optional:** Use custom socket-binding-group. If you want to use a custom **socket-binding-group**, you must ensure the **management-https** binding is defined, which by default is bound to port **9993**. You can verify this by reviewing the **socket-binding-group** section of the server's configuration file or by using the management CLI:

   Example Reading the socket-binding-group Configuration Using the Management CLI

   ```
   /socket-binding-group=standard-sockets/socket-binding=management-
   https:read-resource(recursive=true)

   {
       "outcome" => "success",
   ```

```
    "result" => {
        "client-mappings" => undefined,
        "fixed-port" => false,
        "interface" => "management",
        "multicast-address" => undefined,
        "multicast-port" => undefined,
        "name" => "management-https",
        "port" => expression "${jboss.management.https.port:9993}"
    }
}
```

4. Create a new security realm. In this example, the new security realm using HTTPS, ManagementRealmHTTPS, uses a properties file named **`https-mgmt-users.properties`** located in the **`EAP_HOME/standalone/configuration/`** directory for storing usernames and passwords. Usernames and passwords can be added to the file later, but for now, you need to create an empty file named **`https-mgmt-users.properties`** and save it to that location. The below example shows using the **`touch`** command, but you may also use other mechanisms, such as a text editor.

   Example using the `touch` command to create an empty file

   ```
   $ touch EAP_HOME/standalone/configuration/https-mgmt-
   users.properties
   ```

   Next, enter the following CLI commands to create a new security realm named ManagementRealmHTTPS:

   ```
   /core-service=management/security-realm=ManagementRealmHTTPS:add

   /core-service=management/security-
   realm=ManagementRealmHTTPS/authentication=properties:add(path=https-
   mgmt-users.properties,relative-to=jboss.server.config.dir)
   ```

   At this point, you have created a new security realm and configured it to use a properties file for authentication. You must now add users to that properties file using the **`add-user`** script, which is available in the **`EAP_HOME/bin/`** directory. When running the **`add-user`** script, you must specify both the properties file and the security realm using the **`-up`** and **`-r`** options respectively. From there, the **`add-user`** script will interactively prompt you for the username and password information to store in the **`https-mgmt-users.properties`** file.

   ```
   $ EAP_HOME/bin/add-user.sh -up
   EAP_HOME/standalone/configuration/https-mgmt-users.properties -r
   ManagementRealmHTTPS
   ...
   Enter the details of the new user to add.
   Using realm 'ManagementRealmHTTPS' as specified on the command line.
   ...
   Username : httpUser
   Password requirements are listed below. To modify these restrictions
   edit the add-user.properties configuration file.
    - The password must not be one of the following restricted values
   {root, admin, administrator}
    - The password must contain at least 8 characters, 1 alphabetic
   character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
   ```

```
 - The password must be different from the username
...
Password :
Re-enter Password :
About to add user 'httpUser' for realm 'ManagementRealmHTTPS'
...
Is this correct yes/no? yes
..
Added user 'httpUser' to file 'EAP_HOME/configuration/https-mgmt-
users.properties'
...
Is this new user going to be used for one AS process to connect to
another AS process?
e.g. for a slave host controller connecting to the master or for a
Remoting connection for server to server EJB calls.
yes/no? no
```

IMPORTANT

When configuring security realms that use properties files to store usernames and passwords, it is recommended that each realm use a distinct properties file that is not shared with another realm.

5. Configure the management interfaces to use the new security realm.

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

6. Configure the management interfaces to use the keystore.
Use the below CLI command to configure the management interfaces to use the keystore. For the parameters file, password and alias their values must be copied from the first step.

CLI Command for Adding a Keystore to a Security Realm

```
/core-service=management/security-realm=ManagementRealmHTTPS/server-
identity=ssl:add(keystore-path=identity.jks,keystore-relative-
to=jboss.server.config.dir,keystore-password=password1,
alias=appserver)
```

NOTE

To update the keystore password, use the following CLI command:

```
/core-service=management/security-
realm=ManagementRealmHTTPS/server-identity=ssl:write-
attribute(name=keystore-password,value=newpassword)
```

At this point, you need to reload the server's configuration:

```
reload
```

After reloading the server configuration, the log should contain the following, just before the text which states the number of services that are started:

```
13:50:54,160 INFO  [org.jboss.as] (Controller Boot Thread)
WFLYSRV0061: Http management interface listening on
https://127.0.0.1:9993/management
13:50:54,162 INFO  [org.jboss.as] (Controller Boot Thread)
WFLYSRV0052: Admin console listening on https://127.0.0.1:9993
```

The management interfaces are now listening on port **9993**, which confirms that the procedure was successful.

IMPORTANT

At this point, the CLI will disconnect and will be unable to reconnect since the port bindings have changed. Proceed to the next step to update the **jboss-cli.xml** to allow the CLI to reconnect.

7. Update the **jboss-cli.xml**.
   If using the management CLI to perform management actions, the following changes must to be made to the **EAP_HOME/bin/jboss-cli.xml** file:

   - Update the value of **<default-protocol>** to **https-remoting**.
   - In **<default-controller>**, update the value of **<protocol>** to **https-remoting**.
   - In **<default-controller>**, update the value of **<port>** to **9993**.

   Example jboss-cli.xml

```
<jboss-cli xmlns="urn:jboss:cli:2.0">
    <default-protocol use-legacy-override="true">https-
remoting</default-protocol>
    <!-- The default controller to connect to when 'connect'
command is executed w/o arguments -->
    <default-controller>
        <protocol>https-remoting</protocol>
        <host>localhost</host>
        <port>9993</port>
    </default-controller>
...
```

   The next time you connect to the management interface using the management CLI, you must accept the server certificate and authenticate against the **ManagementRealmHTTPS** security realm:

   Example Accepting Server Certificate and Authenticating

```
$ ./jboss-cli.sh -c
Unable to connect due to unrecognised server certificate
Subject    - CN=appserver,OU=Sales,O=Systems
Inc,L=Raleigh,ST=NC,C=US
Issuer     - CN=appserver, OU=Sales, O=Systems Inc, L=Raleigh,
ST=NC, C=US
```

```
Valid From - Tue Jun 28 13:38:48 CDT 2016
Valid To   - Thu Jun 28 13:38:48 CDT 2018
MD5 : 76:f4:81:8b:7e:c3:be:6d:ee:63:c1:7a:b7:b8:f0:fb
SHA1 :
ea:e3:f1:eb:53:90:69:d0:c9:69:4a:5a:a3:20:8f:76:c1:e6:66:b6

Accept certificate? [N]o, [T]emporarily, [P]ermenantly : p
Authenticating against security realm: ManagementRealmHTTPS
Username: httpUser
Password:
[standalone@localhost:9993 /]
```

2.2.3. Disabling Just the Management Console

Other clients, such as JBoss Operations Network, operate using the HTTP interface for managing JBoss EAP. In order to continue using these services, just the web-based management console itself may be disabled. This is accomplished by setting the *console-enabled* attribute to *false*:

CLI Command for Disabling the Web-Based Management Console

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

2.2.4. Setting up Two-Way SSL/TLS for the Management Interfaces

Two-way SSL/TLS authentication, also known as *client authentication*, authenticates both the client and the server using SSL/TLS certificates. This differs from the Configure the Management Interfaces for HTTPS section in that both the client and server each have a certificate. This provides assurance that not only is the server who it says it is, but the client is also who it says it is.

In this section the following conventions are used:

HOST1

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

HOST2

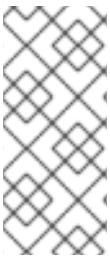
A suitable name for the client. For example: **`myclient`**. Note this is not necessarily an actual hostname.

CA_HOST1

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

CA_HOST2

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



PREREQUISITES

If a password vault is used to store the keystore and truststore passwords (recommended), the password vault should already be created. For more information on the password vault, please see the Password Vault section as well as the *Password Vault System* section of the Red Hat JBoss Enterprise Application Platform 7 Security Architecture Guide.

WARNING

Red Hat recommends that SSLv2, SSLv3, and TLSv1.0 be explicitly disabled in favor of TLSv1.1 or TLSv1.2 in all affected packages.

1. Generate the keystores.

```
$ keytool -genkeypair -alias HOST1_alias -keyalg RSA -keysize 1024 -
validity 365 -keystore HOST1.keystore.jks -dname "CA_HOST1" -keypass
secret -storepass secret

$ keytool -genkeypair -alias HOST2_alias -keyalg RSA -keysize 1024 -
validity 365 -keystore HOST2.keystore.jks -dname "CA_HOST2" -keypass
secret -storepass secret
```

2. Export the certificates.

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

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

3. Import the certificates into the opposing truststores.

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

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

4. Define a CertificateRealm.
   Define a CertificateRealm in the configuration for the server (**`host.xml`** or **`standalone.xml`**) and point the interface to it. This can be done using the following commands:

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

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

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

5. Change the **`security-realm`** of the **`http-interface`** to the new CertificateRealm.

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

6. Add the SSL/TLS configuration for the CLI.

IMPORTANT

In addition to adding the two-way SSL/TLS, the management interface should also be configured to bind to HTTPS.

Add the SSL/TLS configuration for the CLI, which uses **`EAP_HOME/bin/jboss-cli.xml`** as a settings file.

To store the keystore and truststore passwords in plain text, edit **`EAP_HOME/bin/jboss-cli.xml`** and add the SSL/TLS configuration using the appropriate values for the variables:

Example jboss-cli.xml XML

```
<ssl>
  <alias>HOST2_alias</alias>
  <key-store>/path/to/HOST2.keystore.jks</key-store>
  <key-store-password>secret</key-store-password>
  <trust-store>/path/to/HOST2.truststore.jks</trust-store>
  <trust-store-password>secret</trust-store-password>
  <modify-trust-store>true</modify-trust-store>
</ssl>
```

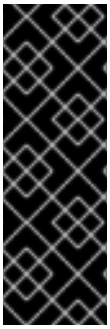
To use the keystore and truststore passwords stored in a password vault, you need to add the vault configuration and appropriate vault values to **`EAP_HOME/bin/jboss-cli.xml`**:

Example jboss-cli.xml XML

```
<ssl>
  <vault>
    <vault-option name="KEYSTORE_URL" value="path-
to/vault/vault.keystore"/>
    <vault-option name="KEYSTORE_PASSWORD" value="MASK-
5WNXs8oEbrs"/>
    <vault-option name="KEYSTORE_ALIAS" value="vault"/>
    <vault-option name="SALT" value="12345678"/>
    <vault-option name="ITERATION_COUNT" value="50"/>
    <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
  </vault>
  <alias>HOST2_alias</alias>
  <key-store>/path/to/HOST2.keystore.jks</key-store>
  <key-store-password>VAULT::VB::cli_pass::1</key-store-password>
  <key-password>VAULT::VB::cli_pass::1</key-password>
  <trust-store>/path/to/HOST2.truststore.jks</trust-store>
  <trust-store-password>VAULT::VB::cli_pass::1</trust-store-
password>
  <modify-trust-store>true</modify-trust-store>
</ssl>
```

2.2.5. Setting Up SSL/TLS for Applications

In addition to supporting HTTPS and two-way SSL/TLS for the management interfaces, JBoss EAP also enables SSL/TLS (via an HTTPS listener) to be set up for use by security domains.



IMPORTANT

As a prerequisite, an SSL/TLS Encryption Key and Certificate should be created and place in an accessible directory. Additionally, relevant information (e.g. keystore aliases and passwords, desired cipher suites, etc) should also be accessible. For examples on generating SSL/TLS Keys and Certificates, please see the first two steps in the Setting up Two-Way SSL/TLS for the Management Interfaces section. For more information about the HTTPS listener (including cipher suites) please see the HTTPS Listener Reference section.

Setting up One-Way SSL/TLS

This example assumes that the keystore, **`identity.jks`**, has been copied to the server configuration directory and configured with the given properties. Administrators should substitute their own values for the example ones.

NOTE

The management CLI commands shown assume that you are running a JBoss EAP standalone server. For more details on using the management CLI for a JBoss EAP managed domain, please see the JBoss EAP *Management CLI Guide*.

1. Add and configure an HTTPS security realm first. Once the HTTPS security realm has been configured, configure an **`https-listener`** in the **`undertow`** subsystem that references the security realm:

```
batch

/core-service=management/security-realm=HTTPSRealm/:add

/core-service=management/security-realm=HTTPSRealm/server-identity=
\
ssl:add(keystore-path=identity.jks, \
keystore-relative-to=jboss.server.config.dir, \
keystore-password=password1, alias=appserver)

/subsystem=undertow/server=default-server/https-listener=https:add(
\
socket-binding=https, security-realm=HTTPSRealm)

run-batch
```

WARNING

Red Hat recommends that SSLv2, SSLv3, and TLSv1.0 be explicitly disabled in favor of TLSv1.1 or TLSv1.2 in all affected packages.

2. Reload the server for the changes to take effect.

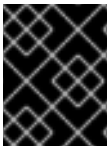
```
reload
```

2.2.6. Setting up Two-Way SSL/TLS for Applications

Setting up two-way SSL/TLS for applications follows many of the same procedures outlined in Setting up Two-Way SSL/TLS for the Management Interfaces. To set up Two-Way SSL/TLS for applications, you need to do the following:

1. Generate the stores for both the client and server
2. Export the certificates for both the client and server
3. Import the certificates into the opposing truststores
4. Define a security realm, for example **`CertificateRealm`**, on the server that uses the server's keystore and truststore
5. Update the **`undertow`** subsystem to use the security realm and require client verification

The first four steps are covered in Setting up Two-Way SSL/TLS for the Management Interfaces.



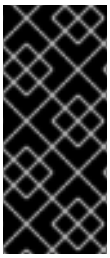
IMPORTANT

If the server has not been reloaded since the new security realm has been added, you must reload the server before performing the next step.

Update the Undertow Subsystem

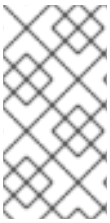
Once the keystores, certificates, truststores, and security realms have been created and configured, you need to add an HTTPS listener to the **`undertow`** subsystem, use the security realm you created, and require client verification:

```
/subsystem=undertow/server=default-server/https-listener=https:add( \
socket-binding=https, security-realm=CertificateRealm, verify-
client=REQUIRED)
```



IMPORTANT

Any client connecting to a JBoss EAP instance with two-way SSL/TLS enabled for applications must have access to a client certificate or keystore, in other words a client keystore whose certificate is included in the server's truststore. If the client is using a browser to connect to the JBoss EAP instance, you need to import that certificate or keystore into the browser's certificate manager.



NOTE

More details on using certificate-based authentication in applications in addition to two-way SSL/TLS with applications can be found in Configuring a Security Domain to Use Certificate-Based Authentication in *How to Configure Identity Management* for JBoss EAP.

2.2.7. HTTPS Listener Reference

For a full list of attributes available for the HTTPS listener, please see Undertow Subsystem Attributes in the JBoss EAP *Configuration Guide*.

2.2.7.1. About Cipher Suites

You can configure a list of the encryption ciphers which are allowed. For JSSE syntax, it must be a comma-separated list. For OpenSSL syntax, it must be a colon-separated list. Ensure that only one syntax is used. The default is the JVM default.

IMPORTANT

Using weak ciphers is a significant security risk. See http://www.nist.gov/manuscript-publication-search.cfm?pub_id=915295 for NIST recommendations on cipher suites.

For a list of available OpenSSL ciphers, see https://www.openssl.org/docs/manmaster/apps/ciphers.html#CIPHER-STRINGS. Note that the following are not supported:

- *@SECLEVEL*
- *SUITEB128*
- *SUITEB128ONLY*
- *SUITEB192*

For a list of the standard JSSE ciphers, see http://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#Cipher.

To update the list of enabled cipher suites, use the *enabled-cipher-suites* attribute of the HTTPS listener in the **`undertow`** subsystem.

Example CLI

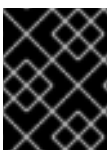
```
/subsystem=undertow/server=default-server/https-listener=https:write-
attribute(name=enabled-cipher-
suites,value="TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA")
```

NOTE

The example only lists two possible ciphers, but real-world examples will likely use more.

2.2.8. Enable FIPS 140-2 Cryptography for SSL/TLS on Red Hat Enterprise Linux 6

You can configure Undertow to use FIPS 140-2 compliant cryptography for SSL/TLS. The scope of this configuration example is limited to Red Hat Enterprise Linux 6, using the Mozilla NSS library in FIPS mode.



IMPORTANT

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

WARNING

The TLS 1.2 protocol is not supported by the Oracle/OpenJDK and JBoss EAP when running in FIPS mode and can cause a *NoSuchAlgorithmException* to occur. More details on this issue can be found here.

IMPORTANT

If you run **`jboss-cli.sh`** in an environment with FIPS 140-2 compliant cryptography for SSL/TLS enabled, you will see the following error: **`FIPS mode: only SunJSSE TrustManagers may be used`**. It is possible to workaround the issue by updating the **`javax.net.ssl.keyStore`** and **`javax.net.ssl.trustStore`** system properties in the **`jboss-cli.sh`** file:

```
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStore=NONE -
Djavax.net.ssl.trustStoreType=PKCS11"
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.keyStore=NONE -
Djavax.net.ssl.keyStoreType=PKCS11 -
Djavax.net.ssl.keyStorePassword=imapassword"
```

To configure Undertow to use FIPS 140-2 compliant cryptography for SSL/TLS, you must do the following:

- Configure the NSS database
- Configure Undertow

2.2.8.1. Configuring the NSS database

1. Create a directory owned by the appropriate user to house the NSS database.

   Example Commands for Creating the NSS Database Directory

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

   NOTE

   The *jboss* user is only an example. You need to replace it with a user on your operating system that you plan on using for running JBoss EAP.

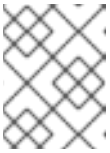
2. Create the NSS configuration file: **`/usr/share/jboss-as/nss_pkcsll_fips.cfg`**.
   It must specify:

   - a name
   - the directory where the NSS library is located

- the directory where the NSS database was created in the previous step

Example `nss_pkcsll_fips.cfg`

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



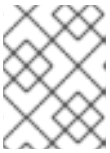
NOTE

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

3. Edit the **`$JAVA_HOME/jre/lib/security/java.security`** configuration file.
Add the following line to **`$JAVA_HOME/jre/lib/security/java.security`**:

Example `java.security`

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



NOTE

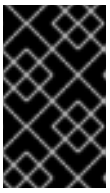
The **`nss_pkcsll_fips.cfg`** configuration file specified in the above line is the file created in the previous step.

You also need to update the following link in **`$JAVA_HOME/jre/lib/security/java.security`** from:

```
security.provider.5=com.sun.net.ssl.internal.ssl.Provider
```

to

```
security.provider.5=com.sun.net.ssl.internal.ssl.Provider
SunPKCS11-nss-fips
```



IMPORTANT

Any other **`security.provider.X`** lines in this file, for example **`security.provider.2`**, must have the value of their X increased by one to ensure that this provider is given priority.

4. Run the **`modutil`** command on the NSS database directory you created in the previous step to enable FIPS mode.

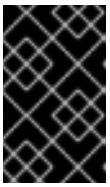
```
modutil -fips true -dbdir /usr/share/jboss-as/nssdb
```

NOTE

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

5. Set the password on the FIPS token.
   The name of the token *must* be *NSS FIPS 140-2 Certificate DB*.

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



IMPORTANT

The password used for the FIPS token must be a FIPS compliant password. If the password is not strong enough, you may receive an error: *ERROR: Unable to change password on token "NSS FIPS 140-2 Certificate DB".*

6. Create a certificate using the NSS tools.

   Example Command

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

2.2.8.2. Configuring Undertow

To complete the setup of FIPS 140-2 compliant cryptography for SSL/TLS:

1. Configure Undertow to use SSL/TLS.

NOTE

The following commands below must either be run in batch mode, or the server must be reloaded after adding the *ssl* server identity. The example below is shown using batch mode.

```
batch

/core-service=management/security-realm=HTTPSRealm:add

/core-service=management/security-realm=HTTPSRealm/server-
identity=ssl:add(keystore-provider=PKCS11, keystore-
password="strongP@ssword1")

/subsystem=undertow/server=default-server/https-
listener=https:add(socket-binding=https, security-realm=HTTPSRealm,
enabled-protocols="TLSv1.1")

run-batch
```

The basic details for configuring Undertow to SSL/TLS are covered in Setting up an SSL/TLS for Applications.

2. Configure the cipher suites used by Undertow.
   Once you have SSL/TLS configured, you need to configure the https listener and security realm to have a specific set of cipher suites enabled:

   Required Cipher Suites

   ```
   SSL_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
   TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
   TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA,
   TLS_DHE_DSS_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
   TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,
   TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
   TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
   TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
   TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
   TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
   TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
   TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
   TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
   TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
   TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
   TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
   TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,
   TLS_ECDH_anon_WITH_AES_128_CBC_SHA,
   TLS_ECDH_anon_WITH_AES_256_CBC_SHA
   ```

   The basics behind enabling cipher suites for the https listener are covered in About Cipher Suites. To enable cipher suites on the https listener:

   Example Command for Enabling Cipher Suites on the Https Listener

   ```
   /subsystem=undertow/server=default-server/https-
   listener=https:write-attribute(name=enabled-cipher-
   suites,value="SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_RSA_WITH_3DES_ED
   E_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_
   SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TL
   S_DHE_DSS_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_
   ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
   ,TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_3DES_EDE_C
   BC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES
   _256_CBC_SHA,TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_RSA_WITH_AE
   S_128_CBC_SHA,TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_3
   DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WIT
   H_AES_256_CBC_SHA,TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_anon_
   WITH_AES_128_CBC_SHA,TLS_ECDH_anon_WITH_AES_256_CBC_SHA")
   ```

3. Enable cipher suites on the security realm.

   Example Command for Enabling Cipher Suites on the Security Realm

   ```
   /core-service=management/security-realm=HTTPSRealm/server-
   identity=ssl:write-attribute(name=enabled-cipher-suites, value=
   ```

```
[SSL_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA, TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_anon_WITH_AES_128_CBC_SHA,
TLS_ECDH_anon_WITH_AES_256_CBC_SHA])
```

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

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

2.2.9. FIPS 140-2 Compliant Cryptography on IBM JDK

On the IBM JDK, the IBM JCE (Java Cryptographic Extension) IBMJCEFIPS provider and the IBM JSSE (Java Secure Sockets Extension) FIPS 140-2 Cryptographic Module (IBMJSSE2) for Multi-platforms provide FIPS 140-2 compliant cryptography.

For more information on the IBMJCEFIPS provider, refer to the IBM Documentation for IBM JCEFIPS, and the NIST IBMJCEFIPS – Security Policy. More information on IBMJSSE2, can be found here.

2.2.9.1. Key storage

The IBM JCE does not provide a keystore. The keys are stored on the computer and do not leave its physical boundary. If the keys are moved between computers they must be encrypted.

To run *keytool* in FIPS-compliant mode use the *-providerClass* option on each command like this:

```
keytool -list -storetype JCEKS -keystore mystore.jck -storepass mystorepass
-providerClass com.ibm.crypto.fips.provider.IBMJCEFIPS
```

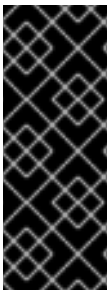
2.2.9.2. Examine FIPS provider information

To examine information about the IBMJCEFIPS used by the server, enable debug-level logging by adding *-Djavax.net.debug=true* to **`standalone.conf`** or **`domain.conf`**. Information about the FIPS provider is logged to **`server.log`**, for example:

```
04:22:45,685 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using
MessageDigest SHA from provider IBMJCEFIPS version 1.7
04:22:45,689 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH
KeyPairGenerator  from provider from init IBMJCEFIPS version 1.7
```

```
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using
KeyFactory DiffieHellman from provider IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using
KeyAgreement DiffieHellman from provider IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH
KeyAgreement  from provider IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH
KeyAgreement  from provider from initIBMJCEFIPS version 1.7
```

2.2.10. Starting a Managed Domain when the JVM is Running in FIPS Mode



IMPORTANT

It is assumed you have a managed domain, FIPS configured, as well as all necessary certificates configured. This includes having imported the domain controller's certificate into each controller's truststore. For more details on configuring a managed domain, see Domain Management in the JBoss EAP Configuration Guide. For more details on configuring FIPS, see Enable FIPS 140-2 Cryptography for SSL/TLS on Red Hat Enterprise Linux 6.

You need to update each host controller and the master domain controller to use SSL/TLS for communication.

WARNING

Red Hat recommends that SSLv2, SSLv3, and TLSv1.0 be explicitly disabled in favor of TLSv1.1 in all affected packages.

1. Create an SSL/TLS security realm on the master domain controller.
   You need to create an SSL/TLS security realm on the master domain controller configured to use your NSS database as a PKCS11 provider.

   Example Security Realm

```
<security-realm name="HTTPSRealm">
    <server-identities>
        <ssl>
            <engine enabled-protocols="TLSv1.1"/>
            <keystore provider="PKCS11" keystore-
password="strongP@ssword1"/>
        </ssl>
    </server-identities>
    <authentication>
        <local default-user="\$local"/>
        <properties path="https-users.properties" relative-
to="jboss.domain.config.dir"/>
    </authentication>
</security-realm>
```

2. Create an SSL/TLS security realm on each host controller.
   You need to create a security realm with an SSL/TLS truststore for authentication.

   Example Security Realm

```
<security-realm name="HTTPSRealm">
  <authentication>
    <truststore provider="PKCS11" keystore-
password="strongP@ssword1"/>
  </authentication>
</security-realm>
```

NOTE

You need to repeat this process on each host.

3. Secure the native interface on the master domain controller.
   You need to ensure that the native interface on the master domain controller is secured with the security realm you just created.

   Example Native Interface

```
<management-interfaces>
  ...
  <native-interface security-realm="HTTPSRealm">
    <socket interface="management"
port="${jboss.management.native.port:9999}"/>
   </native-interface>
</management-interfaces>
```

4. Use the SSL/TLS realm on each host controller to connect to the master domain controller.
   You need to update the security realm used for connecting to the master domain controller. This change must be done directly in the host controller's configuration file, for example **`host.xml`** or **`host-slave.xml`**, while the server is not running.

   Example Host Controller Configuration File

```
<domain-controller>
  <remote security-realm="HTTPSRealm">
    <discovery-options>
      <static-discovery name="primary"
protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}"
port="${jboss.domain.master.port:9999}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

5. Update how each server connects back to its host controller.
   You also need to update how each server connects back to its host controller.

   Example Server Configuration

```
<server name="my-server" group="my-server-group">
  <ssl ssl-protocol="TLS" trust-manager-algorithm="SunX509"
truststore-type="PKCS11" truststore-password="strongP@ssword1"/>
</server>
```

6. Configure two-way SSL/TLS in a managed domain.
   To enable two-way SSL/TLS, add a truststore authentication method to the SSL/TLS security realm for the master domain controller, execute the following management CLI commands:

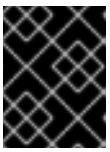
```
/host=master/core-service=management/security-
realm=HTTPSRealm/authentication=truststore:add(keystore-
provider="PKCS11",keystore-password="strongP@ssword1")

reload --host=master
```

   You also need to update each host controller's security realm to have an SSL server identity, execute the following management CLI commands:

```
/host=host1/core-service=management/security-
realm=HTTPSRealm/server-identity=ssl:add(keystore-provider=PKCS11,
keystore-password="strongP@ssword1",enabled-protocols=["TLSv1.1"])

reload --host=host1
```



IMPORTANT

You also need to ensure that each host's certificate is imported into the domain controller's truststore.

2.2.11. Disabling Remote Access to JMX

Remote access to the **jmx** subsystem allows for JDK and application management operations to be triggered remotely. To disable remote access to JMX in JBoss EAP, remove the remoting connector in the **jmx** subsystem:

Removing the Remoting Connector

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

For more information on JMX, please see the JMX section of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide

2.2.12. Using JAAS for Securing the Management Interfaces

JAAS is a declarative security API used by JBoss EAP to manage security. For more details and background regarding JAAS and declarative security, see the Declarative Security and JAAS section of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide.

NOTE

When JBoss EAP instances are configured to run in **ADMIN_ONLY** mode, using JAAS to secure the management interfaces is not supported. For more information on **ADMIN_ONLY** mode, please see the Running JBoss EAP in ADMIN_ONLY Mode section of the JBoss EAP *Configuration Guide*.

To use JAAS to authenticate to the management interfaces, the following steps must be performed:

1. Create a security domain.
   In this example, a security domain is created with the UserRoles login module, but other login modules may be used as well:

```
/subsystem=security/security-domain=UsersLMDomain:add(cache-
type=default)

/subsystem=security/security-
domain=UsersLMDomain/authentication=classic:add

/subsystem=security/security-
domain=UsersLMDomain/authentication=classic/login-
module=UsersRoles:add(code=UsersRoles, flag=required,module-options=
[("usersProperties"=>"users.properties"),
("rolesProperties"=>"roles.properties")])
```

2. Create a security realm with JAAS authentication.

```
/core-service=management/security-realm=SecurityDomainAuthnRealm:add

/core-service=management/security-
realm=SecurityDomainAuthnRealm/authentication=jaas:add(name=UsersLMD
omain)
```

3. Update the **`http-interface`** management interface to use new security realm.

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

4. **Optional:** Assign group membership.
   The attribute **`assign-groups`** determines whether loaded user membership information from the security domain is used for group assignment in the security realm. When set to **`true`**, this group assignment is used for Role-Based Access Control (RBAC).

```
/core-service=management/security-
realm=SecurityDomainAuthnRealm/authentication=jaas:write-
attribute(name=assign-groups,value=true)
```

2.2.13. Silent Authentication

The default installation of JBoss EAP 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 instances as well as domains.

IMPORTANT

The removal of the local element should only be done if the impact on the JBoss EAP instance and its configuration is fully understood.

To remove silent authentication from a realm:

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

2.2.14. Removing Undertow Response Headers

The default JBoss EAP **`undertow`** subsystem includes two response headers that are appended to each HTTP response by the **`default-host`**:

- **`Server`**, which is set to **`JBoss-EAP/7`**
- **`X-Powered-By`**, which is set to **`Undertow/1`**

Although these can be useful for development and debugging purposes, you might want to remove these headers if you do not want to disclose information about the server in use.

Use the following management CLI commands to remove these response headers from the **`default-host`**.

```
/subsystem=undertow/server=default-server/host=default-host/filter-
ref=server-header:remove

/subsystem=undertow/server=default-server/host=default-host/filter-ref=x-
powered-by-header:remove

reload
```

2.3. 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 or the management interfaces. Auditing mechanisms are configured as part of a security domain or management interface.

Auditing uses provider modules. Both included provider modules as well as custom implementations may be used.

2.3.1. Configure Security Auditing for Security Domains

To configure security auditing settings for a security domain, the following steps must be performed from the management console:

1. Open the security domain's detailed view.
   - Click *Configuration* at the top of the screen.
   - In a managed domain, select a profile to modify from the *Profile* selection box at the top left.
   - Click on *Subsystems*, then *Security*.
   - Click on the security domain to edit and click *View*.
2. Navigate to the Auditing configuration.
   Click on *Audit* at the left side of the screen.

   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 *Add* and fill in the *Code* section with the classname of the provider module. Also fill in the *Name* section with the desired name.
4. Verify the 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 includes the **`org.jboss.security.audit.providers.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 **`org.jboss.security.audit.providers.LogAuditProvider`**, perform an action that is likely to trigger a notification and then check the audit log file.
5. **Optional:** Add, edit, or remove module options.
   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 *Add*, and provide the key and value for the option.

   To edit an option that already exists, click *Remove* to remove it, and click *Add* to add it again with the correct options.

CHAPTER 3. SECURING A MANAGED DOMAIN

In addition to securing the management interfaces, you can also secure communication between JBoss EAP instances in a managed domain.

For information on concepts and general configuration for the managed domain operating mode, see the Domain Management section of the JBoss EAP *Configuration Guide*.

3.1. CONFIGURE PASSWORD AUTHENTICATION BETWEEN SLAVES AND THE DOMAIN CONTROLLER

When configuring a managed domain, by default, the master domain controller is configured to require authentication for each slave controller that connects to it. To configure slave controllers with the proper credentials, you must do the following:

1. Add a user to the master domain controller
   You need to add a user to the master domain controller using the *add-user* script. Specifically, you will need to ensure that the user is added to the same realm the master uses to secure its management interface, which by default is *ManagementRealm*. You also need to ensure you answer *yes* to the *Is this new user going to be used for one AS process to connect to another AS process?* question.

IMPORTANT

After adding the user, the script will output a *<secret>* element, which will be used in the next step. You must keep this value for use in the next step.

Example Adding a Slave User

```
$ EAP_HOME/bin/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.
Using realm 'ManagementRealm' as discovered from the existing
property files.
Username : slave-user
Password recommendations are listed below. To modify these
restrictions edit the add-user.properties configuration file.
 - The password should be different from the username
 - The password should not be one of the following restricted values
{root, admin, administrator}
 - The password should contain at least 8 characters, 1 alphabetic
character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
Password :
Re-enter Password :
What groups do you want this user to belong to? (Please enter a
comma separated list, or leave blank for none)[  ]:
About to add user 'slave-user' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'slave-user' to file '/home/user/EAP-7.0.0/jboss-eap-
```

```
7.0/standalone/configuration/mgmt-users.properties'
Added user 'slave-user' to file '/home/user/EAP-7.0.0/jboss-eap-
7.0/domain/configuration/mgmt-users.properties'
Added user 'slave-user' with groups  to file '/home/user/EAP-
7.0.0/jboss-eap-7.0/standalone/configuration/mgmt-groups.properties'
Added user 'slave-user' with groups  to file '/home/user/EAP-
7.0.0/jboss-eap-7.0/domain/configuration/mgmt-groups.properties'
Is this new user going to be used for one AS process to connect to
another AS process?
e.g. for a slave host controller connecting to the master or for a
Remoting connection for server to server EJB calls.
yes/no? yes
To represent the user add the following to the server-identities
definition <secret value="ABCzc3dv11Qx" />
```

2. Configure the slave controllers to use the credential.
   Once you have created the user on the master domain controller, you will need to update each slave controller to use that credential in the host configuration file, for example **`host.xml`** or **`host-slave.xml`**. To do so, you need to add the username to the *<remote>* element in the *domain controller* configuration. You will also need to add the *<secret>* to the *server identities* of the realm used to secure the *<remote>* element. Both the username and *<secret>* were obtained when adding the user to the master domain controller in the previous step.

   Example

```
...
<security-realm name="ManagementRealm">
    <server-identities>
        <!-- Replace this with either a base64 password of your own,
or use a vault with a vault expression -->
        <secret value="ABCzc3dv11Qx"/>
    </server-identities>
...
<domain-controller>
  <remote security-realm="ManagementRealm" username="slave-user">
      <discovery-options>
          <static-discovery name="primary"
protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}"
port="${jboss.domain.master.port:9999}"/>
      </discovery-options>
  </remote>
</domain-controller>
```

3.2. CONFIGURING SSL/TLS BETWEEN DOMAIN AND HOST CONTROLLERS

IMPORTANT

When you configure SSL/TLS to be used between JBoss EAP instances in a managed domain, each instance can have a client or server role depending on the interaction. This includes all host controllers as well as domain controllers. As a result, it is recommended that you setup two-way SSL/TLS between endpoints.

You can configure JBoss EAP instances in a managed domain to use SSL/TLS when communicating with each other, in other words, between the master domain controller and host controllers. To do so, you will need to do the following:

1. Generate and configure all necessary certificates and keystores.
   In order to set up two-way SSL/TLS between endpoints, you need to generate and configure certificates and keystores for the master domain controller as well as each host controller. You also need to import the certificate of the master domain controller into each host controller's keystore as well as import each host controller's certificate into the master domain controller's keystore. The specifics of this process is covered in a Setting up Two-Way SSL/TLS for the Management Interfaces.

2. Configure the master domain controller to use SSL/TLS.
   Once you have configured all certificates and keystores, you need to configure a security realm to use two-way SSL/TLS. This is done by configuring a security realm to use SSL/TLS and to require it for authentication. That security realm is then used to secure the management interface used for connecting between host controllers and the master domain controller.

NOTE

The following commands below must either be run in batch mode, or the server must be reloaded after adding the *ssl* server identity. The example below is shown using batch mode.

```
batch

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

/host=master/core-service=management/security-
realm=CertificateRealm/server-
identity=ssl:add(alias=domaincontroller,keystore-relative-
to=jboss.domain.config.dir,keystore-
path=domaincontroller.jks,keystore-password=secret)

/host=master/core-service=management/security-
realm=CertificateRealm/authentication=truststore:add(keystore-
relative-to=jboss.domain.config.dir,keystore-
path=domaincontroller.jks,keystore-password=secret)

/host=master/core-service=management/security-
realm=CertificateRealm/authentication=local:add(default-
user=\$local)

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

/host=master/core-service=management/management-interface=native-
interface:write-attribute(name=security-
realm,value=CertificateRealm)

run-batch
```

3. Configure all host controllers to use SSL/TLS.

Once you have the master domain controller configured to use two-way SSL/TLS, you need to configure each host controller to use it as well. The process is very much the same as the master domain controller configuration, except you will need to use the keystore specific to each host.

NOTE

The following commands below must either be run in batch mode, or the server must be reloaded after adding the *ssl* server identity. The example below is shown using batch mode.

```
batch

/host=instance1/core-service=management/security-
realm=CertificateRealm:add()

/host=instance1/core-service=management/security-
realm=CertificateRealm/server-
identity=ssl:add(alias=instance1,keystore-relative-
to=jboss.domain.config.dir,keystore-path=instance1.jks,keystore-
password=secret)

/host=instance1/core-service=management/security-
realm=CertificateRealm/authentication=truststore:add(keystore-
relative-to=jboss.domain.config.dir,keystore-
path=instance1.jks,keystore-password=secret)

/host=instance1/core-service=management/security-
realm=CertificateRealm/authentication=local:add(default-
user="\$local")

/host=instance1/core-service=management/security-
realm=CertificateRealm/authentication=properties:add(relative-
to=jboss.domain.config.dir,path=mgmt-users.properties)

/host=instance1/core-service=management/management-interface=native-
interface:write-attribute(name=security-
realm,value=CertificateRealm)

run-batch
```

Additionally, you will need to update the security realm used when connecting the master domain controller. This change must be done directly in the host controller's configuration file, e.g. **`host.xml`** or **`host-slave.xml`**, while the server is not running.

Example Host Controller Configuration File

```
<domain-controller>
        <remote security-realm="CertificateRealm" username="slave-
user">
            <discovery-options>
                <static-discovery name="primary"
protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}"
port="${jboss.domain.master.port:9999}"/>
```

```
            </discovery-options>
        </remote>
    </domain-controller>
```

WARNING

Red Hat recommends that SSLv2, SSLv3, and TLSv1.0 be explicitly disabled in favor of TLSv1.1 or TLSv1.2 in all affected packages.

CHAPTER 4. SECURING USERS OF THE SERVER AND ITS MANAGEMENT INTERFACES

In addition to understanding how to secure the various interfaces of JBoss EAP, its also important to understand how to secure the users that access those interfaces.

4.1. USER AUTHENTICATION

4.1.1. Default User Configuration

All management interfaces in JBoss EAP are secured by default and users can access them in two different ways: local interfaces and remote interfaces. The basics of both of these authentication mechanisms are covered in the Default Security and Red Hat JBoss Enterprise Application Platform Out of the Box sections of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide. By default, access to these interfaces is configured in the *Management Realm* security realm. Initially, the local interface is enabled and requires access to the host machine running the JBoss EAP instance. Remote access is also enabled and is configured to use a file-based identity store. By default it uses **`mgmt-users.properties`** file to store usernames and passwords, and **`mgmt-groups.properties`** to store user group information.

User information is added to these files by using the included **`adduser`** script located in the **`EAP_HOME/bin/`** directory.

To add a user via the `adduser` script:

1. Run the **`add-user.sh`** or **`add-user.bat`** command.
2. Choose whether to add a Management User or Application User.
3. Choose the realm the user will be added to. By default, the only available realms are ManagementRealm and ApplicationRealm. If a custom realm has been added, its name can be manually entered instead.
4. Type the desired username, password, and optional roles when prompted. The changes are written to each of the properties files for the security realm.

4.1.2. Adding Authentication via LDAP

JBoss EAP also supports using LDAP authentication for securing the management interfaces. The basics of LDAP and how it works with JBoss EAP are covered in the LDAP, Using LDAP with the Management Interfaces, and Using LDAP with the ManagementRealm sections of the Red Hat JBoss Enterprise Application Platform 7 Security Architecture Guide. For more specifics on how to secure the management interfaces using LDAP authentication, see the *Securing the Management Interfaces with LDAP* section of the How to Configure Identity Management guide.

4.2. SECURE PASSWORDS

4.2.1. Password Vault

Configuration of JBoss EAP and associated applications requires potentially sensitive information, such as usernames and passwords. Instead of storing the password as plain text in configuration files, the Password Vault feature can be used to mask the password information and store it in an encrypted

keystore. Once the password is stored, references can be included in management CLI commands or applications deployed to JBoss EAP.

The Password Vault uses the Java Keystore as its storage mechanism. Password Vault consists of two parts: storage and key storage. Java Keystore is used to store the key, which is used to encrypt or decrypt sensitive strings in Vault storage.

IMPORTANT

The keytool utility, provided by the Java Runtime Environment (JRE), is utilized for this steps. Locate the path for the file, which on Red Hat Enterprise Linux is `/usr/bin/keytool`.

JCEKS keystore implementations differ between Java vendors so the keystore must be generated using the keytool utility from the same vendor as the JDK used. Using a keystore generated by the keytool from one vendor's JDK in a JBoss EAP 7 instance running on a JDK from a different vendor results in the following exception: **java.io.IOException: com.sun.crypto.provider.SealedObjectForKeyProtector**

4.2.1.1. Set Up a Password Vault

Follow the steps below to set up and use a Password Vault.

1. Create a directory to store the keystore and other encrypted information.
   Create a directory to store the keystore and other important information. The rest of this procedure assumes that the directory is `EAP_HOME/vault/`. Since this directory will contain sensitive information it should be accessible to only limited users. At a minimum the user account under which JBoss EAP is running requires read-write access.

2. Determine the parameters to use with keytool utility.
   Decide on values for the following parameters:

   alias

   The alias is a unique identifier for the vault or other data stored in the keystore. Aliases are case-insensitive.

   storetype

   The storetype specifies the keystore type. The value *jceks* is recommended.

   keyalg

   The algorithm to use for encryption. Use the documentation for the JRE and operating system to see which other choices are available.

   keysize

   The size of an encryption key impacts how difficult it is to decrypt through brute force. For information on appropriate values, see the documentation distributed with the keytool utility.

   storepass

   The value of storepass is the password that 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. If this parameter is omitted, the keytool utility will prompt for it to be entered after the command has been executed

   keypass

   The value of keypass is the password used to access the specific key and must match the value of the storepass parameter.

validity

The value of validity is the period (in days) for which the key will be valid.

keystore

The value of keystore is the filepath and filename in which the keystore's values are to be stored. The keystore file is created when data is first added to it. Ensure the correct file path separator is used: / (forward slash) for Red Hat Enterprise Linux and similar operating systems, \ (backslash) for Microsoft Windows Server.
The **`keytool`** utility has many other options. See the documentation for the JRE or the operating system for more details.

3. Run the keytool command.

```
$ keytool -genseckey -alias vault -storetype jceks -keyalg AES -
keysize 128 -storepass vault22 -keypass vault22 -validity 730 -
keystore EAP_HOME/vault/vault.keystore
```

This results in a keystore that has been created in the file **`EAP_HOME/vault/vault.keystore`**. It stores a single key, with the alias vault, which will be used to store encrypted strings, such as passwords, for JBoss EAP.

4.2.1.2. Initialize the password vault.

The password vault can be initialized either interactively, where you are prompted for each parameter's value, or non-interactively, where all parameter values are provided on the command line. Each method gives the same result, so either may be used.

The following parameters will be needed:

keystore URL (KEYSTORE_URL)

The file system path or URI of the keystore file. The examples use **`EAP_HOME/vault/vault.keystore`**.

keystore password (KEYSTORE_PASSWORD)

The password used to access the keystore.

Salt (SALT)

The salt value is a random string of eight characters used, together with the iteration count, to encrypt the content of the keystore.

keystore Alias (KEYSTORE_ALIAS)

The alias by which the keystore is known.

Iteration Count (ITERATION_COUNT)

The number of times the encryption algorithm is run.

Directory to store encrypted files (ENC_FILE_DIR)

The path in which the encrypted files are to be stored. This is typically the directory containing the password vault. It is convenient but not mandatory to store all of your encrypted information in the same place as the keystore. This directory should be only accessible to limited users. At a minimum the user account under which JBoss EAP 7 is running requires read-write access. The keystore should be located in the directory you created when you set up the password vault. Note that the trailing backslash or forward slash on the directory name is required. Ensure the correct file path separator is used: / (forward slash) for Red Hat Enterprise Linux and similar operating systems, \ (backslash) for Microsoft Windows Server.

Vault Block (VAULT_BLOCK)

The name to be given to this block in the password vault.

Attribute (ATTRIBUTE)

The name to be given to the attribute being stored.

Security Attribute (SEC-ATTR)

The password which is being stored in the password vault.

To run the password vault command non-interactively, the *vault* script (located in **EAP_HOME/bin/**) can be invoked with parameters for the relevant information:

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD -
-alias KEYSTORE_ALIAS --vault-block VAULT_BLOCK --attribute ATTRIBUTE --
sec-attr SEC-ATTR --enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT --
salt SALT
```

Example

```
$ vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password
vault22 --alias vault --vault-block vb --attribute password --sec-attr
0penS3sam3 --enc-dir EAP_HOME/vault/ --iteration 120 --salt 1234abcd
```

Output

```
=========================================================================

  JBoss Vault

  JBOSS_HOME: EAP_HOME

  JAVA: java

=========================================================================

Nov 09, 2015 9:02:47 PM org.picketbox.plugins.vault.PicketBoxSecurityVault
init
INFO: PBOX00361: Default Security Vault Implementation Initialized and
Ready
WFLYSEC0047: Secured attribute value has been stored in Vault.
Please make note of the following:
********************************************
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
VAULT::vb::password::1
********************************************
WFLYSEC0048: Vault Configuration in WildFly configuration file:
********************************************

</extensions>
<vault>
  <vault-option name="KEYSTORE_URL"
value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
```

```
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
************************************************
```

To run the password vault command interactively, the following steps are required:

1. Launch the password vault command interactively.
   Run **`EAP_HOME/bin/vault.sh`** (on Red Hat Enterprise Linux and similar operating systems) or **`EAP_HOME\bin\vault.bat`** (on Microsoft Windows Server). Start a new interactive session by typing *0* (zero).

2. Complete the prompted parameters.
   Follow the prompts to input the required parameters.

3. Make a note of the masked password information.
   The masked password, salt, and iteration count are printed to standard output. Make a note of them in a secure location. They are required to add entries to the Password Vault. Access to the keystore file and these values could allow an attacker access to obtain access to sensitive information in the Password Vault.

4. Exit the interactive console
   Type *2* (two) to exit the interactive console.

Example Input and Output

```
Please enter a Digit::   0: Start Interactive Session  1: Remove
Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password: vault22
Enter Keystore password again: vault22
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault
init
INFO: PBOX000361: Default Security Vault Implementation Initialized and
Ready
Vault Configuration in AS7 config file:
************************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL"
value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
```

```
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
************************************************
Vault is initialized and ready for use
Handshake with Vault complete
```

+ The keystore password has been masked for use in configuration files and deployments. In addition, the vault is initialized and ready to use.

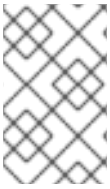
4.2.1.3. Configure JBoss EAP to use the password vault.

Before passwords and other sensitive attributes can be masked and used in configuration files, JBoss EAP 7 must be made aware of the password vault which stores and decrypts them.

The following command can be used to configure JBoss EAP 7 to use the password vault:

```
/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")])

/core-service=vault:add(vault-options=[("KEYSTORE_URL" =>
"EAP_HOME/vault/vault.keystore"),("KEYSTORE_PASSWORD" => "MASK-
5dOaAVafCSd"),("KEYSTORE_ALIAS" => "vault"),("SALT" => "1234abcd"),
("ITERATION_COUNT" => "120"),("ENC_FILE_DIR" => "EAP_HOME/vault/")])
```



NOTE

If Microsoft Windows Server is being used, use two backslashes (\\) in the file path instead using one. For example, **`C:\\data\\vault\\vault.keystore`**. This is because a single backslash character (\) is used for character escaping.

4.2.2. Store a Sensitive String in the Password Vault

Including passwords and other sensitive strings in plaintext configuration files is a security risk. Store these strings instead in the Password Vault for improved security, where they can then be referenced in configuration files, management CLI commands and applications in their masked form.

Sensitive strings can be stored in the Password Vault either interactively, where the tool prompts for each parameter's value, or non-interactively, where all the parameters' values are provided on the command line. Each method gives the same result, so either may be used. Both of these methods are invoked using the *vault* script.

To run the password vault command non-interactively, the *vault* script (located in **`EAP_HOME/bin/`**) can be invoked with parameters for the relevant information:

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD -
-alias KEYSTORE_ALIAS --vault-block VAULT_BLOCK --attribute ATTRIBUTE --
sec-attr SEC-ATTR --enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT --
salt SALT
```

NOTE

The keystore password must be given in plaintext form, not masked form.

Example

```
$ vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password
vault22 --alias vault --vault-block vb --attribute password --sec-attr
0penS3sam3 --enc-dir EAP_HOME/vault/ --iteration 120 --salt 1234abcd
```

Output

```
=========================================================================

  JBoss Vault

  JBOSS_HOME: EAP_HOME

  JAVA: java

=========================================================================

Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault
init
INFO: PBOX00361: Default Security Vault Implementation Initialized and
Ready
WFLYSEC0047: Secured attribute value has been stored in Vault.
Please make note of the following:
********************************************
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
VAULT::vb::password::1
********************************************
WFLYSEC0048: Vault Configuration in WildFly configuration file:
********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="../vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="../vault/"/>
</vault><management> ...
********************************************
```

After invoking the *vault* script, a message prints to standard output, showing the vault block, attribute name, masked string, and advice about using the string in your configuration. Make note of this information in a secure location. An extract of sample output is as follows:

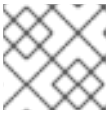
```
Vault Block:vb
Attribute Name:password
```

```
Configuration should be done as follows:
VAULT::vb::password::1
```

To run the password vault command interactively, the following steps are required:

1. Launch the Password Vault command interactively.
   Launch the operating system's command line interface and run **`EAP_HOME/bin/vault.sh`** (on Red Hat Enterprise Linux and similar operating systems) or **`EAP_HOME\bin\vault.bat`** (on Microsoft Windows Server). Start a new interactive session by typing *0* (zero).

2. Complete the prompted parameters.
   Follow the prompts to input the required parameters. These values must match those provided when the Password Vault was created.

   NOTE

   The keystore password must be given in plaintext form, not masked form.

3. Complete the prompted parameters about the sensitive string.
   Enter *0* (zero) to start storing the sensitive string. Follow the prompts to input the required parameters.

4. Make note of the information about the masked string.
   A message prints to standard output, showing the vault block, attribute name, masked string, and advice about using the string in the configuration. Make note of this information in a secure location. An extract of sample output is as follows:

   ```
   Vault Block:ds_Example1
   Attribute Name:password
   Configuration should be done as follows:
   VAULT::ds_Example1::password::1
   ```

5. Exit the interactive console.
   Type *2* (two) to exit the interactive console.

Example Input and Output

```
 ========================================================================
  JBoss Vault
  JBOSS_HOME: EAP_HOME
  JAVA: java
 ========================================================================
 ********************************
 ****  JBoss Vault  ***************
 ********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove
Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
```

```
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault
init
INFO: PBOX000361: Default Security Vault Implementation Initialized and
Ready
Vault Configuration in AS7 config file:
 ********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL"
value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
 ********************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::  0: Store a secured attribute  1: Check whether a
secured attribute exists  2: Remove secured attribute  3: Exit
0
Task: Store a secured attribute
Please enter secured attribute value (such as password):
Please enter secured attribute value (such as password) again:
Values match
Enter Vault Block:ds_Example1
Enter Attribute Name:password
Secured attribute value has been stored in vault.
Please make note of the following:
 ********************************************
Vault Block:ds_Example1
Attribute Name:password
Configuration should be done as follows:
VAULT::ds_Example1::password::1
 ********************************************
Please enter a Digit::  0: Store a secured attribute  1: Check whether a
secured attribute exists  2: Remove secured attribute  3: Exit
```

4.2.2.1. Use an Encrypted Sensitive String in Configuration

Any sensitive string which has been encrypted can be used in a configuration file or management CLI command in its masked form, providing expressions are allowed.

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

```
/subsystem=SUBSYSTEM:read-resource-description(recursive=true)
```

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

Use the following syntax to replace any plaintext string with the masked form.

```
${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::MASKED_STRING}
```

Example - Datasource Definition Using a Password in Masked Form

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

4.2.2.2. Use an Encrypted Sensitive String in an Application

Encrypted strings stored in the Password Vault can be used in an application's source code. The below example is an extract of a servlet's source code, illustrating the use of a masked password in a datasource definition, instead of the plaintext password. The plaintext version is commented out so that you can see the difference.

Servlet Using a Vaulted Password

```
@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "VAULT::DS::thePass::1",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)
/*old (plaintext) definition
@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "sa",
        className = "org.h2.jdbcx.JdbcDataSource",
```

```
        url = "jdbc:h2:tcp://localhost/mem:test"
)*/
```

4.2.2.3. Check if a Sensitive String is in the Password Vault

Before attempting to store or use a sensitive string in the Password Vault it can be useful to first confirm if it is already stored.

This check can be done either interactively, where the user is prompted for each parameter's value, or non-interactively, where all parameters' values are provided on the command line. Each method gives the same result, so either may be used. Both of these methods are invoked using the *vault* script.

Use the non-interative method to provide all parameters' values at once. For a description of all parameters, see Initialize the Password Vault. To run the password vault command non-interactively, the *vault* script (located in **EAP_HOME/bin/**) can be invoked with parameters for the relevant information:

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD --
-alias KEYSTORE_ALIAS --check-sec-attr --vault-block VAULT_BLOCK --
attribute ATTRIBUTE --enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT --
salt SALT
```

Substitute the placeholder values with the actual values. The values for parameters KEYSTORE_URL, KEYSTORE_PASSWORD and KEYSTORE_ALIAS must match those provided when the Password Vault was created.

NOTE

The keystore password must be given in plaintext form, not masked form.

If the sensitive string is stored in the vault block specified, the following message will be displayed:

```
Password already exists.
```

If the value is not stored in the specified block, the following message will be displayed:

```
Password doesn't exist.
```

To run the password vault command interactively, the following steps are required:

1. Launch the password vault command interactively.
   Run **EAP_HOME/bin/vault.sh** (on Red Hat Enterprise Linux and similar operating systems) or **EAP_HOME\bin\vault.bat** (on Microsoft Windows Server). Start a new interactive session by typing *0* (zero).
2. Complete the prompted parameters. Follow the prompts to input the required authentication parameters. These values must match those provided when the Password Vault was created.

   NOTE

   When prompted for authentication, the keystore password must be given in plaintext form, not masked form.

   - Enter *1* (one) to select “Check whether a secured attribute exists”.

- Enter the name of the vault block in which the sensitive string is stored.
- Enter the name of the sensitive string to be checked.

If the sensitive string is stored in the vault block specified, a confirmation message like the following will be output:

```
A value exists for (VAULT_BLOCK, ATTRIBUTE)
```

If the sensitive string is not stored in the specified block, a message like the following will be output:

```
No value has been store for (VAULT_BLOCK, ATTRIBUTE)
```

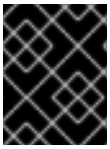
Example- Check For a Sensitive String Interactively

```
 ========================================================================
  JBoss Vault
  JBOSS_HOME: EAP_HOME
  JAVA: java
 ========================================================================
 ********************************
 ****  JBoss Vault  ***************
 ********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove
Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault
init
INFO: PBOX000361: Default Security Vault Implementation Initialized and
Ready
Vault Configuration in AS7 config file:
 ********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL"
value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
 ********************************************
Vault is initialized and ready for use
```

```
Handshake with Vault complete
Please enter a Digit::  0: Store a secured attribute  1: Check whether a
secured attribute exists  2: Remove secured attribute  3: Exit
1
Task: Verify whether a secured attribute exists
Enter Vault Block:vb
Enter Attribute Name:password
A value exists for (vb, password)
Please enter a Digit::  0: Store a secured attribute  1: Check whether a
secured attribute exists  2: Remove secured attribute  3: Exit
```

4.2.2.4. Remove a Sensitive String from the Password Vault

For security reasons it is best to remove sensitive strings from the Password Vault when they are no longer required. For example, if an application is being decommissioned, any sensitive strings used in datasource definitions should be removed at the same time.



IMPORTANT

As a prerequisite, before removing a sensitive string from the Password Vault, confirm if it is used in the configuration of JBoss EAP.

This operation can be done either interactively, where the user is prompted for each parameter's value, or non-interactively, where all parameters' values are provided on the command line. Each method gives the same result, so either may be used. Both of these methods are invoked using the *vault* script.

Use the non-interative method to provide all parameters' values at once. For a description of all parameters, see Initialize the Password Vault. To run the password vault command non-interactively, the *vault* script (located in **`EAP_HOME/bin/`**) can be invoked with parameters for the relevant information:

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD -
-alias KEYSTORE_ALIAS --remove-sec-attr --vault-block VAULT_BLOCK --
attribute ATTRIBUTE --enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT --
salt SALT
```

Substitute the placeholder values with the actual values. The values for parameters KEYSTORE_URL, KEYSTORE_PASSWORD and KEYSTORE_ALIAS must match those provided when the Password Vault was created.

NOTE

The keystore password must be given in plaintext form, not masked form.

If the sensitive string is successfully removed, a confirmation message like the following will be displayed:

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] has been successfully removed
from vault
```

If the sensitive string is not removed, a message like the following will be displayed:

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] was not removed from vault,
check whether it exist
```

Example Output

```
$ ./vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password
vault22 --alias vault --remove-sec-attr --vault-block vb --attribute
password --enc-dir EAP_HOME/vault/ --iteration 120 --salt 1234abcd
 ================================================================================
  JBoss Vault
  JBOSS_HOME: EAP_HOME
  JAVA: java
 ================================================================================
Dec 23, 2015 1:54:24 PM org.picketbox.plugins.vault.PicketBoxSecurityVault
init
INFO: PBOX000361: Default Security Vault Implementation Initialized and
Ready
Secured attribute [vb::password] has been successfully removed from vault
```

Remove a Sensitive String Interactively

To run the password vault command interactively, the following steps are required:

1. Launch the password vault command interactively.
   Run **`EAP_HOME/bin/vault.sh`** (on Red Hat Enterprise Linux and similar operating systems) or **`EAP_HOME\bin\vault.bat`** (on Microsoft Windows Server). Start a new interactive session by typing *0* (zero).

2. Complete the prompted parameters.
   Follow the prompts to input the required authentication parameters. These values must match those provided when the Password Vault was created.

   NOTE

   When prompted for authentication, the keystore password must be given in plaintext form, not masked form.

   - Enter *2* (two) to choose option Remove secured attribute.
   - Enter the name of the vault block in which the sensitive string is stored.
   - Enter the name of the sensitive string to be removed.

If the sensitive string is successfully removed, a confirmation message like the following will be displayed:

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] has been successfully removed
from vault
```

If the sensitive string is not removed, a message like the following will be displayed:

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] was not removed from vault,
check whether it exist
```

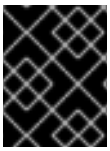
Example Output

```
 ********************************
```

```
 ****  JBoss Vault  ***************
 ********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove
Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Dec 23, 2014 1:40:56 PM org.picketbox.plugins.vault.PicketBoxSecurityVault
init
INFO: PBOX000361: Default Security Vault Implementation Initialized and
Ready
Vault Configuration in configuration file:
 *********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL"
value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
 *********************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::  0: Store a secured attribute  1: Check whether a
secured attribute exists  2: Remove secured attribute  3: Exit
2
Task: Remove secured attribute
Enter Vault Block:vb
Enter Attribute Name:password
Secured attribute [vb::password] has been successfully removed from vault
```

4.2.2.5. Configure Red Hat JBoss Enterprise Application Platform to Use a Custom Implementation of the Password Vault

In addition to using the provided Password Vault implementation, a custom implementation of SecurityVault may also be used.



IMPORTANT

As a prerequisite, please ensure that the Password Vault has been initialized. For more information, please see Initialize the Password Vault.

To use a custom implementation for the Password vault:

1. Create a class that implements the interface *SecurityVault*.
2. Create a module containing the class from the previous step, and specify a dependency on *org.picketbox* where the interface is *SecurityVault*.
3. Enable the custom Password Vault in the JBoss EAP configuration by adding the vault element with the following attributes:
   - **code** - The fully qualified name of class that implements SecurityVault.
   - **module** - The name of the module that contains the custom class.

Optionally, the *vault-options* parameters can be used to initialize the custom class for a Password Vault.

Example - Use vault-options Parameters to Initialize the Custom Class

```
/core-service=vault:add(
code="custom.vault.implementation.CustomSecurityVault",
module="custom.vault.module", 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")])
```

4.2.2.6. Obtain Keystore Password From External Source

The EXT, EXTC, CMD, CMDC or CLASS methods can be used in Vault configuration for obtaining the Java keystore password.

```
<vault-option name="KEYSTORE_PASSWORD" value="[here]"/>
```

The description for the methods are listed as:

{EXT}…

Refers to the exact command, where '…' is the exact command. For example: *{EXT}/usr/bin/getmypassword --section 1 --query company*, run the **`/usr/bin/getmypassword`** command, which displays the password on standard output and use it as password for Security Vault's keystore. In this example, the command is using two options: *--section 1* and *--query company*.

{EXTC[:expiration_in_millis]}…

Refers to the exact command, where the *…* is the exact command line that is passed to the *Runtime.exec(String)* method to execute a platform command. The first line of the command output is used as the password. EXTC variant caches the passwords for *expiration_in_millis* milliseconds. Default cache expiration is *0 = infinity*. For example: *{EXTC:120000}/usr/bin/getmypassword --section 1 --query company* verifies if the cache contains **`/usr/bin/getmypassword`** output, if it contains the output then use it. If it does not contain the output, run the command to output it to cache and use it. In this example, the cache expires in 2 minutes (120000 milliseconds).

{CMD}… or {CMDC[:expiration_in_millis]}…

The general command is a string delimited by *,* (comma) where the first part is the actual command and further parts represents the parameters. The comma can be backslashed to keep it as a part of the parameter. For example, *{CMD}/usr/bin/getmypassword,--section,1,--query,company*.

{CLASS[@jboss_module_spec]}classname[:ctorargs]

Where the *[:ctorargs]* is an optional string delimited by the *:* (colon) from the classname is passed to the classname *ctor*. The *ctorargs* is a comma delimited list of strings. For example,

{CLASS@org.test.passwd}org.test.passwd.ExternamPassworProvider. In this example, the *org.test.passwd.ExternamPassworProvider* class is loaded from *org.test.passwd* module and uses the *toCharArray()* method to get the password. If *toCharArray()* is not available the *toString()* method is used. The *org.test.passwd.ExternamPassworProvider* class must have the default constructor.

4.3. ROLE-BASED ACCESS CONTROL

The basics of or Role-Based Access Control are covered in the Role-Based Access Control and Adding RBAC to the Management Interfaces sections of the *Security Architecture Guide*.

4.3.1. Enabling Role-Based Access Control

By default the Role-Based Access Control (RBAC) system is disabled. It is enabled by changing the *provider* attribute from *simple* to *rbac*. *provider* is attribute of the *access-control* element of the *management* element. This can be done using the management CLI or by editing the server configuration XML file if the server is offline. When RBAC is disabled or enabled on a running server, the server configuration must be reloaded before it takes effect.

Once enabled it can only be disabled by a user of the *Administrator* or *SuperUser* roles. By default the management CLI runs as the *SuperUser* role if it is run on the same machine as the server.

To enable RBAC with the management CLI, use the *write-attribute* operation of the access authorization resource to set the *provider* attribute to *rbac*.

CLI to Enable RBAC

```
/core-service=management/access=authorization:write-
attribute(name=provider, value=rbac)
```

To disable RBAC with the management CLI, use the *write-attribute* operation of the access authorization resource to set the *provider* attribute to *simple*.

CLI to Disable RBAC

```
/core-service=management/access=authorization:write-
attribute(name=provider, value=simple)
```

If the server is offline the XML configuration can be edited to enable or disable RBAC. To do this, edit the provider attribute of the access-control element of the management element. Set the value to *rbac* to enable, and *simple* to disable.

Example XML

```
<management>
  <access-control provider="rbac">
    <role-mapping>
      <role name="SuperUser">
        <include>
          <user name="$local"/>
        </include>
      </role>
    </role-mapping>
  </access-control>
</management>
```

4.3.2. Changing the Permission Combination Policy

The Permission Combination Policy determines how permissions are determined if a user is assigned more than one role. This can be set to permissive or rejecting. The default is *permissive*.

When set to *permissive*, if any role is assigned to the user that permits an action, then the action is allowed.

When set to *rejecting*, if multiple roles are assigned to a user, then no action is allowed. This means that when the policy is set to rejecting each user should only be assigned one role. Users with multiple roles will not be able to use the management console or the management CLI when the policy is set to rejecting.

The Permission Combination Policy is configured by setting the *permission-combination-policy* attribute to either *permissive* or *rejecting*. This can be done using the management CLI or by editing the server configuration XML file if the server is offline. The *permission-combination-policy* attribute is part of the *access-control* element and the *access-control* element can be found in the *management* element.

Setting the Permission Combination Policy

Use the write-attribute operation of the access authorization resource to set the permission-combination-policy attribute to the required policy name.

```
/core-service=management/access=authorization:write-
attribute(name=permission-combination-policy, value=POLICYNAME)
```

The valid policy names are *rejecting* and *permissive*.

Example CLI

```
/core-service=management/access=authorization:write-
attribute(name=permission-combination-policy, value=rejecting)
```

If the server is offline the XML configuration can be edited to change the permission combination policy value. To do this, edit the permission-combination-policy attribute of the access-control element.

Example XML

```
<access-control provider="rbac" permission-combination-policy="rejecting">
  <role-mapping>
    <role name="SuperUser">
      <include>
        <user name="$local"/>
      </include>
    </role>
  </role-mapping>
</access-control>
```

4.3.3. Managing Roles

When Role-Based Access Control (RBAC) is enabled, what a management user is permitted to do is determined by the roles to which the user is assigned. JBoss EAP 7 uses a system of includes and excludes based on both the user and group membership to determine to which role a user belongs.

A user is considered to be assigned to a role if the user is:

- listed as a user to be included in the role, or
- a member of a group that is listed to be included in the role.

A user is also considered to be assigned to a role if the user is not:

- listed as a user to exclude from the role, or
- a member of a group that is listed to be excluded from the role.

Exclusions take priority over inclusions.

Role include and exclude settings for users and groups can be configured using both the management console and the management CLI.

Only users of the *SuperUser* or *Administrator* roles can perform this configuration.

4.3.3.1. Configure User Role Assignment using the Management CLI

The configuration of mapping users and groups to roles is located at: */core-service=management/access=authorization* as *role-mapping* elements.

Only users of the *SuperUser* or *Administrator* roles can perform this configuration.

Viewing Role Assignment Configuration

Use the *:read-children-names* operation to get a complete list of the configured roles:

```
/core-service=management/access=authorization:read-children-names(child-
type=role-mapping)
{
    "outcome" => "success",
    "result" => [
        "Administrator",
        "Deployer",
        "Maintainer",
        "Monitor",
        "Operator",
        "SuperUser"
    ]
}
```

Use the *read-resource* operation of a specified role-mapping to get the full details of a specific role:

```
/core-service=management/access=authorization/role-mapping=ROLENAME:read-
resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "include-all" => false,
        "exclude" => undefined,
        "include" => {
            "user-theboss" => {
                "name" => "theboss",
                "realm" => undefined,
                "type" => "USER"
```

```
            },
            "user-harold" => {
                "name" => "harold",
                "realm" => undefined,
                "type" => "USER"
            },
            "group-SysOps" => {
                "name" => "SysOps",
                "realm" => undefined,
                "type" => "GROUP"
            }
        }
    }
}
```

Add a new role

This procedure shows how to add a role-mapping entry for a role. This must be done before the role can be configured.

Use the *add* operation to add a new role configuration.

```
/core-service=management/access=authorization/role-mapping=ROLENAME:add
```

- ROLENAME is the name of the role that the new mapping is for (e.g. Auditor).

Example CLI

```
/core-service=management/access=authorization/role-mapping=Auditor:add
```

Add a user as included in a role

This procedure shows how to add a user to the included list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be done first.

Use the *add* operation to add a user entry to the includes list of the role.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:add(name=USERNAME, type=USER)
```

- ROLENAME is the name of the role being configured. (e.g. Auditor)
- ALIAS is a unique name for this mapping. Red Hat recommends that the use of a naming convention for aliases such as user-USERNAME. (e.g. user-max)
- USERNAME is the name of the user being added to the include list. (e.g. max)

Example CLI

```
/core-service=management/access=authorization/role-
mapping=Auditor/include=user-max:add(name=max, type=USER)
```

Add a user as excluded in a role

This procedure shows how to add a user to the excluded list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be done first.

Use the *add* operation to add a user entry to the excludes list of the role.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:add(name=USERNAME, type=USER)
```

- ROLENAME is the name of the role being configured. (e.g. Auditor)
- USERNAME is the name of the user being added to the exclude list. (e.g. max)
- ALIAS is a unique name for this mapping. Red Hat recommends that the use of a naming convention for aliases such as user-USERNAME. (e.g. user-max)

Example CLI

```
/core-service=management/access=authorization/role-
mapping=Auditor/exclude=user-max:add(name=max, type=USER)
```

Remove user role include configuration

This procedure shows how to remove a user include entry from a role mapping.

Use the *remove* operation to remove the entry.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:remove
```

- ROLENAME is the name of the role being configured (e.g. Auditor)
- ALIAS is a unique name for this mapping. Red Hat recommends that the use of a naming convention for aliases such as user-USERNAME. (e.g. user-max)

Example CLI

```
/core-service=management/access=authorization/role-
mapping=Auditor/include=user-max:remove
```

NOTE

Removing the user from the list of includes does not remove the user from the system, nor does it guarantee that the role will not be assigned to the user. The role might still be assigned based on group membership.

Remove user role exclude configuration

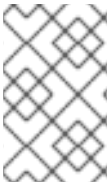
This procedure shows how to remove an user exclude entry from a role mapping.

Use the remove operation to remove the entry.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:remove
```

- ROLENAME is the name of the role being configured. (e.g. Auditor)
- ALIAS is a unique name for this mapping. Red Hat recommends that the use of a naming convention for aliases such as user-USERNAME. (e.g. user-max)

```
/core-service=management/access=authorization/role-
mapping=Auditor/exclude=user-max:remove
```



NOTE

Removing the user from the list of excludes does not remove the user from the system, nor does it guarantee the role will be assigned to the user. Roles might still be excluded based on group membership.

4.3.4. Roles and User Groups

Users authenticated using either the **`mgmt-users.properties`** file or an LDAP server, can be members of user groups. A user group is an arbitrary label that can be assigned to one or more users.

The RBAC system can be configured to automatically assign roles to users depending on what user groups they are members of. It can also exclude users from roles based on group membership.

When using the **`mgmt-users.properties`** file, group information is stored in the **`mgmt-groups.properties`** file. When using LDAP the group information is stored in the LDAP sever and maintained by those responsible for the LDAP server.

4.3.5. Configure Group Role Assignment using the Management CLI

Groups to be included or excluded from a role can be configured in the management console and the management CLI. This topic only shows using the management CLI.

The configuration of mapping users and groups to roles is located in the management API at: */core-service=management/access=authorization* as *role-mapping* elements.

Only users in the *SuperUser* or *Administrator* roles can perform this configuration.

Viewing Group Role Assignment Configuration

Use the *read-children-names* operation to get a complete list of the configured roles:

```
/core-service=management/access=authorization:read-children-names(child-
type=role-mapping)
{
    "outcome" => "success",
    "result" => [
        "Administrator",
        "Deployer",
        "Maintainer",
        "Monitor",
        "Operator",
        "SuperUser"
    ]
}
```

Use the *read-resource* operation of a specified role-mapping to get the full details of a specific role:

```
/core-service=management/access=authorization/role-mapping=ROLENAME:read-
resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "include-all" => false,
        "exclude" => undefined,
        "include" => {
            "user-theboss" => {
                "name" => "theboss",
                "realm" => undefined,
                "type" => "USER"
            },
            "user-harold" => {
                "name" => "harold",
                "realm" => undefined,
                "type" => "USER"
            },
            "group-SysOps" => {
                "name" => "SysOps",
                "realm" => undefined,
                "type" => "GROUP"
            }
        }
    }
}
```

Add a new role

This procedure shows how to add a role-mapping entry for a role. This must be done before the role can be configured.

Use the *add* operation to add a new role configuration.

```
/core-service=management/access=authorization/role-mapping=ROLENAME:add
```

Add a Group as included in a role

This procedure shows how to add a Group to the included list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be done first.

Use the add operation to add a Group entry to the includes list of the role.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:add(name=GROUPNAME, type=GROUP)
```

- ROLENAME is the name of the role being configured. (e.g. Auditor)
- GROUPNAME is the name of the group being added to the include list. (e.g. investigators)
- ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as group-GROUPNAME. (e.g. group-investigators)

Example CLI

-

```
/core-service=management/access=authorization/role-
mapping=Auditor/include=group-investigators:add(name=investigators,
type=GROUP)
```

Add a group as excluded in a role

This procedure shows how to add a group to the excluded list of a role.

If no configuration for a role has been done, then a role-mapping entry for it must be created first.

Use the add operation to add a group entry to the excludes list of the role.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:add(name=GROUPNAME, type=GROUP)
```

- ROLENAME is the name of the role being configured (e.g. Auditor)
- GROUPNAME is the name of the group being added to the include list (e.g. supervisors)
- ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as group-GROUPNAME. (e.g. group-supervisors)

Example CLI

```
/core-service=management/access=authorization/role-
mapping=Auditor/exclude=group-supervisors:add(name=supervisors,
type=GROUP)
```

Remove group role include configuration

This procedure shows how to remove a group include entry from a role mapping.

Use the remove operation to remove the entry.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:remove
```

- ROLENAME is the name of the role being configured (e.g. Auditor)
- ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as group-GROUPNAME. (e.g. group-investigators)

Example CLI

```
/core-service=management/access=authorization/role-
mapping=Auditor/include=group-investigators:remove
```

NOTE

Removing the group from the list of includes does not remove the group from the system, nor does it guarantee that the role will not be assigned to users in this group. The role might still be assigned to users in the group individually.

Remove a user group exclude entry

This procedure shows how to remove a group exclude entry from a role mapping.

Use the remove operation to remove the entry.

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:remove
```

- ROLENAME is the name of the role being configured. (e.g. Auditor)
- ALIAS is a unique name for this mapping. Red Hat recommends that you use a naming convention for your aliases such as group-GROUPNAME. (e.g. group-supervisors)

```
/core-service=management/access=authorization/role-
mapping=Auditor/exclude=group-supervisors:remove
```

NOTE

Removing the group from the list of excludes does not remove the group from the system. It also does not guarantee the role will be assigned to members of the group. Roles might still be excluded based on group membership.

4.3.6. Using RBAC with LDAP

The basics of using RBAC with LDAP as well as how to configure JBoss EAP 7 to use RBAC with LDAP are covered in the *LDAP and RBAC* section in the How to Configure Identity Management guide.

4.3.7. Scoped Roles

Scoped Roles are user-defined roles that grant the permissions of one of the standard roles but only for one or more specified server groups or hosts in an JBoss EAP managed domain. Scoped roles allow for management users to be granted permissions that are limited to only those server groups or hosts that are required.

IMPORTANT

Scoped roles can be created by users assigned the *Administrator* or *SuperUser* roles.

They are defined by five characteristics:

- A unique name.
- Which of the standard roles it is based on.
- If it applies to Server Groups or Hosts
- The list of server groups or hosts that it is restricted to.
- If all users are automatically included. This defaults to false.

Once created a scoped role can be assigned to users and groups the same way that the standard roles are.

Creating a scoped role does not allow for defining new permissions. Scoped roles can only be used to apply the permissions of an existing role in a limited scope. For example, a scoped role could be created based on the *Deployer* role which is restricted to a single server group.

There are only two scopes that roles can be limited to:

Host-scoped roles

A role that is host-scoped restricts the permissions of that role to one or more hosts. This means access is provided to the relevant */host=*/* resource trees but resources that are specific to other hosts are hidden.

Server-Group-scoped roles

A role that is server-group-scoped restricts the permissions of that role to one or more server groups. Additionally the role permissions will also apply to the profile, socket binding group, server configuration, and server resources that are associated with the specified *server-groups*. Any sub-resources within any of those that are not logically related to the server-group will not be visible to the user.

4.3.7.1. Configuring Scoped Roles from the Management CLI

IMPORTANT

Only users in the *SuperUser* or *Administrator* roles can perform this configuration.

Add a New Scoped Role

To add a new Scoped Role, the following operations must be done:

```
/core-service=management/access=authorization/role-mapping=NEW-SCOPED-
ROLE:add
```

```
/core-service=management/access=authorization/server-group-scoped-
role=NEW-SCOPED-ROLE:add(base-role=BASE-ROLE, server-groups=[SERVER-GROUP-
NAME])
```

Replace NEW-SCOPED-ROLE, BASE-ROLE, and SERVER-GROUP-NAME with the proper information.

Viewing and Editing a Scoped Role Mapping

A Scoped Role's details (including members) can be view by issuing the following command:

```
/core-service=management/access=authorization/role-mapping=NEW-SCOPED-
ROLE:read-resource(recursive=true)
```

Replace NEW-SCOPED-ROLE with the proper information.

To edit a Scoped Role's details, the *write-attribute* command may be used. For example:

```
/core-service=management/access=authorization/role-mapping=NEW-SCOPED-
ROLE:write-attribute(name=include-all, value=true)
```

Replace NEW-SCOPED-ROLE with the proper information.

Delete a Scoped Role

```
/core-service=management/access=authorization/role-mapping=NEW-SCOPED-ROLE:remove
```

```
/core-service=management/access=authorization/server-group-scoped-role=NEW-SCOPED-ROLE:remove
```

Replace NEW-SCOPED-ROLE with the proper information.

IMPORTANT

A Scoped Role cannot be deleted if users or groups are assigned to it. Remove the role assignments first, and then delete it.

Adding and Removing Users

Adding and removing users to and from Scoped Roles follows the same process as adding and removing standard roles.

4.3.7.2. Configuring Scoped Roles from the Management Console

IMPORTANT

Only users in the *SuperUser* or *Administrator* roles can perform this configuration.

Scoped role configuration in the management console can be found by following these steps:

1. Login to the management console
2. Click on the *Access Control* tab
3. Click on the *Roles* menu on the left and all roles (including scoped roles) are displayed.

The following procedures show how to perform configuration tasks for scoped roles.

Add a New Scoped Role

- Login to the management console
- Click on the *Access Control* tab
- Click on the *Roles* menu on the left.
- Click *Add*.
- Specify the following details:
  - **Name**, the unique name for the new scoped role.
  - **Base Role**, the role which this role will base its permissions on.
  - **Type**, whether this role will be restricted to hosts or server groups.
  - **Scope**, the list of hosts or server groups that the role is restricted to. Multiple entries can be selected.

- **Include All**, should this role automatically include all users. Defaults to no.

- Click *Save* and the dialog will close and the newly created role will appear in the table.

Edit a Scoped Role

- Login to the management console
- Click on the *Access Control* tab
- Click on the *Roles* menu on the left.
- Click on the desired scoped role to edit and click *Edit*.
- Update the desired details to change and click the *Save* button.

View Scoped Role Members

- Login to the management console
- Click on the *Access Control* tab
- Click on the *Roles* menu on the left.
- Click on the desired scoped role and choose *Include* or *Exclude* to view the included or excluded members.

Delete a Scoped Role

- Login to the management console
- Click on the *Access Control* tab
- Click on the *Roles* menu on the left.
- Click on the desired scoped role, click the dropdown arrow next to the *Edit* button and click *Remove*.
- Click *Confirm*. The dialog closes and the role is removed.

IMPORTANT

A scoped role cannot be deleted if users or groups are assigned to it. Remove the role assignments first, and then delete it.

Adding and Removing Users

Adding and removing users to and from scoped roles follows the same process as adding and removing standard roles. To update a user's scoped roles:

- Login to the management console
- Click on the *Access Control* tab
- Click on the *Roles* menu on the left.

- Click on the desired scoped role and choose *Include* or *Exclude* to view the included or excluded members.
- To add a member, click *Add*, choose the member to include or exclude, and click *Save*.
- To remove a member, select the desired member to remove and click *Remove*.

4.3.8. Configuring Constraints

4.3.8.1. Configuring Sensitivity Constraints

Each Sensitivity Constraint defines a set of resources that are considered *sensitive*. A *sensitive* resource is generally one that either should be secret, like passwords, or one that will have serious impact on the server, like networking, JVM configuration, or system properties. The access control system itself is also considered sensitive. Resource sensitivity limits which roles are able to read, write or address a specific resource.

Sensitivity constraint configuration is in the at */core-service=management/access=authorization/constraint=sensitivity-classification*.

Within the management model each Sensitivity Constraint is identified as a classification. The classifications are then grouped into types. There are 39 included classifications that are arranged into 13 types.

To configure a sensitivity constraint, use the *write-attribute* operation to set the *configured-requires-read*, *configured-requires-write*, or *configured-requires-addressable* attribute. To make that type of operation sensitive set the value of the attribute to *true*, otherwise to make it nonsensitive set it to *false*. By default these attributes are not set and the values of *default-requires-read*, *default-requires-write*, and *default-requires-addressable* are used. Once the configured attribute is set it is that value that is used instead of the default. The default values cannot be changed.

Example - Make reading system properties a sensitive operation

```
/core-service=management/access=authorization/constraint=sensitivity-
classification/type=core/classification=system-property:write-
attribute(name=configured-requires-read,value=true)
```

Result

```
/core-service=management/access=authorization/constraint=sensitivity-
classification/type=core/classification=system-property:read-resource
```

```
{
    "outcome" => "success",
    "result" => {
        "configured-requires-addressable" => undefined,
        "configured-requires-read" => true,
        "configured-requires-write" => undefined,
        "default-requires-addressable" => false,
        "default-requires-read" => false,
        "default-requires-write" => true,
        "applies-to" => {
            "/core-service=platform-mbean/type=runtime" => undefined,
            "/system-property=*" => undefined,
```

```
            "/" => undefined
        }
    }
}
```

What roles will be able to perform what operations depending on the configuration of these attributes is summarized in the following table:

Table 4.1. Sensitivity Constraint Configuration Outcomes

Value	requires-read	requires-write	requires-addressable
true	Read is sensitive. Only Auditor, Administrator, SuperUser can read.	Write is sensitive. Only Administrator and SuperUser can write.	Addressing is sensitive. Only Auditor, Administrator, SuperUser can address.
false	Read is not sensitive. Any management user can read.	Write is not sensitive. Only Maintainer, Administrator and SuperUser can write. Deployers can also write the resource is an application resource.	Addressing is not sensitive. Any management user can address.

4.3.8.2. Configure Application Resource Constraints

Each Application Resource Constraint defines a set of resources, attributes and operations that are usually associated with the deployment of applications and services. When an application resource constraint is enabled management users of the *Deployer* role are granted access to the resources that it applies to.

Application constraint configuration is at */core-service=management/access=authorization/constraint=application-classification/*.

Each Application Resource Constraint is identified as a classification. The classifications are then grouped into types. There are 14 included classifications that are arranged into 8 types. Each classification has an *applies-to* element which is a list of resource path patterns to which the classifications configuration applies.

By default the only Application Resource classification that is enabled is core. Core includes deployments, deployment overlays, and the deployment operations.

To enable an Application Resource, use the *write-attribute* operation to set the *configured-application* attribute of the classification to *true*. To disable an Application Resource, set this attribute to *false*. By default these attributes are not set and the value of *default-application* attribute is used. The default value cannot be changed.

Enabling the logger-profile application resource classification

```
/core-service=management/access=authorization/constraint=application-
classification/type=logging/classification=logging-profile:write-
attribute(name=configured-application,value=true)
```

Result

```
/core-service=management/access=authorization/constraint=application-
classification/type=logging/classification=logging-profile:read-resource
```

```
{
    "outcome" => "success",
    "result" => {
        "configured-application" => true,
        "default-application" => false,
        "applies-to" => {"/subsystem=logging/logging-profile=*" =>
undefined}
    }
}
```

IMPORTANT

Application Resource Constraints apply to all resources that match its configuration. For example, it is not possible to grant a *Deployer* user access to one datasource resource but not another. If this level of separation is required then it is recommended to configure the resources in different server groups and create different scoped *Deployer* roles for each group.

4.3.8.3. Configure the Vault Expression Constraint

By default, reading and writing vault expressions are sensitive operations. Configuring the Vault Expression Constraint allows either or both of those operations to be set to nonsensitive. Changing this constraint allows a greater number of roles to read and write vault expressions.

The vault expression constraint is found in the at */core-service=management/access=authorization/constraint=vault-expression*.

To configure the vault expression constraint, use the *write-attribute* operation to set the attributes of *configured-requires-write* and *configured-requires-read* to *true* or *false*. By default these are not set and the values of *default-requires-read* and *default-requires-write* are used. The default values cannot be changed.

Making writing to vault expressions a nonsensitive operation

```
/core-service=management/access=authorization/constraint=vault-
expression:write-attribute(name=configured-requires-write,value=false)
```

Result

```
/core-service=management/access=authorization/constraint=vault-
expression:read-resource
```

```
{
    "outcome" => "success",
    "result" => {
        "configured-requires-read" => undefined,
        "configured-requires-write" => false,
        "default-requires-read" => true,
```

```
        "default-requires-write" => true
    }
}
```

What roles will be able to read and write to vault expressions depending on this configuration is summarized in the following table:

Table 4.2. Vault Expression Constraint Configuration outcomes

Value	requires-read	requires-write
true	Read operation is sensitive. Only Auditor, Administrator, and SuperUser can read.	Write operation is sensitive. Only Administrator, and SuperUser can write.
false	Read operation is not sensitive. All management users can read.	Write operation is not sensitive. Monitor, Administrator, and SuperUser can write. Deployers can also write if the vault expression is in an Application Resource.

4.3.8.4. Application Resource Constraints Reference

Type: core - Classification: deployment-overlay

- default: true
- PATH: /deployment-overlay=*
- PATH: /deployment=*
- PATH: /
- Operation: upload-deployment-stream, full-replace-deployment, upload-deployment-url, upload-deployment-bytes

Type: datasources - Classification: datasource

- default: false
- PATH: /deployment=*/subdeployment=*/subsystem=datasources/data-source=*
- PATH: /subsystem=datasources/data-source=*
- PATH: /subsystem=datasources/data-source=ExampleDS
- PATH: /deployment=*/subsystem=datasources/data-source=*

Type: datasources - Classification: jdbc-driver

- default: false
- PATH: /subsystem=datasources/jdbc-driver=*

Type: datasources - Classification: xa-data-source

- default: false
- PATH: /subsystem=datasources/xa-data-source=*
- PATH: /deployment=*/subsystem=datasources/xa-data-source=*
- PATH: /deployment=*/subdeployment=*/subsystem=datasources/xa-data-source=*

Type: logging - Classification: logger

- default: false
- PATH: /subsystem=logging/logger=*
- PATH: /subsystem=logging/logging-profile=*/logger=*

Type: datasources - Classification: logging-profile

- default: false
- PATH: /subsystem=logging/logging-profile=*

Type: mail - Classification: mail-session

- default: false
- PATH: /subsystem=mail/mail-session=*

Type: naming - Classification: binding

- default: false
- PATH: /subsystem=naming/binding=*

Type: resource-adapters - Classification: resource-adapters

- default: false
- PATH: /subsystem=resource-adapters/resource-adapter=*

Type: security - Classification: security-domain

- default: false
- PATH: /subsystem=security/security-domain=*

4.3.8.5. Sensitivity Constraints Reference

Type: core - Classification: access-control

- requires-addressable: true
- requires-read: true

- requires-write: true
- PATH: /core-service=management/access=authorization
- PATH: /subsystem=jmx ATTRIBUTE: non-core-mbean-sensitivity

Type: core - Classification: credential

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=mail/mail-session=*/server=pop3 ATTRIBUTE: username , password
- PATH: /subsystem=mail/mail-session=*/server=imap ATTRIBUTE: username , password
- PATH: /subsystem=datasources/xa-data-source=* ATTRIBUTE: user-name, recovery-username, password, recovery-password
- PATH: /subsystem=mail/mail-session=*/custom=* ATTRIBUTE: username, password
- PATH: /subsystem=datasources/data-source=*" ATTRIBUTE: user-name, password
- PATH: /subsystem=remoting/remote-outbound-connection=*" ATTRIBUTE: username
- PATH: /subsystem=mail/mail-session=*/server=smtp ATTRIBUTE: username, password
- PATH: /subsystem=resource-adapters/resource-adapter=*/connection-definitions=*" ATTRIBUTE: recovery-username, recovery-password

Type: core - Classification: domain-controller

- requires-addressable: false
- requires-read: false
- requires-write: true

Type: core - Classification: domain-names

- requires-addressable: false
- requires-read: false
- requires-write: true

Type: core - Classification: extensions

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /extension=*

Type: core - Classification: jvm

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=platform-mbean/type=runtime ATTRIBUTE: input-arguments, boot-class-path, class-path, boot-class-path-supported, library-path

Type: core - Classification: management-interfaces

- requires-addressable: false
- requires-read: false
- requires-write: true
- /core-service=management/management-interface=http-interface

Type: core - Classification: module-loading

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=module-loading

Type: core - Classification: patching

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=patching/addon=*
- PATH: /core-service=patching/layer=*"
- PATH: /core-service=patching

Type: core - Classification: read-whole-config

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: / OPERATION: read-config-as-xml

Type: core - Classification: security-domain

- requires-addressable: true
- requires-read: true
- requires-write: true
- PATH: /subsystem=security/security-domain=*

Type: core - Classification: security-domain-ref

- requires-addressable: true
- requires-read: true
- requires-write: true
- PATH: /subsystem=datasources/xa-data-source=* ATTRIBUTE: security-domain
- PATH: /subsystem=datasources/data-source=* ATTRIBUTE: security-domain
- PATH: /subsystem=ejb3 ATTRIBUTE: default-security-domain
- PATH: /subsystem=resource-adapters/resource-adapter=*/connection-definitions=* ATTRIBUTE: security-domain, recovery- security-domain, security-application, security-domain-and-application

Type: core - Classification: security-realm

- requires-addressable: true
- requires-read: true
- requires-write: true
- PATH: /core-service=management/security-realm=*

Type: core - Classification: security-realm-ref

- requires-addressable: true
- requires-read: true
- requires-write: true
- PATH: /subsystem=remoting/connector=* ATTRIBUTE: security-realm
- PATH: /core-service=management/management-interface=native-interface ATTRIBUTE: security-realm
- PATH: /core-service=management/management-interface=http-interface ATTRIBUTE: security-realm
- PATH: /subsystem=remoting/remote-outbound-connection=* ATTRIBUTE: security-realm

Type: core - Classification: security-vault

- requires-addressable: false

- requires-read: false
- requires-write: true
- PATH: /core-service=vault

Type: core - Classification: service-container

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=service-container

Type: core - Classification: snapshots

- requires-addressable: false
- requires-read: false
- requires-write: false
- PATH: / ATTRIBUTE: take-snapshot, list-snapshots, delete-snapshot

Type: core - Classification: socket-binding-ref

- requires-addressable: false
- requires-read: false
- requires-write: false
- PATH: /subsystem=mail/mail-session=*/server=pop3 ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=mail/mail-session=*/server=imap ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=remoting/connector=* ATTRIBUTE: socket-binding
- PATH: /subsystem=remoting/local-outbound-connection=* ATTRIBUTE: outbound-socket-binding-ref
- PATH: /socket-binding-group=*/local-destination-outbound-socket-binding=* ATTRIBUTE: socket-binding-ref
- PATH: /subsystem=remoting/remote-outbound-connection=* ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=mail/mail-session=*/server=smtp ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=transactions ATTRIBUTE: process-id-socket-binding, status-socket-binding, socket-binding

Type: core - Classification: socket-config

- requires-addressable: false

- requires-read: false
- requires-write: true
- PATH: /interface=* OPERATION: resolve-internet-address
- PATH: /socket-binding-group=*
- PATH: /core-service=management/management-interface=http-interface ATTRIBUTE: port, secure-port, interface, secure-socket-binding, socket-binding
- PATH: / OPERATION: resolve-internet-address
- PATH: /subsystem=transactions ATTRIBUTE: process-id-socket-max-ports

Type: core - Classification: system-property

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=platform-mbean/type=runtime ATTRIBUTE: system-properties
- PATH: /system-property=*
- PATH: / OPERATION: resolve-expression

Type: datasources - Classification: data-source-security

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=datasources/xa-data-source=* ATTRIBUTE: user-name, security-domain, password
- PATH: /subsystem=datasources/data-source=* ATTRIBUTE: user-name, security-domain, password

Type: jdr - Classification: jdr

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /subsystem=jdr OPERATION: generate-jdr-report

Type: jmx - Classification: jmx

- requires-addressable: false

- requires-read: false
- requires-write: true
- PATH: /subsystem=jmx

Type: mail - Classification: mail-server-security

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /subsystem=mail/mail-session=*/server=pop3 ATTRIBUTE: username, tls, ssl, password
- PATH: /subsystem=mail/mail-session=*/server=imap ATTRIBUTE: username, tls, ssl, password
- PATH: /subsystem=mail/mail-session=/**custom=** ATTRIBUTE: username, tls, ssl, password
- PATH: /subsystem=mail/mail-session=*/server=smtp ATTRIBUTE: username, tls, ssl, password

Type: naming - Classification: jndi-view

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=naming OPERATION: jndi-view

Type: naming - Classification: naming-binding

- requires-addressable: false
- requires-read: false
- requires-write: false
- PATH: /subsystem=naming/binding=*

Type: remoting - Classification: remoting-security

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=remoting/connector=* ATTRIBUTE: authentication-provider, security-realm
- PATH: /subsystem=remoting/remote-outbound-connection=* ATTRIBUTE: username, security-realm
- PATH: /subsystem=remoting/connector=*/security=sasl

Type: resource-adapters - Classification: resource-adapter-security

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=resource-adapters/resource-adapter=*/connection-definitions=* ATTRIBUTE: security-domain, recovery-username, recovery-security-domain, security-application, security-domain-and-application, recovery-password

Type: security - Classification: misc-security

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=security ATTRIBUTE: deep-copy-subject-mode

CHAPTER 5. JAVA SECURITY MANAGER

5.1. ABOUT THE 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 allowed or denied.

5.2. DEFINE A JAVA SECURITY POLICY

A Java Security policy is 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.

IMPORTANT

Previous versions of JBoss EAP defined policies using an external file, e.g. **`EAP_HOME/bin/server.policy`**. JBoss EAP 7 defines Java Security Policies in two ways: the **`security-manager`** subsystem and through XML files in the individual deployments. The **`security-manager`** subsystem defines minimum and maximum permission for *ALL* deployments, while the XML files specify the permissions requested by the individual deployment.

5.2.1. Defining Policies in the Security Manager Subsystem

The **`security-manager`** subsystem allows you do define shared or common permissions for all deployments. This is accomplished by defining minimum and maximum permission sets. All deployments will be granted at the least all permissions defined in the minimum permission. The deployment process fails for a deployment if it requests a permission that exceeds the ones defined in the maximum permission set.

Example Command Updating Minimum Permission Set

```
/subsystem=security-manager/deployment-permissions=default:write-
attribute(name=minimum-permissions, value=
[{class="java.util.PropertyPermission", actions="read", name="*"}])
```

Example Command Updating Maximum Permission Set

```
/subsystem=security-manager/deployment-permissions=default:write-
attribute(name=maximum-permissions, value=
[{class="java.util.PropertyPermission", actions="read,write", name="*"},
{class="java.io.FilePermission", actions="read,write", name="/-"}])
```

NOTE

If the maximum permission set is not defined, its value defaults to **`java.security.AllPermission`**.

You can find a full reference of the **security-manager** subsystem in the JBoss EAP *Configuration Guide*.

5.2.2. Defining Policies in the Deployment

In JBoss EAP 7, you can add a **META-INF/permissions.xml** to your deployment, which is part of JSR 342 and is a part of the Java EE 7 specification. This file allows you to specify the permissions needed by the deployment. If a minimum permissions set is defined in the **security-manager** subsystem and a **META-INF/permissions.xml** is added to your deployment, then the union of those permissions is granted. If the permissions requested in the **permissions.xml** exceed the maximum policies defined in the **security-manager** subsystem, its deployment will not succeed. If both **META-INF/permissions.xml** and **META-INF/jboss-permissions.xml** are present in the deployment, then only the permissions requested in the **META-INF/jboss-permissions.xml** are granted.

The Java EE 7 specification dictates that **permissions.xml** cover the entire application or top-level deployment module. In cases where you wish to define specific permissions for a subdeployment, you can use the JBoss EAP-specific **META-INF/jboss-permissions.xml**. It follows the same exact format as **permissions.xml** and will apply only to the deployment module in which it is declared.

Sample permissions.xml

```
<permissions version="7">
  <permission>
  <class-name>java.util.PropertyPermission</class-name>
  <name>*</name>
  <actions>read</actions>
  </permission>
</permissions>
```

5.2.3. Defining Policies in Modules

You can restrict the permissions of a module by adding a **<permissions>** element to the **module.xml** file. The **<permissions>** element contains zero or more **<grant>** elements, which define the permission to grant to the module. Each **<grant>** element contains the following attributes:

permission

The qualified class name of the permission to grant.

name

The permission name to provide to the permission class constructor.

actions

The (optional) list of actions, required by some permission types.

Example `module.xml` with Defined Policies

```
<module xmlns="urn:jboss:module:1.5" name="org.jboss.test.example">
  <permissions>
    <grant permission="java.util.PropertyPermission" name="*"
actions="read,write" />
    <grant permission="java.io.FilePermission" name="/etc/-"
actions="read" />
```

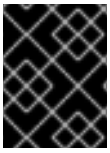
```
    </permissions>
    ...
</module>
```

If the **<permissions>** element is present, the module will be restricted to only the permissions you have listed. If the **<permissions>** element is not present, there will be no restrictions on the module.

5.3. RUN JBOSS EAP WITH THE JAVA SECURITY MANAGER

IMPORTANT

Previous version of JBoss EAP allowed for the use of the **-Djava.security.manager** Java system property as well as custom security managers. Neither of these are supported in JBoss EAP 7. In addition, the Java Security Manager policies are now defined within the **security-manager** subsystem, meaning external policy files and the **-Djava.security.policy** Java system property are not supported JBoss EAP 7.



IMPORTANT

Before starting JBoss EAP with the Java Security Manager enabled, you need make sure all security policies are defined in the **security-manager** subsystem.

To run JBoss EAP with the Java Security Manager, you need to use the **secmgr** option during startup. There are two ways to do this:

- Use the flag with the startup script To use the **-secmgr** flag with the startup script, include it when starting up your JBoss EAP instance:

  Example Startup Script

  ```
  ./standalone.sh -secmgr
  ```

- Using the Startup Configuration File

  IMPORTANT

  The domain or standalone server must be completely stopped before you edit any configuration files.

  NOTE

  If you are using JBoss EAP in a managed domain, you must perform the following procedure on each physical host or instance in your domain.

  To enable the Java Security Manager using the startup configuration file, you need to edit either the **standalone.conf** or **domain.conf** file, depending if you are running a standalone instance or managed domain. If running in Windows, the **standalone.conf.bat** or **domain.conf.bat** files are used instead.

  Uncomment the **SECMGR="true"** line in the configuration file:

  standalone.conf or domain.conf

```
# Uncomment this to run with a security manager enabled
SECMGR="true"
```

standalone.conf.bat or domain.conf.bat

```
rem # Uncomment this to run with a security manager enabled
set "SECMGR=true"
```

5.4. CONSIDERATIONS MOVING FROM PREVIOUS VERSIONS

When moving applications from a previous version of JBoss EAP to JBoss EAP 7 running with the Java Security Manager enabled, you need to be aware of the changes in how policies are defined as well as the necessary configuration needed with both the JBoss EAP configuration and the deployment.

5.4.1. Defining Policies

In previous versions of JBoss EAP, policies were defined in an external configuration file. In JBoss EAP 7, policies are defined using the **`security-manager`** subsystem and with **`permissions.xml`** or **`jboss-permissions.xml`** contained in the deployment. More details on how to use both to define your policies are covered in a previous section.

5.4.2. JBoss EAP Configuration Changes

In previous versions of JBoss EAP, you could use *-Djava.security.manager* and *-Djava.security.policy* Java system properties during JBoss EAP startup. These are no longer supported and the *secmgr* flag should be used instead to enable JBoss EAP to run with the Java Security Manager. More details on the *secmgr* flag are covered in a previous section.

5.4.3. Custom Security Managers

Custom security managers are not supported in JBoss EAP 7.

Revised on 2018-02-08 10:20:50 EST