Red Hat JBoss Enterprise Application Platform 8-beta

Securing applications and management interfaces using multiple identity stores

Guide to securing JBoss EAP management interfaces and deployed applications by combining multiple identity stores such as the filesystem, a database, Lightweight Directory Access Protocol (LDAP), or a custom identity store

Last Updated: 2022-12-16

Red Hat JBoss Enterprise Application Platform 8-beta Securing applications and management interfaces using multiple identity stores

Guide to securing JBoss EAP management interfaces and deployed applications by combining multiple identity stores such as the filesystem, a database, Lightweight Directory Access Protocol (LDAP), or a custom identity store

Legal Notice

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Guide to securing JBoss EAP management interfaces and deployed applications by combining multiple identity stores such as the filesystem, a database, Lightweight Directory Access Protocol (LDAP), or a custom identity store.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 4
MAKING OPEN SOURCE MORE INCLUSIVE 5
CHAPTER 1. CONFIGURING IDENTITY STORES 6
1.1. CREATING AN AGGREGATE REALM 6
1.1.1. Aggregate realm in Elytron 6
1.1.2. Examples of creating security realms required for an aggregate realm 7
1.1.2.1. Creating an ldap-realm in Elytron example 7
1.1.2.2. Creating a filesystem-realm in Elytron example 8
1.1.3. Creating an aggregate-realm in Elytron 10
1.2. CREATING A CACHING REALM 11
1.2.1. Caching realm in Elytron 11
1.2.2. Creating a caching-realm in Elytron 12
1.2.3. Clearing the caching-realm cache 13
1.3. CREATING A DISTRIBUTED REALM 14
1.3.1. Distributed realm in Elytron 14
1.3.2. Examples of creating security realms required for a distributed realm 14
1.3.2.1. Creating an ldap-realm in Elytron example 14
1.3.2.2. Creating a filesystem-realm in Elytron example 16
1.3.3. Creating a distributed-realm in Elytron 18
1.4. CREATING A FAILOVER REALM 19
1.4.1. Failover realm in Elytron 19
1.4.2. Examples of creating security realms required for a failover realm 19
1.4.2.1. Creating an ldap-realm in Elytron example 19
1.4.2.2. Creating a filesystem-realm in Elytron example 21
1.4.3. Creating a failover-realm in Elytron 23
1.5. CREATING A JAAS REALM 24
1.5.1. JAAS realm in Elytron 24
Subject’s principals to attributes mapping and roles association in login modules 24
1.5.2. Developing custom JAAS login modules 25
1.5.2.1. Creating a Maven project for JAAS login module development 25
1.5.2.2. Creating custom JAAS login modules 27
1.5.3. Creating a jaas-realm in Elytron 31
CHAPTER 2. SECURING MANAGEMENT INTERFACES AND APPLICATIONS 33
2.1. ADDING AUTHENTICATION AND AUTHORIZATION TO MANAGEMENT INTERFACES 33
2.2. USING A SECURITY DOMAIN TO AUTHENTICATE AND AUTHORIZE APPLICATION USERS 35
2.2.1. Developing a simple web application for aggregate-realm 35
2.2.1.1. Creating a maven project for web-application development 35
2.2.1.2. Creating a web application 37
2.2.2. Adding authentication and authorization to applications 40
CHAPTER 3. REFERENCE 44
3.1. AGGREGATE-REALM ATTRIBUTES 44
3.2. CACHING-REALM ATTRIBUTES 44
3.3. DISTRIBUTED-REALM ATTRIBUTES 44
3.4. FAILOVER-REALM ATTRIBUTES 45
3.5. HTTP-AUTHENTICATION-FACTORY ATTRIBUTES 45
3.6. JAAS-REALM ATTRIBUTES 46
3.7. MODULE COMMAND ARGUMENTS 47
3.8. SASL-AUTHENTICATION-FACTORY ATTRIBUTES 49

3.9. SECURITY-DOMAIN ATTRIBUTES 50
3.10. SIMPLE-ROLE-DECODER ATTRIBUTES 51

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

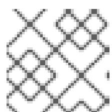
We appreciate your feedback on our documentation. To provide feedback, you can highlight the text in a document and add comments. Follow the steps in the procedure to learn about submitting feedback on Red Hat documentation.

Prerequisites

- Log in to the Red Hat Customer Portal.
- In the Red Hat Customer Portal, view the document in **Multi-page HTML** format.

Procedure

1. Click **Feedback** to see existing reader comments.



NOTE

The feedback feature is enabled only in the **Multi-page HTML** format.

2. Highlight the section of the document where you want to provide feedback.
3. In the prompt menu that displays near the text you selected, click **Add Feedback**.
   A text box opens in the feedback section on the right side of the page.
4. Enter your feedback in the text box and click **Submit**.
   You have created a documentation issue.
5. To view the issue, click the issue tracker link in the feedback view.
6. Highlight the section of the document where you want to provide feedback.
7. In the prompt menu that displays near the text you selected, click **Add Feedback**.
   A text box opens in the feedback section on the right side of the page.
8. Enter your feedback in the text box and click **Submit**.
   You have created a documentation issue.
9. To view the issue, click the issue tracker link in the feedback view.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

CHAPTER 1. CONFIGURING IDENTITY STORES

1.1. CREATING AN AGGREGATE REALM

1.1.1. Aggregate realm in Elytron

With an aggregate realm, **aggregate-realm**, you can use one security realm for authentication and another security realm, or an aggregation of multiple security realms, for authorization in Elytron.

For example, you can configure an **aggregate-realm** to use an **ldap-realm** for authentication and aggregation of a **filesystem-realm** and an **ldap-realm** for authorization.

An identity is created in an aggregate realm configured with multiple authorization realms as follows:

- Attribute values from each authorization realm are loaded.
- If an attribute is defined in more than one authorization realm, the value of the first occurrence of the attribute is used.

The following example illustrates how identity is created when multiple authorization realms contain definitions for the same identity attribute.

Example aggregate realm configuration

```
/subsystem=elytron/aggregate-realm=exampleSecurityRealm:add(authentication-realm=exampleLDAPRealm,authorization-realms=[exampleLDAPRealm,exampleFileSystemRealm])
```

In the example, the configured **aggregate-realm** references two existing security realms: "exampleLDAPRealm", which is an LDAP realm, and "exampleFilesystemRealm", which is a filesystem realm.

- Attribute values obtained from the LDAP realm:

  ```
  mail: administrator@example.com
  telephoneNumber: 0000 0000
  ```

- Attribute values obtained from the filesystem realm:

  ```
  mail: user@example.com
  website: http://www.example.com/
  ```

Resulting identity obtained from the aggregate realm:

```
mail: administrator@example.com
telephoneNumber: 0000 0000
website: http://www.example.com/
```

The example **aggregate-realm** uses the value for the attribute **mail** defined in the LDAP realm because the LDAP realm is referenced before the filesystem realm.

Additional resources

- Creating an **aggregate-realm** in Elytron

1.1.2. Examples of creating security realms required for an aggregate realm

The following examples illustrate creating **ldap-realm** and **filesystem-realm**. You can reference these security realms in an **aggregate-realm**.

1.1.2.1. Creating an ldap-realm in Elytron example

Create an Elytron security realm backed by a Lightweight Directory Access Protocol (LDAP) identity store to secure the JBoss EAP server interfaces or the applications deployed on the server.

For the examples in this procedure, the following LDAP Data Interchange Format (LDIF) is used:

```
dn: ou=Users,dc=wildfly,dc=org
objectClass: organizationalUnit
objectClass: top
ou: Users

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: passwordUser1
mail: administrator@example.com
telephoneNumber: 0000 0000

dn: ou=Roles,dc=wildfly,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=Admin,ou=Roles,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfNames
cn: Admin
member: uid=user1,ou=Users,dc=wildfly,dc=org
```

The LDAP connection parameters used for the example are as follows:

- LDAP URL: **ldap://10.88.0.2**
- LDAP admin password: **secret**
  You need this for Elytron to connect with the LDAP server.
- LDAP admin Distinguished Name (DN): **(cn=admin,dc=wildfly,dc=org)**
- LDAP organization: **wildfly**
  If no organization name is specified, it defaults to **Example Inc**.
- LDAP domain: **wildfly.org**
  This is the name that is matched when the platform receives an LDAP search reference.

Prerequisites

- You have configured an LDAP identity store.
- JBoss EAP is running.

Procedure

1. Configure a directory context that provides the URL and the principal used to connect to the LDAP server.

```
/subsystem=elytron/dir-context=<dir_context_name>:add(url="<LDAP_URL>",principal="<principal_distinguished_name>",credential-reference=<credential_reference>)
```

Example

```
/subsystem=elytron/dir-context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",credential-reference={clear-text="secret"})
{"outcome" => "success"}
```

2. Create an LDAP realm that references the directory context. Specify the Search Base DN and how users are mapped.

Syntax

```
/subsystem=elytron/ldap-realm=<ldap_realm_name>add:(dir-context=<dir_context_name>,identity-mapping=search-base-dn="ou=<organization_unit>,dc=<domain_component>",rdn-identifier="<relative_distinguished_name_identifier>",user-password-mapper={from=<password_attribute_name>},attribute-mapping=[{filter-base-dn="ou=<organization_unit>,dc=<domain_component>",filter="<ldap_filter>",from="<ldap_attribute_name>",to="<identity_attribute_name>"}]})
```

Example

```
/subsystem=elytron/ldap-realm=exampleLDAPRealm:add(dir-context=exampleDirContext,identity-mapping={search-base-dn="ou=Users,dc=wildfly,dc=org",rdn-identifier="uid",user-password-mapper={from="userPassword"},attribute-mapping=[{filter-base-dn="ou=Roles,dc=wildfly,dc=org",filter="(&(objectClass=groupOfNames)(member={1}))",from="cn",to="Roles"},{from="mail",to="mail"},{from="telephoneNumber",to="telephoneNumber"}]})
{"outcome" => "success"}
```

You can now use this realm to create a security domain or to combine with another realm in **failover-realm**, **distributed-realm**, or **aggregate-realm**.

1.1.2.2. Creating a filesystem-realm in Elytron example

Create an Elytron security realm backed by a file system-based identity store to secure the JBoss EAP server interfaces or the applications deployed on the server.

Prerequisites

- JBoss EAP is running.

Procedure

1. Create a **filesystem-realm** in Elytron.

   Syntax

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>)
   ```

   Example

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add(path=fs-realm-users,relative-to=jboss.server.config.dir)
   {"outcome" => "success"}
   ```

2. Add a user to the realm and configure the user's role.

   a. Add a user.

   Syntax

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity(identity=<user_name>)
   ```

   Example

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity(identity=user1)
   {"outcome" => "success"}
   ```

   b. Set roles for the user.

   Syntax

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-attribute(identity=<user_name>,name=<roles_attribute_name>, value=[<role_1>,<role_N>])
   ```

   Example

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-attribute(identity=user1, name=Roles, value=["Admin","Guest"])
   {"outcome" => "success"}
   ```

   c. Set attributes for the user.

   Syntax

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-attribute(identity=<user_name>,name=<attribute_name>, value=[<attribute_value>])
   ```

Example

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-attribute(identity=user1, name=mail, value=["user@example.com"])
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-attribute(identity=user1, name=website, value=["http://www.example.com/"])
```

You can now use this realm to create a security domain or to combine with another realm in **failover-realm**, **distributed-realm**, or **aggregate-realm**.

1.1.3. Creating an aggregate-realm in Elytron

Create an **aggregate-realm** in Elytron that uses one security realm for authentication and aggregation of multiple security realms for authorization. Use the **aggregate-realm** to create a security domain to add authentication and authorization to management interfaces and deployed applications.

Prerequisites

- JBoss EAP is running.
- You have created the realms to reference from the aggregate realm.

Procedure

1. Create an **aggregate-realm** from existing security realms.

   Syntax

   ```
   /subsystem=elytron/aggregate-realm=<aggregate_realm_name>:add(authentication-realm=<security_realm_for_authentication>, authorization-realms=[<security_realm_for_authorization_1>,<security_realm_for_authorization_2>,...,<security_realm_for_authorization_N>])
   ```

   Example

   ```
   /subsystem=elytron/aggregate-realm=exampleSecurityRealm:add(authentication-realm=exampleLDAPRealm,authorization-realms=[exampleLDAPRealm,exampleFileSystemRealm])
   {"outcome" => "success"}
   ```

2. Create a role decoder to map attributes to roles.

   Syntax

   ```
   /subsystem=elytron/simple-role-decoder=<role_decoder_name>:add(attribute=<attribute>)
   ```

   Example

   ```
   /subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
   {"outcome" => "success"}
   ```

3. Create a security domain that references the **aggregate-realm** and the role decoder.

Syntax

```
/subsystem=elytron/security-domain=<security_domain_name>:add(default-realm=<aggregate_realm_name>,permission-mapper=default-permission-mapper,realms=[{realm=<aggregate_realm_name>,role-decoder="<role_decoder_name>"}])
```

Example

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=[{realm=exampleSecurityRealm,role-decoder="from-roles-attribute"}])
{"outcome" => "success"}
```

You now can use the created security domain to add authentication and authorization to management interfaces and applications. For more information, see Securing management interfaces and applications.

Additional resources

- **aggregate-realm** attributes
- **security-domain** attributes
- **simple-role-decoder** attributes

1.2. CREATING A CACHING REALM

1.2.1. Caching realm in Elytron

Elytron provides **caching-realm** to cache the results of a credential lookup from a security realm. The **caching-realm** caches the **PasswordCredential** credential using a *LRU* or *Least Recently Used* caching strategy, in which the least accessed entries are discarded when maximum number of entries is reached.

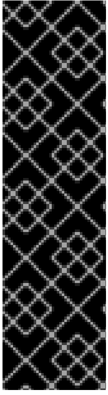
You can use a **caching-realm** with the following security realms:

- **filesystem-realm**
- **jdbc-realm**
- **ldap-realm**
- a custom security realm

If you make changes to your credential source outside of JBoss EAP, those changes are only propagated to a JBoss EAP caching realm if the underlying security realm supports listening. Only **ldap-realm** supports listening. However, filtered attributes, such as **roles**, inside the **ldap-realm** do not support listening.

To ensure that your caching realm has a correct cache of user data, ensure the following:

- Clear the **caching-realm** cache after you modify the user attributes at your credential source.
- Modify your user attributes through the caching realm rather than at your credential source.



IMPORTANT

Making user changes through a caching realm is provided as Technology Preview only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs), might not be functionally complete, and Red Hat does not recommend to use them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

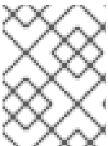
See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

Additional resources

- Creating a **caching-realm** in Elytron
- **caching-realm** attributes
- Clearing the **caching-realm** cache

1.2.2. Creating a caching-realm in Elytron

Create a **caching-realm** and a security domain that references the realm to secure the JBoss EAP server interfaces or the applications deployed on the server.



NOTE

An **ldap-realm** configured as caching realm does not support Active Directory. For more information, see Changing LDAP/AD User Password via JBossEAP CLI for Elytron .

Prerequisites

- You have configured the security realm to cache.

Procedure

1. Create a **caching-realm** that references the security realm to cache.

   Syntax

   ```
   /subsystem=elytron/caching-realm=<caching_realm_name>:add(realm=<realm_to_cache>)
   ```

   Example

   ```
   /subsystem=elytron/caching-realm=exampleSecurityRealm:add(realm=exampleLDAPRealm)
   ```

2. Create a security domain that references the **caching-realm**.

   Syntax

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:add(default-
   realm=<caching_realm_name>,permission-mapper=default-permission-mapper,realms=
   [{realm=<caching_realm_name>,role-decoder="<role_decoder_name>"}])
   ```

Example

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm}])
{"outcome" => "success"}
```

Verification

- To verify that Elytron can load data from the security realm referenced in the **caching-realm** into the **caching-realm**, use the following command:

  #### Syntax

  ```
  /subsystem=elytron/security-domain=<security_domain_name>:read-
  identity(name=<username>)
  ```

  #### Example

  ```
  /subsystem=elytron/security-domain=exampleSecurityDomain:read-identity(name=user1)
  {
      "outcome" => "success",
      "result" => {
          "name" => "user1",
          "attributes" => {"Roles" => ["Admin"]},
          "roles" => ["Admin"]
      }
  }
  ```

You now can use the created security domain to add authentication and authorization to management interfaces and applications. For more information, see Securing management interfaces and applications.

Additional resources

- **caching-realm** attributes
- Clearing the **caching-realm** cache
- **security-domain** attributes

1.2.3. Clearing the caching-realm cache

Clearing a **caching-realm** cache forces Elytron to re-populate the cache by using the latest data from the security realm, which Elytron is configured to cache.

Prerequisites

- A **caching-realm** is configured.

Procedure

- Clear the **caching-realm** cache.

Syntax

```
/subsystem=elytron/caching-realm=<caching_realm_name>:clear-cache
```

Example

```
/subsystem=elytron/caching-realm=exampleSecurityRealm:clear-cache
```

Additional resources

- **caching-realm** attributes

1.3. CREATING A DISTRIBUTED REALM

1.3.1. Distributed realm in Elytron

With a distributed realm, you can search across different identity stores by referencing existing security realms. The identity obtained is used for both authentication and authorization. Elytron invokes the security realms in a distributed realm in the order that you define them in the **distributed-realm** resource.

Example **distributed-realm** configuration

```
/subsystem=elytron/distributed-realm=exampleSecurityRealm:add(realms=
[exampleLDAPRealm,exampleFilesystemRealm])
```

In the example, the configured **distributed-realm** references two existing security realms: "exampleLDAPRealm", which is an LDAP realm, and "exampleFilesystemRealm", which is a filesystem realm. Elytron searches the referenced security realms sequentially as follows:

- Elytron first searches the LDAP realm for a matching identity.
- If Elytron finds a match, the authentication succeeds.
- If Elytron does not find a match, it searches the filesystem realm.

In case the connection to any identity store fails before an identity is matched, the authentication fails with an exception and no more realms are searched. In the example, if the connection to the LDAP server fails, Elytron does not search the next identity store.

Additional resources

- Creating a **distributed-realm** in Elytron

1.3.2. Examples of creating security realms required for a distributed realm

The following examples illustrate creating **ldap-realm** and **filesystem-realm**. You can reference these security realms in a **distributed-realm**.

1.3.2.1. Creating an **ldap-realm** in Elytron example

Create an Elytron security realm backed by a Lightweight Directory Access Protocol (LDAP) identity store to secure the JBoss EAP server interfaces or the applications deployed on the server.

For the examples in this procedure, the following LDAP Data Interchange Format (LDIF) is used:

```
dn: ou=Users,dc=wildfly,dc=org
objectClass: organizationalUnit
objectClass: top
ou: Users

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: userPassword1

dn: ou=Roles,dc=wildfly,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=Admin,ou=Roles,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfNames
cn: Admin
member: uid=user1,ou=Users,dc=wildfly,dc=org
```

The LDAP connection parameters used for the example are as follows:

- LDAP URL: **ldap://10.88.0.2**
- LDAP admin password: **secret**
  You need this for Elytron to connect with the LDAP server.
- LDAP admin Distinguished Name (DN): **(cn=admin,dc=wildfly,dc=org)**
- LDAP organization: **wildfly**
  If no organization name is specified, it defaults to **Example Inc**.
- LDAP domain: **wildfly.org**
  This is the name that is matched when the platform receives an LDAP search reference.

Prerequisites

- You have configured an LDAP identity store.
- JBoss EAP is running.

Procedure

1. Configure a directory context that provides the URL and the principal used to connect to the LDAP server.

```
/subsystem=elytron/dir-
context=<dir_context_name>:add(url="<LDAP_URL>",principal="<principal_distinguished_na
me>",credential-reference=<credential_reference>)
```

Example

```
/subsystem=elytron/dir-
context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",c
redential-reference={clear-text="secret"})
{"outcome" => "success"}
```

2. Create an LDAP realm that references the directory context. Specify the Search Base DN and how users are mapped.

Syntax

```
/subsystem=elytron/ldap-realm=<ldap_realm_name>add:(dir-
context=<dir_context_name>,identity-mapping=search-base-
dn="ou=<organization_unit>,dc=<domain_component>",rdn-
identifier="<relative_distinguished_name_identifier>",user-password-mapper=
{from=<password_attribute_name>},attribute-mapping=[{filter-base-
dn="ou=<organization_unit>,dc=<domain_component>",filter="<ldap_filter>",from="<ldap_attr
ibute_name>",to="<identity_attribute_name>"}]})
```

Example

```
/subsystem=elytron/ldap-realm=exampleLDAPRealm:add(dir-
context=exampleDirContext,identity-mapping={search-base-
dn="ou=Users,dc=wildfly,dc=org",rdn-identifier="uid",user-password-mapper=
{from="userPassword"},attribute-mapping=[{filter-base-
dn="ou=Roles,dc=wildfly,dc=org",filter="(&(objectClass=groupOfNames)(member=
{1}))",from="cn",to="Roles"}]})
{"outcome" => "success"}
```

You can now use this realm to create a security domain or to combine with another realm in **failover-realm**, **distributed-realm** or **aggregate-realm**. You can also configure a **caching-realm** for the **ldap-realm** to cache the result of lookup and improve performance.

1.3.2.2. Creating a filesystem-realm in Elytron example

Create an Elytron security realm backed by a file system-based identity store to secure the JBoss EAP server interfaces or the applications deployed on the server.

Prerequisites

- JBoss EAP is running.

Procedure

1. Create a **filesystem-realm** in Elytron.

Syntax

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>)
```

Example

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add(path=fs-realm-users,relative-to=jboss.server.config.dir)
{"outcome" => "success"}
```

2. Add a user to the realm and configure the user's role.

   a. Add a user.

   #### Syntax

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity(identity=<user_name>)
   ```

   #### Example

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity(identity=user2)
   {"outcome" => "success"}
   ```

   b. Set a password for the user.

   #### Syntax

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:set-password(identity=<user_name>, clear={password=<password>})
   ```

   #### Example

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:set-password(identity=user2, clear={password="passwordUser2"})
   {"outcome" => "success"}
   ```

   c. Set roles for the user.

   #### Syntax

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-attribute(identity=<user_name>, name=<roles_attribute_name>, value=[<role_1>,<role_N>])
   ```

   #### Example

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-attribute(identity=user2, name=Roles, value=["Admin","Guest"])
   {"outcome" => "success"}
   ```

You can now use this realm to create a security domain or to combine with another realm in **failover-realm**, **distributed-realm**, or **aggregate-realm**.

1.3.3. Creating a distributed-realm in Elytron

Create a **distributed-realm** in Elytron that references existing security realms to search for an identity. Use the **distributed-realm** to create a security domain to add authentication and authorization to management interfaces or the applications deployed on the server.

Prerequisites

- JBoss EAP is running.
- You have created the realms to reference in the **distributed-realm**.

Procedure

1. Create a **distributed-realm** referencing existing security realms.

   Syntax

   ```
   /subsystem=elytron/distributed-realm=<distributed_realm_name>:add(realms=
   [<security_realm_1>, <security_realm_2>, ..., <security_realm_N>])
   ```

   Example

   ```
   /subsystem=elytron/distributed-realm=exampleSecurityRealm:add(realms=
   [exampleLDAPRealm, exampleFileSystemRealm])
   {"outcome" => "success"}
   ```

2. Create a role decoder to map attributes to roles.

   Syntax

   ```
   /subsystem=elytron/simple-role-decoder=<role_decoder_name>:add(attribute=<attribute>)
   ```

   Example

   ```
   /subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
   {"outcome" => "success"}
   ```

3. Create a security domain that references the **distributed-realm** and the role decoder.

   Syntax

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:add(realms=
   [{realm=<distributed_realm_name>,role-decoder=<role_decoder_name>}],default-
   realm=<ldap_realm_name>,permission-mapper=<permission_mapper>)
   ```

   Example

   ```
   /subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
   ```

```
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm,role-decoder="from-roles-attribute"}])
{"outcome" => "success"}
```

You now can use the created security domain to add authentication and authorization to management interfaces and applications. For more information, see Securing management interfaces and applications.

Additional resources

- **distributed-realm** attributes
- **security-domain** attributes
- **simple-role-decoder** attributes

1.4. CREATING A FAILOVER REALM

1.4.1. Failover realm in Elytron

You can configure a failover security realm, **failover-realm**, in Elytron that references two existing security realms so that in case one security realm fails, Elytron uses the other as a backup.

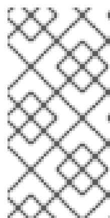
A **failover-realm** in Elytron references two security realms:

- **delegate-realm**: The primary security realm to use.
- **failover-realm**: The security realm to use as the backup.

Example failover-realm configuration

```
/subsystem=elytron/failover-realm=exampleSecurityRealm:add(delegate-
realm=exampleLDAPRealm,failover-realm=exampleFileSystemRealm)
```

In the example, **exampleLDAPRealm**, which is an **ldap-realm**, is used as the delegate realm and **exampleFileSystemRealm**, which is a **filesystem-realm** is used as the **failover-realm**. In the case that the **ldap-realm** fails, Elytron will use the **filesystem-realm** for authentication and authorization.



NOTE

In a **failover-realm**, the **failover-realm** is invoked only when the **delegate-realm** fails. The **fail-over** realm is not invoked if the connection to the **delegate-realm** succeeds but the required identity is not found. To search for identity across multiple security realms, use the **distributed-realm**.

1.4.2. Examples of creating security realms required for a failover realm

The following examples illustrate creating **ldap-realm** and **filesystem-realm**. You can reference these security realms in a **failover-realm**.

1.4.2.1. Creating an ldap-realm in Elytron example

Create an Elytron security realm backed by a Lightweight Directory Access Protocol (LDAP) identity store to secure the JBoss EAP server interfaces or the applications deployed on the server.

For the examples in this procedure, the following LDAP Data Interchange Format (LDIF) is used:

```
dn: ou=Users,dc=wildfly,dc=org
objectClass: organizationalUnit
objectClass: top
ou: Users

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: userPassword1

dn: ou=Roles,dc=wildfly,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=Admin,ou=Roles,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfNames
cn: Admin
member: uid=user1,ou=Users,dc=wildfly,dc=org
```

The LDAP connection parameters used for the example are as follows:

- LDAP URL: **ldap://10.88.0.2**
- LDAP admin password: **secret**
  You need this for Elytron to connect with the LDAP server.
- LDAP admin Distinguished Name (DN): **(cn=admin,dc=wildfly,dc=org)**
- LDAP organization: **wildfly**
  If no organization name is specified, it defaults to **Example Inc**.
- LDAP domain: **wildfly.org**
  This is the name that is matched when the platform receives an LDAP search reference.

Prerequisites

- You have configured an LDAP identity store.
- JBoss EAP is running.

Procedure

1. Configure a directory context that provides the URL and the principal used to connect to the LDAP server.

```
/subsystem=elytron/dir-context=<dir_context_name>:add(url="<LDAP_URL>",principal="<principal_distinguished_name>",credential-reference=<credential_reference>)
```

Example

```
/subsystem=elytron/dir-context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",credential-reference={clear-text="secret"})
{"outcome" => "success"}
```

2. Create an LDAP realm that references the directory context. Specify the Search Base DN and how users are mapped.

Syntax

```
/subsystem=elytron/ldap-realm=<ldap_realm_name>add:(dir-context=<dir_context_name>,identity-mapping=search-base-dn="ou=<organization_unit>,dc=<domain_component>",rdn-identifier="<relative_distinguished_name_identifier>",user-password-mapper={from=<password_attribute_name>},attribute-mapping=[{filter-base-dn="ou=<organization_unit>,dc=<domain_component>",filter="<ldap_filter>",from="<ldap_attribute_name>",to="<identity_attribute_name>"}]})
```

Example

```
/subsystem=elytron/ldap-realm=exampleLDAPRealm:add(dir-context=exampleDirContext,identity-mapping={search-base-dn="ou=Users,dc=wildfly,dc=org",rdn-identifier="uid",user-password-mapper={from="userPassword"},attribute-mapping=[{filter-base-dn="ou=Roles,dc=wildfly,dc=org",filter="(&(objectClass=groupOfNames)(member={1}))",from="cn",to="Roles"}]})
{"outcome" => "success"}
```

You can now use this realm to create a security domain or to combine with another realm in **failover-realm**, **distributed-realm** or **aggregate-realm**. You can also configure a **caching-realm** for the **ldap-realm** to cache the result of lookup and improve performance.

1.4.2.2. Creating a filesystem-realm in Elytron example

Create an Elytron security realm backed by a file system-based identity store to secure the JBoss EAP server interfaces or the applications deployed on the server.

Prerequisites

- JBoss EAP is running.

Procedure

1. Create a **filesystem-realm** in Elytron.

Syntax

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>)
```

Example

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add(path=fs-realm-users,relative-to=jboss.server.config.dir)
{"outcome" => "success"}
```

2. Add a user to the realm and configure the user's role.

    a. Add a user.

    Syntax

    ```
    /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity(identity=<user_name>)
    ```

    Example

    ```
    /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity(identity=user1)
    {"outcome" => "success"}
    ```

    b. Set a password for the user.

    Syntax

    ```
    /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:set-password(identity=<user_name>, clear={password=<password>})
    ```

    Example

    ```
    /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:set-password(identity=user1, clear={password="passwordUser1"})
    {"outcome" => "success"}
    ```

    c. Set roles for the user.

    Syntax

    ```
    /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-attribute(identity=<user_name>,name=<roles_attribute_name>, value=[<role_1>,<role_N>])
    ```

    Example

    ```
    /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-attribute(identity=user1, name=Roles, value=["Admin","Guest"])
    {"outcome" => "success"}
    ```

You can now use this realm to create a security domain or to combine with another realm in **failover-realm**, **distributed-realm**, or **aggregate-realm**.

1.4.3. Creating a failover-realm in Elytron

Create a failover security realm in Elytron that references existing security realms as a delegate realm, the default realm to use, and a failover realm. Elytron uses the configured failover realm in case the delegate realm fails. Use the security realm to create a security domain to add authentication and authorization to management interfaces or the applications deployed on the server.

Prerequisites

- JBoss EAP is running.
- You have created the realms to use as the delegate and failover realm.

Procedure

1. Create a **failover-realm** from existing security realms.

   Syntax

   ```
   /subsystem=elytron/failover-realm=<failover_realm_name>:add(delegate-realm=<realm_to_use_by_default>,failover-realm=<realm_to_use_as_backup>)
   ```

   Example

   ```
   /subsystem=elytron/failover-realm=exampleSecurityRealm:add(delegate-realm=exampleLDAPRealm,failover-realm=exampleFileSystemRealm)
   {"outcome" => "success"}
   ```

2. Create a role decoder to map attributes to roles.

   Syntax

   ```
   /subsystem=elytron/simple-role-decoder=<role_decoder_name>:add(attribute=<attribute>)
   ```

   Example

   ```
   /subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
   {"outcome" => "success"}
   ```

3. Create a security domain that references the **failover-realm** and the role decoder.

   Syntax

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:add(default-realm=<failover_realm_name>,permission-mapper=default-permission-mapper,realms=[{realm=<failover_realm_name>,role-decoder="<role_decoder_name>"}])
   ```

   Example

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=[{realm=exampleSecurityRealm,role-decoder="from-roles-attribute"}])
{"outcome" => "success"}
```

You now can use the created security domain to add authentication and authorization to management interfaces and applications. For more information, see Securing management interfaces and applications.

Additional resources

- **failover-realm** attributes
- **security-domain** attributes
- **simple-role-decoder** attributes

1.5. CREATING A JAAS REALM

1.5.1. JAAS realm in Elytron

The Java Authentication and Authorization Service (JAAS) realm, **jaas-realm**, is a security realm that you can use to configure custom login modules in the **elytron** subsystem for credential verification of users and assigning users roles.

You can use **jaas-realm** for securing both JBoss EAP management interfaces and the deployed applications.

The JAAS realm verifies user credentials by initializing a **javax.security.auth.login.LoginContext**, which uses login modules specified in the JAAS configuration file.

A login module is an implementation of **javax.security.auth.login.LoginContext.LoginModule** interface. Add these implementations as a JBoss EAP module to your server and specify them in the JAAS configuration file.

Example of JAAS configuration file

```
test { (1)
    loginmodules.CustomLoginModule1 optional; (2)
    loginmodules.CustomLoginModule2 optional myOption1=true myOption2=exampleOption; (3)
};
```

(1) Name of the entry that you use when configuring the **jaas-realm**.

(2) Login module with its optional flags. You can use all the flags defined by JAAS. For more information, see JAAS Login Configuration File in the Oracle Java SE documentation.

(3) Login module with its optional flags and options.

Subject's principals to attributes mapping and roles association in login modules

You can add attributes to identities obtained from login modules by utilizing a *subject*'s principals. A *subject* is the user being authenticated and principals are identifiers, such as the user name, contained within a subject.

Elytron obtains and maps identities as follows:

- Login modules use **javax.security.auth.Subject** to represent the user, *subject*, being authenticated.
- A *subject* can have multiple instances of **java.security.Principal**, *principal*, associated with it.
- Elytron uses **org.wildfly.security.auth.server.SecurityIdentity** to represent authenticated users. Elytron maps *subject* to **SecurityIdentity**.

A subject's *principal*s are mapped to security identity's attributes with the following rule:

- The **key** of the attribute is *principal*'s simple class name, obtained by **principal.getClass().getSimpleName()** call.
- The **value** is the *principal*'s name, obtained by **principal.getName()** call.
- For *principal*s of the same type, the values are appended to the collection under the attribute key.

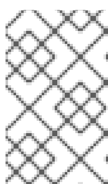
Additional resources

- Developing custom JAAS login modules
- Creating a **jaas-realm** in Elytron
- **Subject** class Javadocs
- **Principal** class Javadocs

1.5.2. Developing custom JAAS login modules

You can create custom Java Authentication and Authorization Service (JAAS) login modules to implement custom authentication and authorization functionality.

You can use the custom JAAS login modules through the **jaas-realm** in the Elytron subsystem to secure JBoss EAP management interfaces and deployed applications. The login modules are not part of a deployment, you include them as JBoss EAP modules.



NOTE

The following procedures are provided as an example only. If you already have custom login modules, you can skip these and proceed to creating a **jaas-realm**. For more information, see Creating a **jaas-realm** in Elytron.

1.5.2.1. Creating a Maven project for JAAS login module development

For creating custom Java Authentication and Authorization Service (JAAS) login modules, create a Maven project with the required dependencies and directory structure.

Prerequisites

- You have installed Maven. For more information, see Downloading Apache Maven.

Procedure

1. Use the **mvn** command in the CLI to set up a Maven project. This command creates the directory structure for the project and the **pom.xml** configuration file.

   Syntax

   ```
   $ mvn archetype:generate \
   -DgroupId=<group-to-which-your-application-belongs> \
   -DartifactId=<name-of-your-application> \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-simple \
   -DinteractiveMode=false
   ```

   Example

   ```
   $ mvn archetype:generate \
   -DgroupId=com.example.loginmodule \
   -DartifactId=example-custom-login-module \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-simple \
   -DinteractiveMode=false
   ```

2. Navigate to the application root directory.

   Syntax

   ```
   $ cd <name-of-your-application>
   ```

   Example

   ```
   $ cd example-custom-login-module
   ```

3. Replace the content of the generated **pom.xml** file with the following text:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <project xmlns="http://maven.apache.org/POM/4.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
       <modelVersion>4.0.0</modelVersion>

       <groupId>custom.loginmodules</groupId>
       <artifactId>custom-login-modules</artifactId>
       <version>1.0</version>
       <dependencies>
           <dependency>
               <groupId>org.wildfly.security</groupId>
               <artifactId>wildfly-elytron</artifactId>
               <version>1.17.2.Final</version>
           </dependency>
           <dependency>
               <groupId>javax.security.enterprise</groupId>
               <artifactId>javax.security.enterprise-api</artifactId>
               <version>1.0</version>
           </dependency>
   ```

```
    </dependencies>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

</project>
```

4. Remove the directories **site** and **test** because they are not required for this example.

```
$ rm -rf src/site/
$ rm -rf src/test/
```

Verification

- In the application root directory, enter the following command:

```
$ mvn install
```

You get an output similar to the following:

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.404 s
[INFO] Finished at: 2022-04-28T13:55:18+05:30
[INFO] ------------------------------------------------------------------------
```

You can now create custom JAAS login modules.

1.5.2.2. Creating custom JAAS login modules

Create a custom Java Authentication and Authorization Service (JAAS) login module by creating a class that implements the **javax.security.auth.spi.LoginModule** interface. Additionally, create JAAS configuration file with flags and options for the custom login module.

In this procedure, *<application_home>* refers to the directory that contains the **pom.xml** configuration file for the application.

Prerequisites

- You have created a Maven project.
  For more information, see Creating a Maven project for JAAS login module development .

Procedure

1. Create a directory to store the Java files.

Syntax

```
$ mkdir -p src/main/java/<path_based_on_artifactID>
```

Example

```
$ mkdir -p src/main/java/com/example/loginmodule
```

2. Navigate to the directory containing the source files.

Syntax

```
$ cd src/main/java/<path_based_on_groupID>
```

Example

```
$ cd src/main/java/com/example/loginmodule
```

3. Delete the generated file **App.java**.

```
$ rm App.java
```

4. Create a file **ExampleCustomLoginModule.java** for custom login module source.

```
package com.example.loginmodule;

import org.wildfly.security.auth.principal.NamePrincipal;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import java.io.IOException;
import java.security.Principal;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;


public class ExampleCustomLoginModule implements LoginModule {

    private final Map<String, char[]> usersMap = new HashMap<String, char[]>();
    private Principal principal;
    private Subject subject;
    private CallbackHandler handler;

    /**
     * In this example, identities are created as fixed Strings.
     *
     * The identities are:
     *     user1 has the password passwordUser1
     *     user2 has the password passwordUser2
     *
     * Use these credentials when you secure management interfaces
```

```
     * or applications with this login module.
     *
     * In a production login module, you would get the identities
     * from a data source.
     *
     */

    @Override
    public void initialize(Subject subject, CallbackHandler callbackHandler, Map<String, ?>
sharedState, Map<String, ?> options) {
        this.subject = subject;
        this.handler = callbackHandler;
        this.usersMap.put("user1", "passwordUser1".toCharArray());
        this.usersMap.put("user2", "passwordUser2".toCharArray());
    }

    @Override
    public boolean login() throws LoginException {
        // obtain the incoming username and password from the callback handler
        NameCallback nameCallback = new NameCallback("Username");
        PasswordCallback passwordCallback = new PasswordCallback("Password", false);
        Callback[] callbacks = new Callback[]{nameCallback, passwordCallback};
        try {
            this.handler.handle(callbacks);
        } catch (UnsupportedCallbackException | IOException e) {
            throw new LoginException("Error handling callback: " + e.getMessage());
        }

        final String username = nameCallback.getName();
        this.principal = new NamePrincipal(username);
        final char[] password = passwordCallback.getPassword();

        char[] storedPassword = this.usersMap.get(username);
        if (!Arrays.equals(storedPassword, password)) {
            throw new LoginException("Invalid password");
        } else {
            return true;
        }
    }

    /**
     * user1 is assigned the roles Admin, User and Guest.
     * In a production login module, you would get the identities
     * from a data source.
     *
     */

    @Override
    public boolean commit() throws LoginException {
        if (this.principal.getName().equals("user1")) {
            this.subject.getPrincipals().add(new Roles("Admin"));
            this.subject.getPrincipals().add(new Roles("User"));
            this.subject.getPrincipals().add(new Roles("Guest"));
        }
        return true;
    }
```

```
    @Override
    public boolean abort() throws LoginException {
        return true;
    }

    @Override
    public boolean logout() throws LoginException {
        this.subject.getPrincipals().clear();
        return true;
    }

    /**
     * Principal with simple classname 'Roles' will be mapped to the identity's attribute with
name 'Roles'.
     */

    private static class Roles implements Principal {

        private final String name;

        Roles(final String name) {
            this.name = name;
        }

        /**
         * @return name of the principal. This will be added as a value to the identity's attribute
which has a name equal to the simple name of this class. In this example, this value will be
added to the attribute with a name 'Roles'.
         */

        public String getName() {
            return this.name;
        }
    }
}
```

5. In the *<application_home>* directory, create JAAS configuration file **JAAS-login-modules.conf**.

```
exampleConfiguration {
 com.example.loginmodule.ExampleCustomLoginModule optional;
};
```

- **exampleConfiguration** is the Entry name.
- **com.example.loginmodule.ExampleCustomLoginModule** is the login module.
- **optional** is the flag.

6. Compile the login module.

```
$ mvn package
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

```
[INFO] Total time: 1.321 s
[INFO] Finished at: 2022-04-28T14:16:03+05:30
[INFO] ------------------------------------------------------------------------
```

You can now use the login module to secure JBoss EAP management interfaces and deployed applications.

1.5.3. Creating a **jaas-realm** in Elytron

Create an Elytron security realm backed by Java Authentication and Authorization Service (JAAS)-compatible custom login module to secure JBoss EAP server interfaces or deployed applications. Use the security realm to create a security domain.

Prerequisites

- You have packaged custom login modules as JAR.
  For an example login module, see Developing custom JAAS login modules .
- JBoss EAP is running.

Procedure

1. Add the login module JAR to JBoss EAP as a module using the management CLI.

   Syntax

   ```
   module add --name=<name_of_the_login_moudle> --resources=<path_to_the_login_module_jar> --dependencies=org.wildfly.security.elytron
   ```

   Example

   ```
   module add --name=exampleLoginModule  --resources=<path_to_login_module>/custom-login-modules-1.0.jar --dependencies=org.wildfly.security.elytron
   ```

2. Create **jaas-realm** from the login module and the JAAS login configuration file.

   Syntax

   ```
   /subsystem=elytron/jaas-realm=<jaas_realm_name>:add(entry=<entry-name>,path=<path_to_module_config_file>,module=<name_of_the_login_module>,callback-handler=<name_of_the_optional_callback_handler>)
   ```

   Example

   ```
   /subsystem=elytron/jaas-realm=exampleSecurityRealm:add(entry=exampleConfiguration,path=<path_to_login_module>/JAAS-login-modules.conf,module=exampleLoginModule)
   ```

3. Create a security domain that references the **jaas-realm**.

   Syntax

```
/subsystem=elytron/security-domain=<security_domain_name>:add(default-realm=<jaas_realm_name>,realms=[{realm=<jaas_realm_name>}],permission-mapper=default-permission-mapper)
```

Example

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-realm=exampleSecurityRealm,realms=[{realm=exampleSecurityRealm}],permission-mapper=default-permission-mapper)
{"outcome" => "success"}
```

You now can use the created security domain to add authentication and authorization to management interfaces and applications. For more information, see Securing management interfaces and applications.

Additional resources

- **module** command arguments
- **jaas-realm** attributes
- **security-domain** attributes

CHAPTER 2. SECURING MANAGEMENT INTERFACES AND APPLICATIONS

2.1. ADDING AUTHENTICATION AND AUTHORIZATION TO MANAGEMENT INTERFACES

You can add authentication and authorization for management interfaces to secure them by using a security domain. To access the management interfaces after you add authentication and authorization, users must enter login credentials.

You can secure JBoss EAP management interfaces as follows:

- Management CLI
  By configuring a **sasl-authentication-factory**.
- Management console
  By configuring an **http-authentication-factory**.

Prerequisites

- You have created a security domain referencing a security realm.
- JBoss EAP is running.

Procedure

1. Create an **http-authentication-factory**, or a **sasl-authentication-factory**.
   - Create an **http-authentication-factory**.

     Syntax

     ```
     /subsystem=elytron/http-authentication-factory=<authentication_factory_name>:add(http-server-mechanism-factory=global, security-domain=<security_domain_name>, mechanism-configurations=[{mechanism-name=<mechanism-name>, mechanism-realm-configurations=[{realm-name=<realm_name>}]}])
     ```

     Example

     ```
     /subsystem=elytron/http-authentication-factory=exampleAuthenticationFactory:add(http-server-mechanism-factory=global, security-domain=exampleSecurityDomain, mechanism-configurations=[{mechanism-name=BASIC, mechanism-realm-configurations=[{realm-name=exampleSecurityRealm}]}])
     {"outcome" => "success"}
     ```

   - Create a **sasl-authentication-factory**.

     Syntax

     ```
     /subsystem=elytron/sasl-authentication-factory=<sasl_authentication_factory_name>:add(security-domain=<security_domain>,sasl-server-factory=configured,mechanism-configurations=
     ```

```
[{mechanism-name=<mechanism-name>,mechanism-realm-configurations=[{realm-
name=<realm_name>}]}])
```

Example

```
/subsystem=elytron/sasl-authentication-
factory=exampleSaslAuthenticationFactory:add(security-
domain=exampleSecurityDomain,sasl-server-factory=configured,mechanism-
configurations=[{mechanism-name=PLAIN,mechanism-realm-configurations=[{realm-
name=exampleSecurityRealm}]}])
{"outcome" => "success"}
```

2. Update the management interfaces.

   - Use the **http-authentication-factory** to secure the management console.

     Syntax

     ```
     /core-service=management/management-interface=http-interface:write-
     attribute(name=http-authentication-factory, value=<authentication_factory_name>)
     ```

     Example

     ```
     /core-service=management/management-interface=http-interface:write-
     attribute(name=http-authentication-factory, value=exampleAuthenticationFactory)
     {
         "outcome" => "success",
         "response-headers" => {
             "operation-requires-reload" => true,
             "process-state" => "reload-required"
         }
     }
     ```

   - Use the **sasl-authentication-factory** to secure the management CLI.

     Syntax

     ```
     /core-service=management/management-interface=http-interface:write-
     attribute(name=http-upgrade,value={enabled=true,sasl-authentication-
     factory=<sasl_authentication_factory>})
     ```

     Example

     ```
     /core-service=management/management-interface=http-interface:write-
     attribute(name=http-upgrade,value={enabled=true,sasl-authentication-
     factory=exampleSaslAuthenticationFactory})
     {
         "outcome" => "success",
         "response-headers" => {
             "operation-requires-reload" => true,
             "process-state" => "reload-required"
         }
     }
     ```

3. Reload the server.

```
reload
```

Verification

- To verify that the management console requires authentication and authorization, navigate to the management console at **http://127.0.0.1:9990/console/index.html**.
  You are prompted to enter user name and password.

- To verify that the management CLI requires authentication and authorization, start the management CLI using the following command:

  ```
  $ bin/jboss-cli.sh --connect
  ```

  You are prompted to enter user name and password.

Additional resources

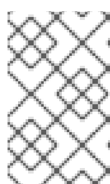
- **http-authentication-factory** attributes
- **sasl-authentication-factory** attributes

2.2. USING A SECURITY DOMAIN TO AUTHENTICATE AND AUTHORIZE APPLICATION USERS

Use a security domain that references a security realm to authenticate and authorize application users. The procedures for developing an application are provided only as an example.

2.2.1. Developing a simple web application for aggregate-realm

You can create a simple web application to follow along with the configuring security realms examples.



NOTE

The following procedures are provided as an example only. If you already have an application that you want to secure, you can skip these and go directly to Adding authentication and authorization to applications.

2.2.1.1. Creating a maven project for web-application development

For creating a web-application, create a Maven project with the required dependencies and the directory structure.

Prerequisites

- You have installed Maven. For more information, see Downloading Apache Maven.

Procedure

1. Set up a Maven project using the **mvn** command. The command creates the directory structure for the project and the **pom.xml** configuration file.

Syntax

```
$ mvn archetype:generate \
-DgroupId=${group-to-which-your-application-belongs} \
-DartifactId=${name-of-your-application} \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

Example

```
$ mvn archetype:generate \
-DgroupId=com.example.app \
-DartifactId=simple-webapp-example \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

2. Navigate to the application root directory:

Syntax

```
$ cd <name-of-your-application>
```

Example

```
$ cd simple-webapp-example
```

3. Replace the content of the generated **pom.xml** file with the following text:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>
```

```
  <dependencies>
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.0.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.wildfly.security</groupId>
      <artifactId>wildfly-elytron-auth-server</artifactId>
      <version>1.19.0.Final</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <plugin>
        <groupId>org.wildfly.plugins</groupId>
        <artifactId>wildfly-maven-plugin</artifactId>
        <version>2.1.0.Final</version>
      </plugin>
    </plugins>
  </build>

</project>
```

Verification

- In the application root directory, enter the following command:

```
$ mvn install
```

You get an output similar to the following:

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.795 s
[INFO] Finished at: 2022-04-28T17:39:48+05:30
[INFO] ------------------------------------------------------------------------
```

You can now create a web-application.

2.2.1.2. Creating a web application

Create a web application containing a servlet that returns the user name obtained from the logged-in user's principal and attributes. If there is no logged-in user, the servlet returns the text "NO AUTHENTICATED USER".

Prerequisites

- You have created a Maven project.

- JBoss EAP is running.

Procedure

1. Create a directory to store the Java files.

   Syntax

   ```
   $ mkdir -p src/main/java/<path_based_on_artifactID>
   ```

   Example

   ```
   $ mkdir -p src/main/java/com/example/app
   ```

2. Navigate to the new directory.

   Syntax

   ```
   $ cd src/main/java/<path_based_on_artifactID>
   ```

   Example

   ```
   $ cd src/main/java/com/example/app
   ```

3. Create a file **SecuredServlet.java** with the following content:

   ```
   package com.example.app;

   import java.io.IOException;
   import java.io.PrintWriter;
   import java.security.Principal;
   import java.util.ArrayList;
   import java.util.Collection;
   import java.util.Iterator;
   import java.util.List;
   import java.util.Set;

   import jakarta.servlet.ServletException;
   import jakarta.servlet.annotation.WebServlet;
   import jakarta.servlet.http.HttpServlet;
   import jakarta.servlet.http.HttpServletRequest;
   import jakarta.servlet.http.HttpServletResponse;
   import org.wildfly.security.auth.server.SecurityDomain;
   import org.wildfly.security.auth.server.SecurityIdentity;
   import org.wildfly.security.authz.Attributes;
   import org.wildfly.security.authz.Attributes.Entry;
   /**
    * A simple secured HTTP servlet. It returns the user name and
    * attributes obtained from the logged-in user's Principal. If
    * there is no logged-in user, it returns the text
    * "NO AUTHENTICATED USER".
    */
   ```

```
@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

            Principal user = req.getUserPrincipal();
            SecurityIdentity identity = SecurityDomain.getCurrent().getCurrentSecurityIdentity();
            Attributes identityAttributes = identity.getAttributes();
            Set <String> keys = identityAttributes.keySet();
            String attributes = "<ul>";

            for (String attr : keys) {
             attributes += "<li> " +  attr + " : " + identityAttributes.get(attr).toString() + "</li>";
            }

            attributes+="</ul>";
            writer.println("<html>");
            writer.println("  <head><title>Secured Servlet</title></head>");
            writer.println("  <body>");
            writer.println("    <h1>Secured Servlet</h1>");
            writer.println("    <p>");
            writer.print(" Current Principal '");
            writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
            writer.print("'");
            writer.print(user != null ? "\n" + attributes : "");
            writer.println("    </p>");
            writer.println("  </body>");
            writer.println("</html>");
        }
    }

}
```

4. In the application root directory, compile your application with the following command:

```
$ mvn package
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.015 s
[INFO] Finished at: 2022-04-28T17:48:53+05:30
[INFO] ------------------------------------------------------------------------
```

5. Deploy the application.

```
$ mvn wildfly:deploy
```

Verification

- In a browser, navigate to **http://localhost:8080/simple-webapp-example/secured**.
  You get the following message:

```
Secured Servlet
Current Principal 'NO AUTHENTICATED USER'
```

Because no authentication mechanism is added, you can access the application.

You can now secure this application by using a security domain so that only authenticated users can access it.

2.2.2. Adding authentication and authorization to applications

You can add authentication and authorization to web applications to secure them by using a security domain. To access the web applications after you add authentication and authorization, users must enter login credentials.

Prerequisites

- You have created a security domain referencing a security realm.
- You have deployed applications on JBoss EAP.
- JBoss EAP is running.

Procedure

1. Configure an **application-security-domain** in the **undertow subsystem**:

   Syntax

   ```
   /subsystem=undertow/application-security-
   domain=<application_security_domain_name>:add(security-
   domain=<security_domain_name>)
   ```

   Example

   ```
   /subsystem=undertow/application-security-
   domain=exampleApplicationSecurityDomain:add(security-domain=exampleSecurityDomain)
   {"outcome" => "success"}
   ```

2. Configure the application’s **web.xml** to protect the application resources.

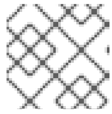
   Syntax

   ```
   <!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd" >

   <web-app>

    <!-- Define the security constraints for the application resources.
        Specify the URL pattern for which a challenge is -->

    <security-constraint>
           <web-resource-collection>
               <web-resource-name><!-- Name of the resources to protect --></web-resource-name>
   ```

```
        <url-pattern> <!-- The URL to protect  --></url-pattern>
      </web-resource-collection>

      <!-- Define the role that can access the protected resource -->
      <auth-constraint>
        <role-name> <!-- Role name as defined in the security domain --></role-name>
        <!-- To disable authentication you can use the wildcard *
          To authenticate but allow any role, use the wildcard **. -->
      </auth-constraint>
    </security-constraint>

    <login-config>
      <auth-method>
       <!-- The authentication method to use. Can be:
        BASIC
        CLIENT-CERT
        DIGEST
        FORM
        SPNEGO
        -->
      </auth-method>

      <realm-name><!-- The name of realm to send in the challenge  --></realm-name>
    </login-config>
  </web-app>
```

Example

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>

 <!-- Define the security constraints for the application resources.
     Specify the URL pattern for which a challenge is -->

 <security-constraint>
      <web-resource-collection>
        <web-resource-name>all</web-resource-name>
        <url-pattern>/*</url-pattern>
      </web-resource-collection>

      <!-- Define the role that can access the protected resource -->
      <auth-constraint>
        <role-name>Admin</role-name>
        <!-- To disable authentication you can use the wildcard *
          To authenticate but allow any role, use the wildcard **. -->
      </auth-constraint>
    </security-constraint>

    <login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>exampleSecurityRealm</realm-name>
    </login-config>
  </web-app>
```



NOTE

You can use a different **auth-method**.

3. Configure your application to use a security domain by either creating a **jboss-web.xml** file in your application or setting the default security domain in the **undertow** subsystem.

   - Create **jboss-web.xml** file in the your application's **WEB-INF** directory referencing the **application-security-domain**.

     Syntax

     ```
     <jboss-web>
       <security-domain> <!-- The security domain to associate with the application --
     ></security-domain>
     </jboss-web>
     ```

     Example

     ```
     <jboss-web>
       <security-domain>exampleApplicationSecurityDomain</security-domain>
     </jboss-web>
     ```

   - Set the default security domain in the **undertow** subsystem for applications.

     Syntax

     ```
     /subsystem=undertow:write-attribute(name=default-security-
     domain,value=<application_security_domain_to_use>)
     ```

     Example

     ```
     /subsystem=undertow:write-attribute(name=default-security-
     domain,value=exampleApplicationSecurityDomain)
     {
         "outcome" => "success",
         "response-headers" => {
             "operation-requires-reload" => true,
             "process-state" => "reload-required"
         }
     }
     ```

4. Reload the server.

   ```
   reload
   ```

Verification

1. In the application root directory, compile your application with the following command:

   ```
   $ mvn package
   ```

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.015 s
[INFO] Finished at: 2022-04-28T17:48:53+05:30
[INFO] ------------------------------------------------------------------------
```

2. Deploy the application.

```
$ mvn wildfly:deploy
```

3. In a browser, navigate to **http://localhost:8080/simple-webapp-example/secured**. You get a login prompt confirming that authentication is now required to access the application.

Your application is now secured with a security domain and users can log in only after authenticating. Additionally, only users with specified roles can access the application.

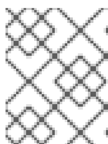
CHAPTER 3. REFERENCE

3.1. AGGREGATE-REALM ATTRIBUTES

You can configure **aggregate-realm** by setting its attributes.

Table 3.1. aggregate-realm sttributes

Attribute	Description
authentication-realm	Reference to the security realm to use for authentication steps. This is used for obtaining or validating credentials.
authorization-realm	Reference to the security realm to use for loading the identity for authorization steps.
authorization-realms	Reference to the security realms to aggregate for loading the identity for authorization steps. If an attribute is defined in more than one authorization realm, the value of the first occurrence of the attribute is used.
principal-transformer	Reference to a principal transformer to apply between loading the identity for authentication and loading the identity for authorization.



NOTE

The **authorization-realm** and **authorization-realms** attributes are mutually exclusive. Define only one of the two attributes in a realm.

3.2. CACHING-REALM ATTRIBUTES

You can configure **caching-realm** by setting its attributes.

Table 3.2. caching-realm Attributes

Attribute	Description
maximum-age	The time in milliseconds that an item can stay in the cache. A value of **-1** keeps items indefinitely. This defaults to**-1**.
maximum-entries	The maximum number of entries to keep in the cache. This defaults to **16**.
realm	A reference to a cacheable security realm such as **jdbc-realm**, **ldap-realm**, **filesystem-realm** or a custom security realm.

3.3. DISTRIBUTED-REALM ATTRIBUTES

You can configure **distributed-realm** by setting its attributes.

Table 3.3. distributed-realm attributes

Attribute	Description
realms	A list of the security realms to search. The security realms are invoked sequentially in the order they are provided in this attribute.

3.4. FAILOVER-REALM ATTRIBUTES

You can configure **failover-realm** by setting its attributes.

Table 3.4. failover-realm attributes

Attribute	Description
delegate-realm	The security realm to use by default.
emit-events	Specifies whether a security event of the type **SecurityEvent** that signifies the unavailability of a **delegate-realm** should be emitted. When enabled, you can capture these events in the audit log. The default values is **true**.
failover-realm	The security realm to use in case the **delegate-realm** is unavailable.

3.5. HTTP-AUTHENTICATION-FACTORY ATTRIBUTES

You can configure **http-authentication-factory** by setting its attributes.

Table 3.5. http-authentication-factory attributes

Attribute	Description
http-server-mechanism-factory	The **HttpServerAuthenticationMechanismFactory** to associate with this resource.
mechanism-configurations	The list of mechanism-specific configurations.
security-domain	The security domain to associate with the resource.

Table 3.6. http-authentication-factory mechanism-configurations attributes

Attribute	Description

Attribute	Description
credential-security-factory	The security factory to use to obtain a credential as required by the mechanism.
final-principal-transformer	A final principal transformer to apply for this mechanism realm.
host-name	The host name this configuration applies to.
mechanism-name	This configuration will only apply where a mechanism with the name specified is used. If this attribute is omitted then this will match any mechanism name.
mechanism-realm-configurations	The list of definitions of the realm names as understood by the mechanism.
pre-realm-principal-transformer	A principal transformer to apply before the realm is selected.
post-realm-principal-transformer	A principal transformer to apply after the realm is selected.
protocol	The protocol this configuration applies to.
realm-mapper	The realm mapper to be used by the mechanism.

Table 3.7. http-authentication-factory mechanism-configurations mechanism-realm-configurations attributes

Attribute	Description
final-principal-transformer	A final principal transformer to apply for this mechanism realm.
post-realm-principal-transformer	A principal transformer to apply after the realm is selected.
pre-realm-principal-transformer	A principal transformer to apply before the realm is selected.
realm-mapper	The realm mapper to be used by the mechanism.
realm-name	The name of the realm to be presented by the mechanism.

3.6. JAAS-REALM ATTRIBUTES

You can configure **jaas-realm** by setting its attributes. All the attributes except **entry** are optional.

Table 3.8. jaas-realm attributes

attribute	description
callback-handler	Callback handler to use with the Login Context. Security property **auth.login.defaultCallbackHandler** can be used instead. The default callback handler of the realm is used if none of these are defined.
entry	The entry name to use to initialize **LoginContext**.
module	The module with custom **LoginModules** and **CallbackHandler** classes.
path	The optional path to JAAS configuration file. You can also specify the location with java system property **java.security.auth.login.config** or with java security property **login.config.url**.
relative-to	If you provide **relative-to**, the value of the**path** attribute is treated as relative to the path specified by this attribute.

3.7. MODULE COMMAND ARGUMENTS

You can use different arguments with the **module** command.

Table 3.9. module command arguments

Argument	Description
--absolute-resources	Use this argument to specify a list of absolute file system paths to reference from its **module.xml** file. The files specified are not copied to the module directory. See **--resource-delimiter** for delimiter details.
--allow-nonexistent-resources	Use this argument to create empty directories for resources specified by **--resources** that do not exist. The**module add** command will fail if there are resources that do not exist and this argument is not used.
--dependencies	Use this argument to provide a comma-separated list of module names that this module depends on.
--export-dependencies	Use this argument to specify exported dependencies. `module add --name=com.mysql --resources=/path/to/{MySQLDriverJarName} --export-dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api`

Argument	Description
--main-class	Use this argument to specify the fully qualified class name that declares the module's main method.
--module-root-dir	Use this argument if you have defined an external JBoss EAP module directory to use instead of the default ***EAP_HOME*/modules/** directory. `module add --module-root-dir=/path/to/my-external-modules/ --name=com.mysql --resources=/path/to/{MySQLDriverJarName} --dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api`
--module-xml	Use this argument to provide a file system path to a **module.xml** to use for this new module. This file is copied to the module directory. If this argument is not specified, a **module.xml** file is generated in the module directory.
--name	Use this argument to provide the name of the module to add. This argument is required.
--properties	Use this argument to provide a comma-separated list of ***PROPERTY_NAME*=*PROPERTY_VALUE*** pairs that define module properties.
--resource-delimiter	Use this argument to set a user-defined file path separator for the list of resources provided to the **--resources** or **absolute-resources** argument. If not set, the file path separator is a colon (**:**) for Linux and a semicolon (**;**) for Windows.
--resources	Use this argument to specify the resources for this module by providing a list of file system paths. The files are copied to this module directory and referenced from its **module.xml** file. If you a provide a path to a directory, the directory and its contents are copied to the module directory. Symbolic links are not preserved; linked resources are copied to the module directory. This argument is required unless **--absolute-resources** or **--module-xml** is provided. See **--resource-delimiter** for delimiter details.

Argument	Description
--slot	Use this argument to add the module to a slot other than the default **main** slot. `module add --name=com.mysql --slot=8.0 --resources=/path/to/{MySQLDriverJarName} --dependencies=javaee.api,sun.jdk,ibm.jdk,javax.api,javax.transaction.api`

3.8. SASL-AUTHENTICATION-FACTORY ATTRIBUTES

You can configure **sasl-authentication-factory** by setting its attributes.

Table 3.10. sasl-authentication-factory attributes

Attribute	Description
mechanism-configurations	The list of mechanism specific configurations.
sasl-server-factory	The SASL server factory to associate with this resource.
security-domain	The security domain to associate with this resource.

Table 3.11. sasl-authentication-factory mechanism-configurations attributes

Attribute	Description
credential-security-factory	The security factory to use to obtain a credential as required by the mechanism.
final-principal-transformer	A final principal transformer to apply for this mechanism realm.
host-name	The host name this configuration applies to.
mechanism-name	This configuration will only apply where a mechanism with the name specified is used. If this attribute is omitted then this will match any mechanism name.
mechanism-realm-configurations	The list of definitions of the realm names as understood by the mechanism.
protocol	The protocol this configuration applies to.
post-realm-principal-transformer	A principal transformer to apply after the realm is selected.
pre-realm-principal-transformer	A principal transformer to apply before the realm is selected.

Attribute	Description
realm-mapper	The realm mapper to be used by the mechanism.

Table 3.12. sasl-authentication-factory mechanism-configurations mechanism-realm-configurations attributes

Attribute	Description
final-principal-transformer	A final principal transformer to apply for this mechanism realm.
post-realm-principal-transformer	A principal transformer to apply after the realm is selected.
pre-realm-principal-transformer	A principal transformer to apply before the realm is selected.
realm-mapper	The realm mapper to be used by the mechanism.
realm-name	The name of the realm to be presented by the mechanism.

3.9. SECURITY-DOMAIN ATTRIBUTES

You can configure **security-domain** by setting its attributes.

Attribute	Description
default-realm	The default realm contained by this security domain.
evidence-decoder	A reference to an EvidenceDecoder to be used by this domain.
outflow-anonymous	This attribute specifies whether the anonymous identity should be used if outflow to a security domain is not possible. Outflowing anonymous identity has the effect of clearing any identity already established for that domain.
outflow-security-domains	The list of security domains that the security identity from this domain should automatically outflow to.
permission-mapper	A reference to a PermissionMapper to be used by this domain.
post-realm-principal-transformer	A reference to a principal transformer to be applied after the realm has operated on the supplied identity name.
pre-realm-principal-transformer	A reference to a principal transformer to be applied before the realm is selected.
principal-decoder	A reference to a PrincipalDecoder to be used by this domain.

Attribute	Description
realm-mapper	Reference to the RealmMapper to be used by this domain.
realms	The list of realms contained by this security domain.
role-decoder	Reference to the RoleDecoder to be used by this domain.
role-mapper	Reference to the RoleMapper to be used by this domain.
security-event-listener	Reference to a listener for security events.
trusted-security-domains	The list of security domains that are trusted by this security domain.

3.10. SIMPLE-ROLE-DECODER ATTRIBUTES

You can configure simple role decoder by setting its attribute.

Table 3.13. simple-role-decoder attributes

Attribute	Description
attribute	The name of the attribute from the identity to map directly to roles.