# Red Hat JBoss Enterprise Application Platform 7.0

## How to Configure Identity Management

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 Identity Management

For Use with Red Hat JBoss Enterprise Application Platform 7.0

## Legal Notice

## Abstract

This guide explores how to use LDAP directories and other identity stores for use with JBoss EAP management interfaces and security domains. This guide expands on the concepts provided in the JBoss EAP Security Architecture, and should be reviewed after administrators have basic knowledge of LDAP and a solid understanding of security concepts within JBoss EAP.

# Table of Contents

# CHAPTER 1. IDENTITY MANAGEMENT OVERVIEW

The basics behind identity management and identity stores are covered in the *Single Sign On (SSO)* section of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide . These concepts can also be applied to providing security for the JBoss EAP management interfaces and web applications outside of SSO. More specifically, this document covers configuring different identity stores for identity management in JBoss EAP for securing the management interfaces (LDAP) as well with security domains (LDAP, Database, and Filesystem).

## 1.1. LDAP OVERVIEW

The basics of LDAP as well as a basic use case are described in the LDAP and Example Scenarios sections of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide document. It also describes the LDAP-related login modules for security domains in the Security Subsystem section.

In general, LDAP, in the context of JBoss EAP, is used as an identity store for security domains and security realms. It can be used to make authentication decisions as well as serve as an authorization authority for JBoss EAP and applications deployed to it. Various role and access information about principals can be stored in an LDAP directory which may be used directly by JBoss EAP or mapped to existing JBoss EAP roles.

**NOTE**

Similar to using a database or other external datastores to house identity information, performance of data access and data transport between the datastore and the JBoss EAP instance may have a performance impact on authenticating and authorizing principals.

# CHAPTER 2. SECURING THE MANAGEMENT INTERFACES WITH LDAP

The Management Interfaces can authenticate against an LDAP server (including Microsoft Active Directory) instead of the property-file based security realms configured by default. This is accomplished by using an LDAP authenticator. An LDAP authenticator operates by first establishing a connection (using an outbound LDAP connection) to the remote directory server. It then performs a search using the username which the user passed to the authentication system, to find the fully-qualified distinguished name (DN) of the LDAP record. If successful, a new connection is established, using the DN of the user as the credential, and password supplied by the user. If this second connection and authentication to the LDAP server is successful, the DN is verified to be valid and authentication has succeeded.

**NOTE**

Securing the Management Interfaces with LDAP changes the authentication from digest to BASIC/Plain, which by default, will cause usernames and passwords to be sent over the network in the clear. SSL/TLS can be enabled on the outbound connection to encrypt this traffic and avoid sending this information in the clear.

## 2.1. BASIC SETUP

To use an LDAP directory server as the authentication source for any or all of the management interfaces, the following steps must be performed:

1. Create an outbound connection to the LDAP server.
   The purpose of creating an outbound LDAP connection is to allow the security realm (and the JBoss EAP instance) to establish a connection to the LDAP server. This is similar to the case of creating a datasource for use with the *Database* login module in a security domain.

   The LDAP outbound connection allows the following attributes:

| Attribute | Required | Description |
|---|---|---|
| url | yes | The URL address of the directory server. |
| search-dn | no | The fully distinguished name (DN) of the user authorized to perform searches. |
| search-credential | no | The password of the user authorized to perform searches. |
| initial-context-factory | no | The initial context factory to use when establishing the connection. Defaults to **com.sun.jndi.ldap.LdapCtxFactory**. |

| Attribute | Required | Description |
|---|---|---|
| security-realm | no | The security realm to reference to obtain a configured **SSLContext** to use when establishing the connection. |
| referrals | no | Specifies the behavior when encountering a referral when doing a search. Valid options are *IGNORE*, *FOLLOW*, and *THROW*. *IGNORE* (default behavior) simply ignores the referral. If set to *FOLLOW*, when referrals are encountered during a search, the DirContext being used will attempt to follow that referral. This assumes the same connection settings can be used to connect to the second server and the name used in the referral is reachable. If set to *THROW*, then the DirContext will throw an exception (*LdapReferralException*) to indicate that a referral is required, which the security realm will handle and attempt to identify an alternative connection to use for the referral. |

| Attribute | Required | Description |
|---|---|---|
| handles-referrals-for | no | Specifies the referrals a connection can handle. If specifying list of URIs, they should be separated by spaces. This enables a connection with connection properties to be defined and used when different credentials are needed to follow a referral. This is useful in situations where different credentials are needed to authenticate against the second server or for situations where the server returns a name in the referral that is not reachable from the JBoss EAP installation and an alternative address can be substituted. |

### NOTE

The *search-dn* and *search-credential* are **different** than the username and password provided by the user. The information provided here is specifically for establishing an initial connection between the JBoss EAP instance and the LDAP server. This connection allows JBoss EAP to perform a subsequent search for the DN of the user trying to authenticate. The DN of the user (which is a result of the search) trying to authenticate and the password they provided are used to establish a second (and separate) connection for completing the authentication process.

Given the example LDAP server, below are the CLI commands for configuring an outbound LDAP connection to it:

**Table 2.1. Example LDAP Server**

| Attribute | Value |
|---|---|
| url | 127.0.0.1:389 |
| search-credential | myPass |
| search-dn | cn=search,dc=acme,dc=com |

### CLI for Adding the Outbound Connection

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

reload
```

**NOTE**

This creates a *clear* connection between the JBoss EAP instance and the LDAP server. For more details on setting up an encrypted connection using SSL/TLS, please see the Section 2.2, “Using SSL/TLS for the Outbound LDAP Connection”.

2. Create a new LDAP-enabled security realm.
   Once the outbound LDAP connection has been created, a new LDAP-Enabled security realm must be created to use it.

   The LDAP security realm has the following configuration attributes:

| Attribute | Description |
|---|---|
| connection | The name of the connection defined in outbound-connections to use to connect to the LDAP directory. |
| base-dn | The distinguished name of the context to begin searching for the user. |
| recursive | Whether the search should be recursive throughout the LDAP directory tree, or only search the specified context. Defaults to false. |
| user-dn | The attribute of the user that holds the distinguished name. This is subsequently used to test authentication as the user can complete. Defaults to dn. |
| allow-empty-passwords | This attribute determines whether an empty password is accepted. The default value for this attribute is false. |
| username-attribute | The name of the attribute to search for the user. This filter performs a simple search where the user name entered by the user matches the specified attribute. |

| Attribute | Description |
|---|---|
| advanced-filter | The fully defined filter used to search for a user based on the supplied user ID. This attribute contains a filter query in standard LDAP syntax. The filter must contain a variable in the following format: {0}. This is later replaced with the user name supplied by the user. More specifics and examples on using *advanced-filter* can be found in the `advanced-filter` part of the Combining LDAP and RBAC for Authorization section. |

**WARNING**

It is important to ensure that empty LDAP passwords are not allowed since it is a serious security concern. Unless this behavior is specifically desired in the environment, ensure empty passwords are not allowed and *allow-empty-passwords* remains false.

Here are the CLI commands for configuring an LDAP-enabled security realm using the *ldap-connection* outbound LDAP connection.

### CLI

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

/core-service=management/security-realm=ldap-security-realm/authentication=ldap:add(connection="ldap-connection", base-dn="cn=users,dc=acme,dc=com",username-attribute="sambaAccountName")

reload
```

3. Reference the new security realm in the management interface. Once a security realm has been created and is using the outbound LDAP connection, that new security realm must be referenced by the management interfaces.

### CLI Command for Updating the HTTP Interface

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

**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. USING SSL/TLS FOR THE OUTBOUND LDAP CONNECTION

JBoss EAP can be configured to use an outbound connection to an LDAP server using SSL/TLS. To create an outbound LDAP connection secured by SSL/TLS, the following procedures must be performed:

**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. Configure a security realm for the outbound LDAP connection to use.
   The security realm must contain a keystore configured with the key that the JBoss EAP server will use to decrypt/encrypt communications between itself and the LDAP server. This keystore will also allow the JBoss EAP instance to verify itself against the LDAP server. The security realm must also contain a truststore that contains the LDAP server's certificate (or the certificate of the certificate authority used to sign the LDAP server's certificate). See Setting up Two-Way SSL/TLS for the Management Interfaces in the JBoss EAP How to Configure Server Security guide for instructions on configuring keystores and truststores and creating a security realm that uses them.

2. Create an outbound LDAP connection with the SSL/TLS URL and security realm.
   Similar to the process defined in Basic Setup of Securing the Management Interfaces with LDAP, an outbound LDAP connection should be created, but using the SSL/TLS URL for the LDAP server and the SSL/TLS security realm.

   Once the outbound LDAP connection and SSL/TLS security realm for the LDAP server have been created, the outbound LDAP connection needs to be updated with that information.

   **Example CLI for Adding the Outbound Connection with an SSL/TLS URL**

   ```
   /core-service=management/ldap-connection=ldap-
   connection/:add(search-credential=myPass,
   url=ldaps://LDAP_HOST:LDAP_PORT, search-
   dn="cn=search,dc=acme,dc=com")
   ```

   **Adding the security realm with the SSL/TLS certificates**

   ```
   /core-service=management/ldap-connection=ldap-connection:write-
   attribute(name=security-realm,value="CertificateRealm")

   reload
   ```

3. Create a new security realm that uses the outbound LDAP connection for use by the management interfaces.
   Follow the procedures outlined in steps *Create a new LDAP-Enabled Security Realm*and *Reference the new security realm in the Management Interface*Securing the Management Interfaces with LDAP Basic Setup.

**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*.

## 2.3. LDAP AND RBAC

*RBAC (Role-Based Access Control)*is a mechanism for specifying a set of permissions (roles) for a set of management users, allowing users to be granted different management responsibilities without giving them full, unrestricted access. For more specific details on RBAC, please see the *Role-Based Access Control* section of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide .

RBAC is used only for authorization, with authentication being handled separately. Since LDAP can be used for authentication as well as authorization, JBoss EAP may be configured to use RBAC for authorization without LDAP (using LDAP or any other mechanism for authentication). JBoss EAP may also be configured to use RBAC combined with LDAP for making authorization decisions in the management interfaces.

### 2.3.1. Using LDAP and RBAC Independently

JBoss EAP allows for authentication and authorization to be configured independently in security realms. This enables LDAP to be configured as an authentication mechanism and RBAC to be configured as an authorization mechanism. If configured in this manner, when a user attempts to access a management interface, they will first be authenticated using the configured LDAP server. If successful, the user's role (and configured permissions of that role) will be determined using only RBAC (independent of any group information found in the LDAP server).

For more details on using just RBAC as an authorization mechanism for the management interfaces, please see the Server Security guide. For more details on configuring LDAP for authentication with the management interfaces, please see the previous section.

### 2.3.2. Combining LDAP and RBAC for Authorization

Users who have authenticated using an LDAP server (or using the properties file), can be members of user groups. A user group is simply an arbitrary label that can be assigned to one or more users. RBAC can be configured to use this group information to automatically assign a role to a user or exclude a user from a role.

An LDAP directory contains entries for user accounts and groups, cross referenced by attributes. Depending on the LDAP server configuration, a user entity may map the groups the user belongs to through *memberOf* attributes; a group entity may map which users belong to it through *uniqueMember* attributes; or a combination of the two. Once a user is successfully authenticated to the LDAP server, a group search is performed to load that user's group information. Depending on the directory server in use, group searches can be performed using their simple name (usually the username used in authentication) or by using the distinguished name of the user's entry in the directory. Group searches (`group-search`) as well as mapping between a username and a distinguished name ( `username-to-dn`) are configured when setting up LDAP as an authorization mechanism in a security realm.

Once a user's group membership information is determined from the LDAP server, a mapping within the RBAC configuration is used to determine what role(s) a user has. This mapping is configured to explicitly include or exclude groups as well as individual users.

**NOTE**

The authentication step of a user connecting to the server always happens first. Once the user is successfully authenticated the server loads the user's groups. The authentication step and the authorization step each require a connection to the LDAP server. The security realm optimizes this process by reusing the authentication connection for the group loading step.

### 2.3.2.1. Using group-search

There are two different styles that can be used when searching for group membership information: *Principal to Group* and *Group to Principal*. Principal to Group has the user's entry containing references to the group(s) it is a member of, *memberOf* attribute. Group to Principal has the group's entry contain the references to the user(s) who are members of it, *uniqueMember* attribute.

**NOTE**

JBoss EAP supports both Principal to Group as well as Group to Principal searches, but Principal to Group is recommended over Group to Principal. If Principal to Group is used, group information can be loaded directly by reading attributes of known distinguished names without having to perform any searches. Group to Principal requires extensive searches to identify the all groups that reference a user.

Both Principal to Group and Group to Principal use `group-search` which contains the following attributes:

| Attribute | Description |
|---|---|
| group-name | This attribute is used to specify the form that should be used for the group name returned as the list of groups of which the user is a member. This can either be the simple form of the group name or the group's distinguished name. If the distinguished name is required this attribute can be set to DISTINGUISHED_NAME. Defaults to SIMPLE. |
| iterative | This attribute is used to indicate if, after identifying the groups a user is a member of, we should also iteratively search based on the groups to identify which groups the groups are a member of. If iterative searching is enabled we keep going until either we reach a group that is not a member if any other groups or a cycle is detected. Defaults to false. |
| group-dn-attribute | On an entry for a group which attribute is its distinguished name. Defaults to dn. |

| Attribute | Description |
|---|---|
| group-name-attribute | On an entry for a group which attribute is its simple name. Defaults to uid. |

**NOTE**

Cyclic group membership is not a problem. A record of each search is kept to prevent groups that have already been searched from being searched again.

**IMPORTANT**

For iterative searching to work the group entries need to look the same as user entries. The same approach used to identify the groups a user is a member of is then used to identify the groups of which the group is a member. This would not be possible if for group to group membership the name of the attribute used for the cross reference changes or if the direction of the reference changes.

### Principal to Group (memberOf) for Group Search

Consider an example where a user *TestUserOne* who is a member of *GroupOne*, and *GroupOne* is in turn a member of *GroupFive*. The group membership would be shown by the use of a *memberOf* attribute at the member level. This means, *TestUserOne* would have a *memberOf* attribute set to the *dn* of *GroupOne*. *GroupOne* in turn would have a *memberOf* attribute set to the *dn* of *GroupFive*.

To use this type of searching, the *principal-to-group* element is added to the *group-search* element:

### Principal to Group, memberOf, Configuration

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

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap/group-search=principal-to-group:add(group-
attribute="memberOf",iterative=true,group-dn-attribute="dn", group-
name="SIMPLE",group-name-attribute="cn")

run-batch
```

**IMPORTANT**

The above example assumes you already have **`ldap-connection`** defined. You also need to configure the authentication mechanism which is covered earlier in this section.

Notice that the **`group-attribute`** attribute is used with the **`group-search=principal-to-group`**. For reference:

**Table 2.2. principal-to-group**

| Attribute | Description |
|---|---|
| group-attribute | The name of the attribute on the user entry that matches the distinguished name of the group the user is a member of. Defaults to memberOf. |
| prefer-original-connection | This value is used to indicate which group information to prefer when following a referral. Each time a principal is loaded, attributes from each of their group memberships are subsequently loaded. Each time attributes are loaded, either the original connection or connection from the last referral can be used. Defaults to *true*. |

### Group to Principal, uniqueMember, Group Search

Consider the same example as Principal to Group where a user *TestUserOne* who is a member of *GroupOne*, and *GroupOne* is in turn a member of *GroupFive*. However, in this case the group membership would be shown by the use of the *uniqueMember* attribute set at the group level. This means, *GroupFive* would have a *uniqueMember* set to the *dn* of *GroupOne*. *GroupOne* in turn would have a *uniqueMember* set to the *dn* of *TestUserOne*.

To use this type of searching, the *group-to-principal* element is added to the *group-search* element:

### Group to Principal, uniqueMember, Configuration

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

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap/group-search=group-to-
principal:add(iterative=true, group-dn-attribute="dn", group-
name="SIMPLE", group-name-attribute="uid", base-dn="ou=groups,dc=group-to-
principal,dc=example,dc=org", principal-attribute="uniqueMember", search-
by="DISTINGUISHED_NAME")

run-batch
```

**IMPORTANT**

The above example assumes you already have **`ldap-connection`** defined. You also need to configure the authentication mechanism which is covered earlier in this section.

Notice that the **`principal-attribute`** attribute is used with **`group-search=group-to-principal`**. **`group-to-principal`** is used to define how searches for groups that reference the user entry will be performed, and **`principal-attribute`** is used to define the group entry that references the principal.

For reference:

**Table 2.3. group-to-principal**

| Attribute | Description |
|---|---|
| base-dn | The distinguished name of the context to use to begin the search. |
| recursive | Whether sub-contexts also be searched. Defaults to false. |
| search-by | The form of the role name used in searches. Valid values are SIMPLE and DISTINGUISHED_NAME. Defaults to DISTINGUISHED_NAME. |
| prefer-original-connection | This value is used to indicate which group information to prefer when following a referral. Each time a principal is loaded, attributes from each of their group memberships are subsequently loaded. Each time attributes are loaded, either the original connection or connection from the last referral can be used. |

**Table 2.4. membership-filter**

| Attribute | Description |
|---|---|
| principal-attribute | The name of the attribute on the group entry that references the user entry. Defaults to member. |

### 2.3.2.2. Using username-to-dn

It is possible to define rules within the authorization section to convert a user’s simple user name to their distinguished name. The `username-to-dn` element specifies how to map the user name to the distinguished name of their entry in the LDAP directory. This element is **optional** and only required when both of the following are true:

- The authentication and authorization steps are against different LDAP servers.
- The group search uses the distinguished name.

**NOTE**

This could also be applicable in instances where the security realm supports both LDAP and Kerberos authentication and a conversion is needed for Kerberos, if LDAP authentication has been performed the DN discovered during authentication can be used.

It contains the following attributes:

**Table 2.5. username-to-dn**

| Attribute | Description |
|---|---|
| force | The result of a *user name to distinguished name mapping* search during authentication is cached and reused during the authorization query when the force attribute is set to `false`. When force is true, the search is performed again during authorization (while loading groups). This is typically done when different servers perform authentication and authorization. |

`username-to-dn` can be configured with one of the following:

**username-is-dn**

This specifies that the user name entered by the remote user is the user’s distinguished name.

### username-is-dn Example

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

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap/group-search=group-to-
principal:add(iterative=true, group-dn-attribute="dn", group-
name="SIMPLE", group-name-attribute="uid", base-dn="ou=groups,dc=group-
to-principal,dc=example,dc=org", principal-attribute="uniqueMember",
search-by="DISTINGUISHED_NAME")

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap/username-to-dn=username-is-dn:add(force=false)

run-batch
```

This defines a 1:1 mapping and there is no additional configuration.

**username-filter**

A specified attribute is searched for a match against the supplied user name.

### username-filter Example

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

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-
```

```
realm/authorization=ldap/group-search=group-to-
principal:add(iterative=true, group-dn-attribute="dn", group-
name="SIMPLE", group-name-attribute="uid", base-dn="ou=groups,dc=group-
to-principal,dc=example,dc=org", principal-attribute="uniqueMember",
search-by="DISTINGUISHED_NAME")

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap/username-to-dn=username-filter:add(force=false,
base-dn="dc=people,dc=harold,dc=example,dc=com", recursive="false",
attribute="sn", user-dn-attribute="dn")

run-batch
```

| Attribute | Description |
|---|---|
| base-dn | The distinguished name of the context to begin the search. |
| recursive | Whether the search will extend to sub contexts. Defaults to false. |
| attribute | The attribute of the users entry to try and match against the supplied user name. Defaults to uid. |
| user-dn-attribute | The attribute to read to obtain the users distinguished name. Defaults to dn. |

**advanced-filter**

This option uses a custom filter to locate the users distinguished name.

### advanced-filter Example

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

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap/group-search=group-to-
principal:add(iterative=true, group-dn-attribute="dn", group-
name="SIMPLE", group-name-attribute="uid", base-dn="ou=groups,dc=group-
to-principal,dc=example,dc=org", principal-attribute="uniqueMember",
search-by="DISTINGUISHED_NAME")

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap/username-to-dn=advanced-filter:add(force=true,
base-dn="dc=people,dc=harold,dc=example,dc=com", recursive="false",
user-dn-attribute="dn",filter="sAMAccountName={0}")

run-batch
```

For the attributes that match those in the *username-filter* example, the meaning and default values are the same. There is one additional attribute:

| Attribute | Description |
|---|---|
| filter | Custom filter used to search for a user’s entry where the user name will be substituted in the {0} place holder. |

**IMPORTANT**

This must remain valid after the filter is defined so if any special characters are used (such as &) ensure the proper form is used. For example *&amp;* for the *&* character.

### 2.3.2.3. Mapping LDAP Group Information to RBAC Roles

Once the connection to the LDAP server has been created and the group searching has been properly configured, a mapping needs to be created between the LDAP groups and RBAC roles. This mapping can be both inclusive as well as exclusive and enables users to be automatically assigned one or more roles based on their group membership information.

**WARNING**

If RBAC is not already configured, pay close attention when doing so, especially if switching to a newly-created LDAP-enabled realm. Enabling RBAC without having users and roles properly configured could result in administrators being unable to login to the management interfaces.

**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*.

### Ensure RBAC is Enabled and Configured

Before mappings between LDAP and RBAC Roles can be used, RBAC must enabled and initially configured.

```
/core-service=management/access=authorization:read-
attribute(name=provider)
```

It should yield the following result:

```
{ "outcome" => "success", "result" => "rbac" }
```

For more information on enabling and configuring RBAC, please see the *Enabling Role-Based Access Control* section of the Red Hat JBoss Enterprise Application Platform How to Configure Server Security guide.

### Verify Existing List of Roles

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)
```

Which should yield a list of roles:

```
{
  "outcome" => "success",
  "result" =>
    [ "Administrator", "Deployer", "Maintainer", "Monitor", "Operator",
"SuperUser" ]
}
```

In addition, all existing mappings for a role may be checked:

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

```
{
  "outcome" => "success",
  "result" =>
  {
    "include-all" => false,
    "exclude" => undefined,
    "include" => {
      "user-theboss" => {
        "name" => "theboss",
        "realm" => undefined,
        "type" => "USER"
      },
      "user-harold" => {
        "name" => "harold",
        "realm" => undefined,
        "type" => "USER"
      },
      "group-SysOps" => {
        "name" => "SysOps",
        "realm" => undefined,
        "type" => "GROUP"
      }
    }
  }
}
```

### Configure a Role-Mapping entry

If a role does not already have a Role-Mapping entry, one needs to be created. For instance:

```
/core-service=management/access=authorization/role-mapping=Auditor:read-resource()
```

```
{
  "outcome" => "failed",
  "failure-description" => "WFLYCTL0216: Management resource '[ (\"core-service\" => \"management\"), (\"access\" => \"authorization\"), (\"role-mapping\" => \"Auditor\") ]' not found"
}
```

To add a role mapping:

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

```
{
  "outcome" => "success"
}
```

To verify:

```
/core-service=management/access=authorization/role-mapping=Auditor:read-resource()
```

```
{
  "outcome" => "success",
  "result" => {
    "include-all" => false,
    "exclude" => undefined,
    "include" => undefined
  }
}
```

### Add Groups to the Role for Inclusion and Exclusion

Groups may be added for inclusion or exclusion from a role.

**NOTE**

The exclusion mapping takes precedence or the inclusion mapping.

To add a group for inclusion:

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

To add a group for exclusion:

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

To check the result:

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

```
{
  "outcome" => "success",
  "result" => {
    "include-all" => false,
    "exclude" => {
      "group-GroupToExclude" => {
        "name" => "GroupToExclude",
        "realm" => undefined,
        "type" => "GROUP"
      }
    },
    "include" => {
      "group-GroupToInclude" => {
        "name" => "GroupToInclude",
        "realm" => undefined,
        "type" => "GROUP"
      }
    }
  }
}
```

### Removing a Group from a Role Groups that are added for exclusion or inclusion may also be removed

To remove a group from inclusion:

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

To remove a group from exclusion:

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

## 2.4. ENABLING CACHING

Security Realms also offer the ability to cache the results of LDAP queries for both authentication as well as group loading. This enables the results of different queries to be reused across multiple searches by different users in certain circumstances, for example iteratively querying the group membership information of groups. There are three different caches available, each of which are configured separately and operate independently:

- authentication
- group-to-principal
- username-to-dn

### 2.4.1. Cache Configuration

Even though the caches are independent of one another, all three are configured in the same manner. Each cache offers the following configuration options:

| Attribute | Description |
|---|---|
| type | This defines the eviction strategy that the cache will adhere to. Options are **by-access-time** and **by-search-time**. **by-access-time** evicts items from the cache after a certain period of time has elapsed since their last access. **by-search-time** evicts items based on how long they have been in the cache regardless of their last access. |
| eviction-time | This defines the time (in seconds) used for evictions depending on the strategy. |
| cache-failures | This is a boolean that enables/disables the caching of failed searches. This has the potential for preventing an LDAP server from being repeatedly access by the same failed search, but it also has the potential to fill up the cache with searches for users that do not exist. This setting is particularly important for the authentication cache. |
| max-cache-size | This defines maximum size (number of items) of the cache, which in-turn dictates when items will begin getting evicted. Old items are evicted from the cache to make room for new authentication and searches as needed, meaning *max-cache-size* will not prevent new authentication attempts or searches from occurring. |

### 2.4.2. Example

**NOTE**

This example assumes a security realm has been created named *LDAPRealm* that connects to an existing LDAP server and is configured for authentication and authorization. The commands to display the current configuration are detailed in a later section. More details on creating a security realm that uses LDAP can be found here.

**Example Base Configuration**

```
"core-service" : {
  "management" : {
    "security-realm" : {
      "LDAPRealm" : {
        "authentication" : {
          "ldap" : {
            "allow-empty-passwords" : false,
            "base-dn" : "...",
            "connection" : "MyLdapConnection",
            "recursive" : false,
            "user-dn" : "dn",
            "username-attribute" : "uid",
            "cache" : null
          }
        },
        "authorization" : {
          "ldap" : {
            "connection" : "MyLdapConnection",
            "group-search" : {
              "group-to-principal" : {
                "base-dn" : "...",
                "group-dn-attribute" : "dn",
                "group-name" : "SIMPLE",
                "group-name-attribute" : "uid",
                "iterative" : true,
                "principal-attribute" : "uniqueMember",
                "search-by" : "DISTINGUISHED_NAME",
                "cache" : null
              }
            },
            "username-to-dn" : {
              "username-filter" : {
                "attribute" : "uid",
                "base-dn" : "...",
                "force" : false,
                "recursive" : false,
                "user-dn-attribute" : "dn",
                "cache" : null
                }
              }
            }
          },
        }
      }
  }
}
```

In all areas where **"cache" : null** appear, a cache may be configured:

**Authentication**

During authentication the user's distinguished name is discovered using this definition and an attempt to connect to the LDAP server and verify their identity is made using these credentials.

**A group-search definition**

There is the group search definition, in this case it is an iterative search (because **iterative** is set to **true** in the sample configuration above). First, a search will be performed to find all groups the user is a direct member of. After that, a search will be performed for each of those groups to identify if they have membership to other groups. This process continues until either a cyclic reference is detected or the final groups are not members of any further groups.

**A username-to-dn definition in group search**

Group searching relies on the availability of the users distinguished name. This section is not used in all situations, but it can be used as a second attempt to discover a user's distinguished name. This, for instance, may be useful or even required when a second form of authentication was supported, for example local authentication.

#### 2.4.2.1. Reading the Current Cache Configuration

**NOTE**

The CLI commands used in this and subsequent sections use *LDAPRealm* for the name of the security realm. This should be substituted for the name of the actual realm being configured.

**CLI Command to Read the Current Cache Configuration**

```
/core-service=management/security-realm=LDAPRealm:read-
resource(recursive=true)
```

**Output**

```
{
  "outcome" => "success",
  "result" => {
    "map-groups-to-roles" => true,
    "authentication" => {
      "ldap" => {
        "advanced-filter" => undefined,
        "allow-empty-passwords" => false,
        "base-dn" => "dc=example,dc=com",
         "connection" => "ldapConnection",
         "recursive" => true,
         "user-dn" => "dn",
         "username-attribute" => "uid",
         "cache" => undefined
        }
      },
      "authorization" => {
        "ldap" => {
          "connection" => "ldapConnection",
          "group-search" => {
            "principal-to-group" => {
              "group-attribute" => "description",
              "group-dn-attribute" => "dn",
              "group-name" => "SIMPLE",
              "group-name-attribute" => "cn",
              "iterative" => false,
              "prefer-original-connection" => true,
              "skip-missing-groups" => false,
              "cache" => undefined
            }
          },
          "username-to-dn" => {
            "username-filter" => {
              "attribute" => "uid",
              "base-dn" => "ou=Users,dc=jboss,dc=org",
              "force" => true,
              "recursive" => false,
              "user-dn-attribute" => "dn",
              "cache" => undefined
            }
          }
        }
      },
      "plug-in" => undefined,
      "server-identity" => undefined
    }
   }
```

### 2.4.2.2. Enabling a Cache

**NOTE**

The CLI commands used in this and subsequent sections configure the cache in the authentication portion of the security realm, in other words **`/authentication=ldap/`**. Caches in the authorization portion may also be configured in a similar manner by updating the path of the command.

**CLI for Enabling a Cache**

```
/core-service=management/security-
realm=LDAPRealm/authentication=ldap/cache=by-access-time:add(eviction-
time=300, cache-failures=true, max-cache-size=100)
```

This commands adds a **by-access-time** cache for authentication with an eviction time of 300 seconds (5 minutes) and a max cache size of 100 items. In addition, failed searches will be cached. Alternatively, a **by-search-time** cache could also be configured:

```
/core-service=management/security-
realm=LDAPRealm/authentication=ldap/cache=by-search-time:add(eviction-
time=300, cache-failures=true, max-cache-size=100)
```

### 2.4.2.3. Inspecting an Existing Cache

**CLI for Inspecting an Existing Cache**

```
/core-service=management/security-
realm=LDAPRealm/authentication=ldap/cache=by-access-time:read-
resource(include-runtime=true)
```

**Output**

```
{
  "outcome" => "success",
  "result" => {
    "cache-failures" => true,
    "cache-size" => 1,
    "eviction-time" => 300,
    "max-cache-size" => 100
  }
}
```

The *include-runtime* attribute adds the **cache-size** element, which displays the current number of items in the cache (1 in this case).

#### 2.4.2.4. Testing an Existing Cache's Contents

**CLI for Testing an Existing Cache's Contents**

```
/core-service=management/security-
realm=LDAPRealm/authentication=ldap/cache=by-access-
time:contains(name=TestUserOne)
```

**Output**

```
{
  "outcome" => "success",
  "result" => true
}
```

This shows that an entry for *TestUserOne* exists the in the cache.

#### 2.4.2.5. Flushing a Cache

An entire cache may be flushed as can individual items from a cache.

**CLI for Flushing a Single Item**

```
/core-service=management/security-
realm=LDAPRealm/authentication=ldap/cache=by-access-time:flush-
cache(name=TestUserOne)
```

**CLI for Flushing an Entire Cache**

```
/core-service=management/security-
realm=LDAPRealm/authentication=ldap/cache=by-access-time:flush-cache()
```

#### 2.4.2.6. Removing a Cache

**CLI for Removing a Cache**

```
/core-service=management/security-
realm=LDAPRealm/authentication=ldap/cache=by-access-time:remove()

reload
```

# CHAPTER 3. CONFIGURING A SECURITY DOMAIN TO USE LDAP

Security domains can be configured to use an LDAP server for authentication and authorization by using a login module. The basics of security domains and login modules are covered in the Red Hat JBoss Enterprise Application Platform Security Architecture Guide. *LdapExtended* is the preferred login module for integrating with LDAP servers (including Active Directory), but there are several other LDAP login modules that can be used as well. Specifically, the *Ldap*, *AdvancedLdap*, and *AdvancedAdLdap* can also be used to configure a security domain to use LDAP. This section uses the *LdapExtended* login module to illustrate how to create a security domain that uses an LDAP for authentication and authorization, but the other LDAP login modules may be used as well. For more details on the other LDAP login modules, please see the Red Hat JBoss Enterprise Application Platform Login Module Reference.

## 3.1. LDAPEXTENDED LOGIN MODULE

LdapExtended (**`org.jboss.security.auth.spi.LdapExtLoginModule`**) is a login module implementation that uses searches to locate the bind user and associated roles on LDAP server. The roles query recursively follows DNs to navigate a hierarchical role structure. For the vast majority of cases when using LDAP with security domains, the *LdapExtended* login module should be used, especially with LDAP implementations that are not Active Directory. For a full list of configuration options for the LdapExtended login module, please see the LdapExtend login module section Red Hat JBoss Enterprise Application Platform Login Module Reference.

The authentication happens as follows:

1. An initial bind to the LDAP server is done using the *bindDN* and *bindCredential* options. The *bindDN* is a LDAP user with the ability to search both the *baseCtxDN* and *rolesCtxDN* trees for the user and roles. The user DN to authenticate against is queried using the filter specified by the baseFilter attribute.

2. The resulting user DN is authenticated by binding to the LDAP server using the user DN as the InitialLdapContext environment *Context.SECURITY_PRINCIPAL*. The *Context.SECURITY_CREDENTIALS* property is set to the String password obtained by the callback handler.

### 3.1.1. Configuring a Security Domain to use the LdapExtended Login Module

**Example Data (LDIF format)**

```
dn: uid=jduke,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: Java Duke
sn: duke
uid: jduke
userPassword: theduke
# =============================
dn: uid=hnelson,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: Horatio Nelson
```

```
sn: Nelson
uid: hnelson
userPassword: secret
# =============================
dn: ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: groups
# =============================
dn: uid=ldap,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: LDAP
sn: Service
uid: ldap
userPassword: randall
# =============================
dn: ou=Users,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Users
# =============================
dn: dc=jboss,dc=org
objectclass: top
objectclass: domain
dc: jboss
# =============================
dn: uid=GroupTwo,ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
objectClass: uidObject
cn: GroupTwo
member: uid=jduke,ou=Users,dc=jboss,dc=org
uid: GroupTwo
# =============================
dn: uid=GroupThree,ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: GroupThree
uid: GroupThree
uniqueMember: uid=GroupOne,ou=groups,dc=jboss,dc=org
# =============================
dn: uid=HTTP,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: HTTP
sn: Service
uid: HTTP
userPassword: httppwd
# =============================
dn: uid=GroupOne,ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: groupOfUniqueNames
```

```
objectClass: uidObject
cn: GroupOne
uid: GroupOne
uniqueMember: uid=jduke,ou=Users,dc=jboss,dc=org
uniqueMember: uid=hnelson,ou=Users,dc=jboss,dc=org
```

### CLI Commands for Adding the LdapExtended Login Module

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

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

/subsystem=security/security-
domain=testLdapExtendedExample/authentication=classic/login-
module=LdapExtended:add(code=LdapExtended, flag=required, module-options=[
("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"),
("java.naming.provider.url"=>"ldap://localhost:10389"),
("java.naming.security.authentication"=>"simple"),
("bindDN"=>"uid=ldap,ou=Users,dc=jboss,dc=org"),
("bindCredential"=>"randall"),
("baseCtxDN"=>"ou=Users,dc=jboss,dc=org"), ("baseFilter"=>"(uid={0})"),
("rolesCtxDN"=>"ou=groups,dc=jboss,dc=org"),     ("roleFilter"=>"
(uniqueMember={1})"), ("roleAttributeID"=>"uid")])

reload
```

**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*.

### 3.1.1.1. Configuring a Security Domain to use the LdapExtended Login Module for Active Directory

For Microsoft Active Directory, the LdapExtended Login Module may be used.

The example below represents the configuration for a default Active Directory configuration. Some Active Directory configurations may require searching against the Global Catalog on port 3268 instead of the usual port 389. This is most likely when the Active Directory forest includes multiple domains.

### Example Configuration for the LdapExtended Login Module for a Default AD Configuration

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

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

/subsystem=security/security-
domain=AD_Default/authentication=classic/login-
module=LdapExtended:add(code=LdapExtended,flag=required,module-options=[
("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"),
("bindDN"=>"JBOSSsearchuser"), ("bindCredential"=>"password"),
("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"), ("baseFilter"=>"
(sAMAccountName={0})"), ("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"),
("roleFilter"=>"(sAMAccountName={0})"), ("roleAttributeID"=>"memberOf"),
("roleAttributeIsDN"=>"true"), ("roleNameAttributeID"=>"cn"),
("searchScope"=>"ONELEVEL_SCOPE"), ("allowEmptyPasswords"=>"false")])

reload
```

The example below implements a recursive role search within Active Directory. The key difference between this example and the default Active Directory example is that the role search has been replaced to search the member attribute using the DN of the user. The login module then uses the DN of the role to find groups of which the group is a member.

### Example Configuration for the LdapExtended Login Module for a Default AD Configuration with Recursive Search

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

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

/subsystem=security/security-
domain=AD_Recursive/authentication=classic/login-
module=LdapExtended:add(code=LdapExtended,flag=required,module-options=
[("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"),
("java.naming.referral"=>"follow"), ("bindDN"=>"JBOSSsearchuser"),
("bindCredential"=>"password"), ("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"),
("baseFilter"=>"(sAMAccountName={0})"),
("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"), ("roleFilter"=>"(member=
{1})"), ("roleAttributeID"=>"cn"), ("roleAttributeIsDN"=>"false"),
("roleRecursion"=>"2"), ("searchScope"=>"ONELEVEL_SCOPE"),
("allowEmptyPasswords"=>"false")])

reload
```

# CHAPTER 4. CONFIGURING A SECURITY DOMAIN TO USE A DATABASE

Similar to LDAP, security domains can be configured to use a database for authentication and authorization by using a login module.

## 4.1. DATABASE LOGIN MODULE

The Database login module is a Java Database Connectivity-based (JDBC) login module that supports authentication and role mapping. This login module is used if username, password and role information are stored in a relational database.

This works by providing a reference to logical tables containing Principals and Roles in the expected format. For example:

```
Table Principals(PrincipalID text, Password text) Table Roles(PrincipalID
text, Role text, RoleGroup text)
```

The Principals table associates the user PrincipalID with the valid password and the Roles table associates the user PrincipalID with its role sets. The roles used for user permissions must be contained in rows with a RoleGroup column value of Roles.

The tables are logical in that users can specify the SQL query that the login module uses. The only requirement is that the **`java.sql.ResultSet`** has the same logical structure as the Principals and Roles tables described previously. The actual names of the tables and columns are not relevant as the results are accessed based on the column index.

To clarify this notion, consider a database with two tables, Principals and Roles, as already declared. The following statements populate the tables with the following data:

- PrincipalID *java* with a Password of *echoman* in the Principals table
- PrincipalID *java* with a role named *Echo* in the *RolesRoleGroup* in the Roles table
- PrincipalID *java* with a role named *caller-java* in the *CallerPrincipalRoleGroup* in the Roles table

For a full list of configuration options for the Database login module, please see the Database login module section Red Hat JBoss Enterprise Application Platform Login Module Reference.

### 4.1.1. Configuring a Security Domain to use the Database Login Module

Before configuring a security domain to use the Database login module, a datasource must be properly configured.

For more information on creating and configure datasources in JBoss EAP please see the Datasource Management section of the Red Hat JBoss Enterprise Application Platform Configuration Guide .

Once a datasource has been properly configured, a security domain may be configured to use the Database login module. The below example assumes a datasource named *MyDatabaseDS* has been created and properly configured with a database that is constructed with the following:

```
CREATE TABLE Users(username VARCHAR(64) PRIMARY KEY, passwd VARCHAR(64))
CREATE TABLE UserRoles(username VARCHAR(64), role VARCHAR(32))
```

### CLI Commands for Adding the Database Login Module

```
/subsystem=security/security-domain=testDB:add

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

/subsystem=security/security-domain=testDB/authentication=classic/login-module=Database:add(code=Database,flag=required,module-options=[("dsJndiName"=>"java:/MyDatabaseDS"),("principalsQuery"=>"select passwd from Users where username=?"),("rolesQuery"=>"select role, 'Roles' from UserRoles where username=?")])

reload
```

# CHAPTER 5. CONFIGURING A SECURITY DOMAIN TO USE A FILESYSTEM

Security domains can also be configured to use a filesystem as an identity store for authentication and authorization by using a login module.

## 5.1. USERSROLES LOGIN MODULE

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

**NOTE**

This login module supports password stacking, password hashing, and unauthenticated identity.

The properties files are loaded during initialization using the initialize method thread context class loader. This means that these files can be placed on the classpath of the Java EE deployment (for example, into the **`WEB-INF/classes`** folder in the WAR archive), or into any directory on the server classpath.

For a full list of configuration options for the UsersRoles login module, please see the UsersRoles login module section Red Hat JBoss Enterprise Application Platform Login Module Reference.

### 5.1.1. Configuring a Security Domain to use the UsersRoles Login Module

The below example assumes the following files have been created and are available on the application’s classpath:

- sampleapp-users.properties
- sampleapp-roles.properties

**CLI Commands for Adding the UserRoles Login Module**

```
/subsystem=security/security-domain=sampleapp:add

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

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

reload
```

# CHAPTER 6. CONFIGURING A SECURITY DOMAIN TO USE CERTIFICATE-BASED AUTHENTICATION

JBoss EAP provides you with the ability to use certificate-based authentication with security domains to secure web applications or EJBs.

**IMPORTANT**

Before you can configure certificate-based authentication, you need to have Two-Way SSL/TLS for Applications enabled and configured, which requires X509 certificates configured for both the JBoss EAP instance as well as any clients accessing the web application or EJB secured by the security domain.

Once the certificates, truststores, and two-way SSL/TLS are configured, you then can proceed with configuring a security domain that uses certificate-based authentication, configuring an application to use that security domain, and configuring your client to use the client certificate.

## 6.1. CREATING A SECURITY DOMAIN WITH CERTIFICATE-BASED AUTHENTICATION

To create a security domain that uses certificate-based authentication, you need to specify a truststore as well as a Certificate login module or one of its subclasses.

The truststore must contain any trusted client certificates used for authentication, or it must contain the certificate of the certificate authority used to sign the client's certificate. The login module is used to authenticate the certificate presented by the client using the configured truststore. The security domain as a whole also must provide a way to map a role to the principal once it is authenticated. The Certificate login module itself will not map any role information to the principal, but it may be combined with another login module to do so. Alternatively, two subclasses of the Certificate login module, CertificateRoles and DatabaseCertificate, do provide a way to map roles to a principal after it is authenticated. The below example shows how to configure a security domain with certificate-based authentication using the CertificateRoles login module.

**WARNING**

When performing authentication, the security domain will use the same certificate presented by the client when establishing two-way SSL/TLS. As a result, the client must use the same certificate for **BOTH** two-way SSL/TLS and the certificate-based authentication with the application or EJB.

### Example Security Domain with Certificate-Based Authentication

```
/subsystem=security/security-domain=cert-roles-domain:add

/subsystem=security/security-domain=cert-roles-
domain/jsse=classic:add(truststore={password=secret,
url="/path/to/server.truststore.jks"}, keystore={password=secret,
url="/path/to/server.keystore.jks"}, client-auth=true)
```

```
/subsystem=security/security-domain=cert-roles-
domain/authentication=classic:add

/subsystem=security/security-domain=cert-roles-
domain/authentication=classic/login-
module=CertificateRoles:add(code=CertificateRoles, flag=required, module-
options=[ securityDomain="cert-roles-domain",
rolesProperties="${jboss.server.config.dir}/cert-
roles.properties",password-stacking="useFirstPass",
verifier="org.jboss.security.auth.certs.AnyCertVerifier"])
```

**NOTE**

The above example uses the CertificateRoles login module to handle authentication and map roles to authenticated principals. It does so by referencing a properties file using the **`rolesProperties`** attribute. This file lists usernames and roles using the following format:

```
user1=roleA
user2=roleB,roleC
user3=
```

Since usernames are presented as the DN from the provided certificate, for example **`CN=valid-client, OU=JBoss, O=Red Hat, L=Raleigh, ST=NC, C=US`**, you have to escape special characters such as = and spaces when using a properties file:

**Example Roles Properties File**

```
CN\=valid-client,\ OU\=JBoss,\ O\=Red\ Hat,\ L\=Raleigh,\
ST\=NC,\ C\=US=Admin
```

To view, the DN of certificate:

```
$ keytool -printcert -file valid-client.crt
Owner: CN=valid-client, OU=JBoss, O=Red Hat, L=Raleigh, ST=NC,
C=US
...
```

## 6.2. CONFIGURING AN APPLICATION TO USE A SECURITY DOMAIN WITH CERTIFICATE-BASED AUTHENTICATION

Similar to configuring an application to use a security domain with other forms of authentication, you need to configure both the **`jboss-web.xml`** and **`web.xml`** files appropriately.

For **`jboss-web.xml`**, you add a reference to the security domain you configured for certificate-based authentication.

**Example jboss-web.xml**

```
<jboss-web>
  <security-domain>cert-roles-domain</security-domain>
</jboss-web>
```

For the **`web.xml`**, set the **`<auth-method>`** attribute in **`<login-config>`** to **`CLIENT-CERT`**. You also need to have a **`<security-constraint>`** as well as **`<security-roles>`** defined as well.

### Example web.xml

```
<web-app>
  <!-- URL for secured portion of application-->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secure</web-resource-name>
      <url-pattern>/secure/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>All</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Security roles referenced by this web application -->
  <security-role>
    <description>The role that is required to log in to the
application</description>
    <role-name>All</role-name>
  </security-role>

  <login-config>
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>cert-roles-domain</realm-name>
  </login-config>
</web-app>
```

## 6.3. CONFIGURING THE CLIENT

For a client to authenticate against an application secured with certificate-based authentication, the client needs access to a client certificate that is contained in the JBoss EAP instance’s truststore. For example, if accessing the application using a browser, the client will need to import the trusted certificate into the browser’s truststore.

# CHAPTER 7. CONFIGURING A SECURITY DOMAIN TO USE A SECURITY MAPPING

Adding a security mapping to a security domain allows for authentication and authorization information to be combined after the authentication or authorization happens, but before the information is passed to the application. For more information on security mapping, please see the Security Mapping section of the Red Hat JBoss Enterprise Application Platform Security Architecture Guide.

To add a security mapping to an existing security domain, a *code*, *type*, and relevant module options must be configured. The *code* field is the short name, for example *SimpleRoles*, *PropertiesRoles*, *DatabaseRoles*, or class name of the security mapping module. The *type* field refers to the type of mapping this module performs, and the allowed values are *principal*, *role*, *attribute*, or *credential*. For a full list of the available security mapping modules and their module options, see the Security Mapping Modules section of the Red Hat JBoss Enterprise Application Platform Login Module Reference.

### Example CLI Commands for Adding a SimpleRoles Security Mapping to an Existing Security Domain

```
/subsystem=security/security-domain=sampleapp/mapping=classic:add

/subsystem=security/security-domain=sampleapp/mapping=classic/mapping-
module=SimpleRoles:add(code=SimpleRoles,type=role,module-options=
[("user1"=>"specialRole")])

reload
```

# CHAPTER 8. STANDALONE SERVER VS. MANAGED DOMAIN CONSIDERATIONS

Setting up identity management with an LDAP server, including Microsoft Active Directory, is essentially the same regardless or whether it is used in a standalone server or a managed domain. In general, this also applies to setting up most identity stores with both security realms and security domains. Just as with any other configuration setting, the standalone configuration resides in the **`standalone.xml`** file and the configuration for a managed domain resides in the **`domain.xml`** and **`host.xml`** files.

*Revised on 2018-02-08 10:17:55 EST*