Red Hat JBoss Enterprise Application Platform 8.0

Using single sign-on with JBoss EAP

Guide to using single sign-on to add authentication to applications deployed on JBoss EAP

Last Updated: 2024-05-07

Red Hat JBoss Enterprise Application Platform 8.0 Using single sign-on with JBoss EAP

Guide to using single sign-on to add authentication to applications deployed on JBoss EAP

Legal Notice

Abstract

Guide to using single sign-on to add authentication to applications deployed on JBoss EAP.

Table of Contents

PROVIDING FEEDBACK ON JBOSS EAP DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Submit** creates and routes the issue to the appropriate documentation team.

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. SINGLE SIGN-ON IN JBOSS EAP

Single sign-on (SSO) is a process of authenticating identities for multiple clients from a central identity provider. For example, a user needs only one set of login credentials to log in to different applications that use the same SSO provider.

JBoss EAP supports the following SSO protocols:

OpenID Connect (OIDC)

OpenID Connect is an authentication protocol based on the OAuth 2.0 framework of specifications specified in RFC 6749 and RFC 6750.

Security Assertion Mark-up Language v2 (SAML v2)

SAML is a data format and protocol that enables the exchange of authentication and authorization information between two parties, typically an identity provider and a service provider. This information is exchanged in the form of SAML tokens that contain assertions, and are issued by Identity Providers to subjects for authenticating with Service Providers. Subjects can reuse SAML tokens issued by an identity provider with multiple service providers, supporting browser-based Single Sign-On in SAML v2.

You can use SSO to secure applications deployed on JBoss EAP running on bare metal as well as JBoss EAP running on Red Hat OpenShift Container Platform. For information about securing applications deployed on JBoss EAP running on Red Hat OpenShift Container Platform with SSO, see the Using JBoss EAP on OpenShift Container Platform.

CHAPTER 2. SECURING APPLICATIONS DEPLOYED ON JBOSS EAP WITH SINGLE SIGN-ON

You can secure applications with Single Sign-on (SSO) to delegate authentication to an SSO-provider such as Red Hat build of Keycloak. You can use either OpenID Connect (OIDC) or Security Assertion Markup Language v2 (SAML v2) as the SSO protocols.

To secure applications with SSO, follow these procedures:

- Create an example application to secure with Single sign-on : Use this procedure to create a simple web-application for securing with SSO. If you already have an application to secure with SSO, skip this step.
- Create a realm and users in Red Hat build of Keycloak
- Secure your application with SSO by using either OIDC or SAML as the protocol:
  - Secure applications with OIDC
  - Secure applications with SAML

2.1. CREATING AN EXAMPLE APPLICATION TO SECURE WITH SINGLE SIGN-ON

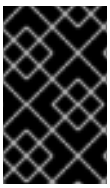
Create a web-application to deploy on JBoss EAP and secure it with Single sign-on (SSO) with OpenID Connect (OIDC) or Security Assertion Mark-up Language (SAML).

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 Creating a realm and users in Red Hat build of Keycloak.

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.



IMPORTANT

The following procedure is provided only as an example and should not be used in a production environment. For information about creating applications for JBoss EAP, see Getting started with developing applications for JBoss EAP deployment .

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>
      <version.maven.war.plugin>3.4.0</version.maven.war.plugin>
    </properties>
```

```
    <dependencies>
        <dependency>
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <version>6.0.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <finalName>${project.artifactId}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>${version.maven.war.plugin}</version>
            </plugin>
            <plugin>
                <groupId>org.wildfly.plugins</groupId>
                <artifactId>wildfly-maven-plugin</artifactId>
                <version>4.2.2.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] ------------------------------------------------------------------------
```

Next steps

- Creating a web application

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. If there is no logged-in user, the servlet returns the text "NO AUTHENTICATED USER".

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 web-application development .
- 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 jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

/**
 * A simple secured HTTP servlet. It returns the user name of 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()) {
            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 '");
            Principal user = req.getUserPrincipal();
            writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
            writer.print("'");
            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.

Next steps

- Creating a realm and users in Red Hat build of Keycloak

2.2. CREATING A REALM AND USERS IN RED HAT BUILD OF KEYCLOAK

A realm in Red Hat build of Keycloak is equivalent to a tenant. Each realm allows an administrator to create isolated groups of applications and users.

The following procedure outlines the minimum steps required to get started with securing applications deployed to JBoss EAP with Red Hat build of Keycloak for testing purposes. For detailed configurations, see the Red Hat build of Keycloak Server Administration Guide .

NOTE

The following procedure is provided as an example only. If you already have configured a realm and users in Red Hat build of Keycloak, you can skip this procedure and go directly to securing applications. For more information, see:

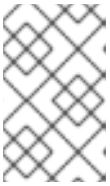
- Securing applications with OIDC
- Securing applications with SAML

Prerequisites

- You have administrator access to Red Hat build of Keycloak.

Procedure

1. Start the Red Hat build of Keycloak server at a port other than 8080 because JBoss EAP default port is 8080.



NOTE

The **start-dev** command is not meant for production environments. For more information, see Trying Red Hat build of Keycloak in development mode in the Red Hat build of Keycloak Server Guide.

Syntax

```
$ <path_to_rhbk>/bin/kc.sh start-dev --http-port <offset-number>
```

Example

```
$ /home/servers/rhbk-22.0/bin/kc.sh start-dev --http-port 8180
```

2. Log in to the Admin Console at **http://localhost:<port>/**. For example, **http://localhost:8180/**.
3. Create a realm.
   a. Hover over **Master**, and click **Create Realm**.
   b. Enter a name for the realm. For example, **example_realm**.
   c. Ensure that **Enabled** is set to **ON**.
   d. Click **Create**.

   For more information, see Creating a realm in the Red Hat build of Keycloak Server Administration Guide.
4. Create a user.

a. Click **Users**, then click **Add user**,

b. Enter a user name. For example, **user1**.

c. Click **Create**.

For more information, see Creating users in the Red Hat build of Keycloak Server Administration Guide.

5. Set credentials for the user.

a. Click **Credentials**.

b. Set a password for the user. For example, **passwordUser1**. Toggle **Temporary** to **OFF** and click **Set Password**. In the confirmation prompt, click **Save**.

For more information, see Defining user credentials in the Red Hat build of Keycloak Server Administration Guide.

6. Create a role.
This is the role name you configure in JBoss EAP for authorization.

a. Click **Realm Roles**, then **Create role**.

b. Enter a role name, such as *Admin*.

c. Click **Save**.

7. Assign the role to the user.

a. Click **Users**.

b. Click the user to which you want to assign the role.

c. Click **Role Mapping**.

d. Click **Assign role**.

e. Select the role to assign. For example, *Admin*. Click **Assign**.

For more information, see Creating a realm role in the Red Hat build of Keycloak Server Administration Guide.

Next steps

- To use this realm to secure applications deployed to JBoss EAP, follow these procedures:
  - Securing applications with OIDC
  - Securing applications with SAML

Additional resources

- Red Hat build of Keycloak Getting Started Guide

2.3. SECURING APPLICATIONS WITH OIDC

Use the JBoss EAP native OpenID Connect (OIDC) client to secure your applications using an external OpenID provider. OIDC is an identity layer that enables clients, such as JBoss EAP, to verify a user's identity based on authentication performed by an OpenID provider. For example, you can secure your JBoss EAP applications using Red Hat build of Keycloak as the OpenID provider.

To secure applications with OIDC, follow these procedures:

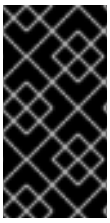
- Creating an OIDC client in JBoss EAP
- Securing a web application using OpenID Connect

2.3.1. Application security with OpenID Connect in JBoss EAP

When you secure your applications using an OpenID provider, you do not need to configure any security domain resources locally. The **elytron-oidc-client** subsystem provides a native OpenID Connect (OIDC) client in JBoss EAP to connect with OpenID providers (OP). JBoss EAP automatically creates a virtual security domain for your application, based on your OpenID provider configurations. The **elytron-oidc-client** subsystem acts as the Relying Party (RP).

NOTE

The JBoss EAP native OIDC client does not support RP-Initiated logout.



IMPORTANT

It is recommended to use the OIDC client with Red Hat build of Keycloak. You can use other OpenID providers if they can be configured to use access tokens that are JSON Web Tokens (JWTs) and can be configured to use the RS256, RS384, RS512, ES256, ES384, or ES512 signature algorithm.

To enable the use of OIDC, you can configure either the **elytron-oidc-client** subsystem or an application itself. JBoss EAP activates the OIDC authentication as follows:

- When you deploy an application to JBoss EAP, the **elytron-oidc-client** subsystem scans the deployment to detect if the OIDC authentication mechanism is required.
- If the subsystem detects OIDC configuration for the deployment in either the **elytron-oidc-client** subsystem or the application deployment descriptor, JBoss EAP enables the OIDC authentication mechanism for the application.
- If the subsystem detects OIDC configuration in both places, the configuration in the **elytron-oidc-client** subsystem **secure-deployment** attribute takes precedence over the configuration in the application deployment descriptor.

Deployment configuration
To secure an application with OIDC by using a deployment descriptor, update the application's deployment configuration as follows:

- Set the **auth-method** property to **OIDC** in the application deployment descriptor **web.xml** file.

Example deployment descriptor update

```
<login-config>
    <auth-method>OIDC</auth-method>
</login-config>
```

- Create a file called **oidc.json** in the **WEB-INF** directory with the OIDC configuration information.

  Example oidc.json contents

```
{
  "client-id" : "customer-portal", (1)
  "provider-url" : "http://localhost:8180/realms/demo", (2)
  "ssl-required" : "external", (3)
   "credentials" : {
      "secret" : "234234-234234-234234" (4)
   }
}
```

(1) The name to identify the OIDC client with the OpenID provider.

(2) The OpenID provider URL.

(3) Require HTTPS for external requests.

(4) The client secret that was registered with the OpenID provider.

Subsystem configuration

You can secure applications with OIDC by configuring the **elytron-oidc-client** subsystem in the following ways:

- Create a single configuration for multiple deployments if you use the same OpenID provider for each application.
- Create a different configuration for each deployment if you use different OpenID providers for different applications.

Example XML configuration for a single deployment:

```
<subsystem xmlns="urn:wildfly:elytron-oidc-client:1.0">
   <secure-deployment name="DEPLOYMENT_RUNTIME_NAME.war"> (1)
      <client-id>customer-portal</client-id> (2)
      <provider-url>http://localhost:8180/realms/demo</provider-url> (3)
      <ssl-required>external</ssl-required> (4)
      <credential name="secret" secret="0aa31d98-e0aa-404c-b6e0-e771dba1e798" /> (5)
   </secure-deployment>
</subsystem>
```

(1) The deployment runtime name.

(2) The name to identify the OIDC client with the OpenID provider.

(3) The OpenID provider URL.

(4) Require HTTPS for external requests.

(5) The client secret that was registered with the OpenID provider.

To secure multiple applications using the same OpenID provider, configure the **provider** separately, as shown in the example:

```
<subsystem xmlns="urn:wildfly:elytron-oidc-client:1.0">
    <provider name="${OpenID_provider_name}">
        <provider-url>http://localhost:8080/realms/demo</provider-url>
        <ssl-required>external</ssl-required>
    </provider>
    <secure-deployment name="customer-portal.war"> 1
        <provider>${OpenID_provider_name}</provider>
        <client-id>customer-portal</client-id>
        <credential name="secret" secret="0aa31d98-e0aa-404c-b6e0-e771dba1e798" />
    </secure-deployment>
    <secure-deployment name="product-portal.war"> 2
        <provider>${OpenID_provider_name}</provider>
        <client-id>product-portal</client-id>
        <credential name="secret" secret="0aa31d98-e0aa-404c-b6e0-e771dba1e798" />
    </secure-deployment>
</subsystem>
```

1 A deployment: **customer-portal.war**

2 Another deployment: **product-portal.war**

Additional resources

- OpenID Connect specification
- **elytron-oidc-client** subsystem attributes
- OpenID Connect Libraries

2.3.2. Creating an OIDC client in Red Hat build of Keycloak

Create an OpenID Connect (OIDC) client in Red Hat build of Keycloak to use with JBoss EAP to secure applications.

The following procedure outlines the minimum steps required to get started with securing applications deployed to JBoss EAP with Red Hat build of Keycloak for testing purposes. For detailed configurations, see Managing OpenID Connect clients in the Red Hat build of Keycloak Server Administration Guide.

Prerequisites

- You have created a realm and defined users in Red Hat build of Keycloak.
  For more information, see Creating a realm and users in JBoss EAP

Procedure

1. Navigate to the Red Hat build of Keycloak Admin Console.
2. Create a client.
   a. Click **Clients**, then click **Create client**.

b. Ensure that **Client type** is set to **OpenID Connect**.

c. Enter a client ID. For example, **jbeap-oidc**.

d. Click **Next**.

e. In the **Capability Config** tab, ensure that **Authentication Flow** is set to **Standard flow** and **Direct access grants**.

f. Click **Next**.

g. In the **Login settings** tab, enter the value for **Valid redirect URIs**. Enter the URL where the page should redirect after successful authentication, for example, **http://localhost:8080/simple-webapp-example/secured/***.

h. Click **Save**.

3. View the adapter configuration.

a. Click **Action**, then **Download adapter config**.

b. Select **Keycloak OIDC JSON** as the **Format Option** to see the connection parameters.

```
{
  "realm": "example_realm",
  "auth-server-url": "http://localhost:8180/",
  "ssl-required": "external",
  "resource": "jbeap-oidc",
  "public-client": true,
  "confidential-port": 0
}
```

When configuring your JBoss EAP application to use Red Hat build of Keycloak as the identity provider, you use the parameters as follows:

```
"provider-url" : "http://localhost:8180/realms/example_realm",
"ssl-required": "external",
"client-id": "jbeap-oidc",
"public-client": true,
"confidential-port": 0
```

Next steps

- Securing a web application using OpenID Connect

Additional resources

- Securing Applications and Services Guide

2.3.3. Securing a web application using OpenID Connect

You can secure an application by either updating its deployment configuration or by configuring the **elytron-oidc-client subsystem**.

If you use the application created in the procedure, Creating a web application , the value of the Principal

comes from the ID token from the OpenID provider. By default, the Principal is the value of the "sub" claim from the token. You can also use the value of “email”, “preferred_username”, “name”, “given_name”, “family_name”, or “nickname” claims as the Principal. Specify which claim value from the ID token is to be used as the Principal in one of the following places:

- The **elytron-oidc-client** subsystem attribute **principal-attribute**.
- **The oidc.json file**.

There are two ways in which you can configure applications to use OIDC:

- By configuring the **elytron-oidc-client** subsystem.
  Use this method if you do not want to add configuration to the application deployment.
- By updating the deployment configuration
  Use this method if you do not want to add configuration to the server and prefer to keep the configuration within the application deployment.

Prerequisites

- You have deployed applications on JBoss EAP.

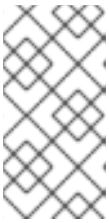
Procedure

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

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

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">

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

        <auth-constraint>
            <role-name>Admin</role-name>  (1)
        </auth-constraint>
    </security-constraint>

    <security-role>
        <role-name>*</role-name>
    </security-role>
</web-app>
```

(1) Only allow the users with the role **Admin** to access the application. To allow users with any role to access the application, use the wildcard ****** as the value for **role-name**.

2. To secure the application with OpenID Connect, either update the deployment configuration or configure the **elytron-oidc-client** subsystem.



NOTE

If you configure OpenID Connect in both the deployment configuration and the **elytron-oidc-client** subsystem, the configuration in the **elytron-oidc-client** subsystem **secure-deployment** attribute takes precedence over the configuration in the application deployment descriptor.

- Updating the deployment configuration.

  i. Add login configuration to the application's **web.xml** specifying authentication method as OIDC.

```
<web-app>
...
   <login-config>
      <auth-method>OIDC</auth-method>     (1)
   </login-config>
...
</web-app>
```

(1) Use OIDC to secure the application.

  ii. Create a file **oidc.json** in the **WEB-INF** directory, like this:

```
{
  "provider-url" : "http://localhost:8180/realms/example_realm",
  "ssl-required": "external",
  "client-id": "jbeap-oidc",
  "public-client": true,
  "confidential-port": 0
}
```

- Configuring the **elytron-oidc-client** subsystem:

  - To secure your application, use the following management CLI command:

```
/subsystem=elytron-oidc-client/secure-deployment=simple-oidc-example.war/:add(client-id=jbeap-oidc,provider-url=http://localhost:8180/realms/example_realm,public-client=true,ssl-required=external)
```

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

```
$ mvn package
```

4. Deploy the application.

```
$ mvn wildfly:deploy
```

Verification

1. In a browser, navigate to **http://localhost:8080/simple-webapp-example/secured**. You are redirected to Red Hat build of Keycloak login page.
2. You can log in with your credentials for the user you defined in Red Hat build of Keycloak.

Your application is now secured using OIDC.

Additional resources

- **elytron-oidc-client** subsystem attributes

2.4. SECURING APPLICATIONS WITH SAML

You can use the Galleon layers provided by the Keycloak SAML adapter feature pack to secure web applications with Security Assertion Markup Language (SAML).

For information about Keycloak SAML adapter feature pack, see Keycloak SAML adapter feature pack for securing applications using SAML.

To secure applications with SAML, follow these procedures:

- Securing web applications using SAML

2.4.1. Application security with SAML in JBoss EAP

Keycloak SAML adapter Galleon pack is a Galleon feature pack that includes three layers: **keycloak-saml**, **keycloak-client-saml**, and **keycloak-client-saml-ejb**. Use the layers in the feature pack to install the necessary modules and configurations in JBoss EAP to use Red Hat build of Keycloak as an identity provider for single sign-on using Security Assertion Markup Language (SAML).

The following table describes the use cases for each layer.

Layer	Applicable for	Description
keycloak-saml	OpenShift	Use this layer for Source to Image (s2i) with automatic registration of the SAML client. You must use this layer along with the **cloud-default-config** layer.
keycloak-client-saml	Bare metal, OpenShift	Use this layer for web-applications on bare metal, and for Source to Image (s2i) with **keycloak-saml** subsystem configuration provided in a CLI script or in the deployment configuration.
keycloak-client-saml-ejb	Bare metal	Use this layer for applications where you want to propagate identities to Jakarta Enterprise Beans.

To enable the use of SAML, you can configure either the **keycloak-saml** subsystem or an application itself.

Deployment configuration
To secure an application with SAML by using a deployment descriptor, update the application's deployment configuration as follows:

- Set the **auth-method** property to **SAML** in the application deployment descriptor **web.xml** file.

  Example deployment descriptor update

```
<login-config>
    <auth-method>SAML</auth-method>
</login-config>
```

- Create a file called **keycloak-saml.xml** in the **WEB-INF** directory with the SAML configuration information. You can obtain this file from the SAML provider.

  Example keycloak-saml.xml

```
<keycloak-saml-adapter>
    <SP entityID=""
        sslPolicy="EXTERNAL"
        logoutPage="SPECIFY YOUR LOGOUT PAGE!">
        <Keys>
            <Key signing="true">
                <PrivateKeyPem>PRIVATE KEY NOT SET UP OR KNOWN</PrivateKeyPem>
                <CertificatePem>...</CertificatePem>
            </Key>
        </Keys>
        <IDP entityID="idp"
             signatureAlgorithm="RSA_SHA256"
             signatureCanonicalizationMethod="http://www.w3.org/2001/10/xml-exc-c14n#">
            <SingleSignOnService signRequest="true"
                                 validateResponseSignature="true"
                                 validateAssertionSignature="false"
                                 requestBinding="POST"

bindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"/>
            <SingleLogoutService signRequest="true"
                                 signResponse="true"
                                 validateRequestSignature="true"
                                 validateResponseSignature="true"
                                 requestBinding="POST"
                                 responseBinding="POST"

postBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"

redirectBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"/>
        </IDP>
    </SP>
</keycloak-saml-adapter>
```

  The values of **PrivateKeyPem**, and **CertificatePem** are unique for each client.

Subsystem configuration
You can secure applications with SAML by configuring the **keycloak-saml** subsystem. You can obtain the client configuration file containing the subsystem configuration commands from Red Hat build of Keycloak. For more information, see Generating client adapter config .

2.4.2. Creating a SAML client in Red Hat build of Keycloak

Create a Security Assertion Markup Language (SAML) client in Red Hat build of Keycloak to use with JBoss EAP to secure applications.

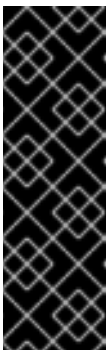
The following procedure outlines the minimum steps required to get started with securing applications deployed to JBoss EAP with Red Hat build of Keycloak for testing purposes. For detailed configurations, see Creating a SAML client in the Red Hat build of Keycloak Server Administration Guide.

Prerequisites

- You have created a realm and defined users in Red Hat build of Keycloak.
  For more information, see Creating a realm and users in JBoss EAP

Procedure

1. Navigate to the Red Hat build of Keycloak Admin Console.
2. Create a client.
   a. Click **Clients**, then click **Create client**.
   b. Select **SAML** as the **Client type**.
   c. Enter the URL for the application you want to secure as the **Client ID**. For example, **http://localhost:8080/simple-webapp-example/secured/**.



IMPORTANT

The client ID must exactly match the URL of your application. If the client ID does not match, you get an error similar to the following:

```
2023-05-17 19:54:31,586 WARN  [org.keycloak.events] (executor-thread-
0) type=LOGIN_ERROR, realmId=eba0f106-389f-4216-a676-
05fcd0c0c72e, clientId=null, userId=null, ipAddress=127.0.0.1,
error=client_not_found, reason=Cannot_match_source_hash
```

   d. Enter a client name. For example, **jbeap-saml**.
   e. Click **Next**.
   f. Enter the following information:
      - **Root URL**: The URL for your application, for example, **http://localhost:8080/simple-webapp-example/**.
      - **Home URL**: The URL for your application, for example, **http://localhost:8080/simple-webapp-example/**.

IMPORTANT

If you do not set the Home URL, **SP entityID** in the client configuration remains blank and causes errors.

- If using the management CLI commands, you get the following error:

```
Can't reset to root in the middle of the path @72
```

You can resolve the error by defining the value for **SP entityID** in the respective configuration files.

- **Valid Redirect URIs**: The URIs that are allowed after a user logs in, for example, **http://localhost:8080/simple-webapp-example/secured/***.
- **Master SAML Processing URL**: The URL for your application followed by **saml**. For example, **http://localhost:8080/simple-webapp-example/saml**.

IMPORTANT

If you do not append **saml** to the URL, you get a redirection error.

For more information, see Creating a SAML client .

You can now use the configured client to secure web applications deployed on JBoss EAP. For more information, see Securing web applications using SAML .

Next steps

- Securing web applications using SAML

Additional resources

- Red Hat build of Keycloak Server Administration Guide

2.4.3. Securing web applications using SAML

The Keycloak SAML adapter feature pack provides two layers for non-OpenShift deployments: **keycloak-client-saml**, and **keycloak-client-saml-ejb**. Use the **keycloak-client-saml** layer to secure servlet based-web applications, and the **keycloak-client-saml-ejb** to secure Jakarta Enterprise Beans applications.

There are two ways in which you can configure applications to use SAML:

- By configuring the **keycloak-saml** subsystem.
  Use this method if you do not want to add configuration to the application deployment.
- By updating the deployment configuration
  Use this method if you do not want to add configuration to the server and prefer to keep the configuration within the application deployment.

Prerequisites

- A SAML client has been created in Red Hat build of Keycloak.

For more information, see Creating a SAML client in Red Hat build of Keycloak .

- JBoss EAP has been installed by using the **jboss-eap-installation-manager**.
  For more information, see Installing JBoss EAP 8.0 using the **jboss-eap-installation-manager** in the *Red Hat JBoss Enterprise Application Platform Installation Methods* guide.

Procedure

1. Add the required Keycloak SAML adapter layer to the server by using **jboss-eap-installation-manager**. Following are the details of the available layers:
   - Feature pack: **org.keycloak:keycloak-saml-adapter-galleon-pack**.
   - Layers:
     - **keycloak-client-saml**: Use this layer to secure servlets.
     - **keycloak-client-saml-ejb**: Use this layer to propagate identities from servlets to Jakarta Enterprise Beans.
       For information about adding feature packs and layers in JBoss EAP, see Adding Feature Packs to existing JBoss EAP Servers using the jboss-eap-installation-manager in the *Red Hat JBoss Enterprise Application Platform Installation Methods* guide.
2. Configure the application's **web.xml** to protect the application resources.

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

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">

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

        <auth-constraint>
            <role-name>Admin</role-name>  (1)
        </auth-constraint>
    </security-constraint>

    <security-role>
        <role-name>*</role-name>
    </security-role>
</web-app>
```

(1) Only allow the users with the role **Admin** to access the application. To allow users with any role to access the application, use the wildcard ** as the value for **role-name**.

3. Secure your applications with SAML either by using the management CLI, or by updating the application deployment.

- By updating the application deployment.

  a. Add login configuration to the application's **web.xml** specifying authentication method as SAML.

```
<web-app>
...
   <login-config>
      <auth-method>SAML</auth-method>    (1)
   </login-config>
...
</web-app>
```

(1) Use SAML to secure the application.

  b. Download the configuration **keycloak-saml.xml** file from Red Hat build of Keycloak and save it in the **WEB-INF**/ directory of your application.
  For more information, see Generating client adapter config .

  Example keycloak-saml.xml

```
<keycloak-saml-adapter>
   <SP entityID=""
      sslPolicy="EXTERNAL"
      logoutPage="SPECIFY YOUR LOGOUT PAGE!">
      <Keys>
         <Key signing="true">
            <PrivateKeyPem>PRIVATE KEY NOT SET UP OR
KNOWN</PrivateKeyPem>
            <CertificatePem>...</CertificatePem>
         </Key>
      </Keys>
      <IDP entityID="idp"
          signatureAlgorithm="RSA_SHA256"
          signatureCanonicalizationMethod="http://www.w3.org/2001/10/xml-exc-
c14n#">
         <SingleSignOnService signRequest="true"
                              validateResponseSignature="true"
                              validateAssertionSignature="false"
                              requestBinding="POST"

bindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"/>
         <SingleLogoutService signRequest="true"
                              signResponse="true"
                              validateRequestSignature="true"
                              validateResponseSignature="true"
                              requestBinding="POST"
                              responseBinding="POST"

postBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"

redirectBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"
/>
```

```
        </IDP>
    </SP>
</keycloak-saml-adapter>
```

The values of **PrivateKeyPem**, and **CertificatePem** are unique for each client.

- By using the management CLI.

  a. Download the client configuration file **keycloak-saml-subsystem.cli** from Red Hat build of Keycloak.
  For more information, see Generating client adapter config .

  Example keycloak-saml-subsystem.cli

```
/subsystem=keycloak-saml/secure-deployment=YOUR-WAR.war/:add

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-
example/"/:add(sslPolicy=EXTERNAL,logoutPage="SPECIFY YOUR LOGOUT
PAGE!"

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-
example/"/Key=KEY1:add(signing=true, \
PrivateKeyPem="...", CertificatePem="...")

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-example/"/IDP=idp/:add( \
    SingleSignOnService={ \
        signRequest=true, \
        validateResponseSignature=true, \
        validateAssertionSignature=false, \
        requestBinding=POST, \
        bindingUrl=http://localhost:8180/realms/example-saml-realm/protocol/saml}, \
    SingleLogoutService={ \
        signRequest=true, \
        signResponse=true, \
        validateRequestSignature=true, \
        validateResponseSignature=true, \
        requestBinding=POST, \
        responseBinding=POST, \
        postBindingUrl=http://localhost:8180/realms/example-saml-realm/protocol/saml,
\
        redirectBindingUrl=http://localhost:8180/realms/example-saml-
realm/protocol/saml} \
)

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-example/"/IDP=idp/:write-
attribute(name=signatureAlgorithm,value=RSA_SHA256)

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-example/"/IDP=idp/:write-
attribute(name=signatureCanonicalizationMethod,value=http://www.w3.org/2001/10/xml-exc-c14n#)
```

The values of **PrivateKeyPem**, and **CertificatePem** are unique for each client.

b. Update every occurrence of **YOUR-WAR.war** in the client configuration file with the name of your application WAR, for example **simple-webapp-example.war**.

NOTE

The generated CLI script has a missing **)** at the end of the second statement:

```
/subsystem=keycloak-saml/secure-deployment=YOUR-WAR.war/SP=""/:add(sslPolicy=EXTERNAL,logoutPage="SPECIFY YOUR LOGOUT PAGE!"
```

You must add the missing **)**

c. Configure JBoss EAP by running **keycloak-saml-subsystem.cli** script using the management CLI.

```
$ <EAP_HOME>/bin/jboss-cli.sh -c --file=<path_to_the_file>/keycloak-saml-subsystem.cli
```

4. Deploy the application.

```
$ mvn wildfly:deploy
```

Verification

1. In a browser, navigate to the application URL. For example, **http://localhost:8080/simple-webapp-example/secured**.
   You are redirected to the Red Hat build of Keycloak login page.
2. You can log in with your credentials for the user you defined in Red Hat build of Keycloak.

Your application is now secured using SAML.

Additional resources

- Keycloak SAML adapter feature pack for securing applications using SAML

CHAPTER 3. PROPAGATING AN IDENTITY FROM A SERVLET TO A JAKARTA ENTERPRISE BEAN WHEN USING OIDC

You can propagate the security identity obtained from an OpenID Connect (OIDC) provider from a Servlet to Jakarta Enterprise Beans in two ways:

- Using the same virtual security domain to secure both Servlet and Jakarta Enterprise Beans.
- Propagating the identity from a virtual security domain associated with a Servlet to the security domain securing Jakarta Enterprise Beans.

3.1. IDENTITY PROPAGATION TO JAKARTA ENTERPRISE BEANS WHEN USING OIDC

When you secure an application with OpenID Connect (OIDC), the **elytron-oidc-client** subsystem automatically creates a virtual security domain for you. You can propagate the security identity in the virtual security domain, obtained from the OIDC provider, to Jakarta Enterprise Beans that your application invokes.

The following table illustrates the required configurations depending on the security domain you use and how the applications are deployed.

Security domain to use to secure Jakarta Enterprise Beans	Servlet and Jakarta Enterprise Beans are in the same WAR or EAR	Servlet and Jakarta Enterprise Beans are in different WAR or EAR
Virtual security domain	No configuration is required. The virtual security domain automatically outflows the security identity to the Jakarta Enterprise Beans provided that no security domain configuration has been explicitly specified for the Jakarta Enterprise Beans.	Configure as follows: • Create a **virtual-security-domain** resource. • Add a **@SecurityDomain** annotation to your Jakarta Enterprise Beans that references the name of a **virtual-security-domain** resource. For more information, see Securing Jakarta Enterprise Beans applications using virtual security domain.

Security domain to use to secure Jakarta Enterprise Beans	Servlet and Jakarta Enterprise Beans are in the same WAR or EAR	Servlet and Jakarta Enterprise Beans are in different WAR or EAR
A different security domain	To outflow a security identity from a virtual security domain to another security domain, you must configure the following resources: • **virtual-security-domain**: Specify that security identities established by a virtual security domain should automatically be outflowed to other security domains. • **security-domain**: Indicate that it should trust security identities established by the virtual security domain you configured. For more information, see Propagating identity from virtual security domain to a security domain.	

3.2. SECURING JAKARTA ENTERPRISE BEANS APPLICATIONS USING VIRTUAL SECURITY DOMAIN

You can use the virtual security domain created by the **elytron-oidc-client** subsystem to secure Jakarta Enterprise Beans both when the Jakarta Enterprise Beans is located in the same deployment as the Servlet that invokes it, and when they are located in different deployments.

If the Jakarta Enterprise Beans is located in the same deployment as the Servlet invoking it, no configuration is required to outflow a security identity from the Servlet to the Jakarta Enterprise Beans.

Follow the steps in this procedure to outflow a security identity from a Servlet to Jakarta Enterprise Beans that are located in different deployments.

Prerequisites

- You have secured the application from which you invoke the Jakarta Enterprise Beans using an OpenID Connect (OIDC) provider.
- You have created the Jakarta Enterprise Beans to secure.

Procedure

1. Create a **virtual-security-domain** resource referencing the WAR that contains the Servlet secured with OIDC or the EAR that contains a subdeployment that is secured with OIDC.

   #### Syntax

   ```
   /subsystem=elytron/virtual-security-domain=<deployment_name>:add()
   ```

   #### Example

   ```
   /subsystem=elytron/virtual-security-domain=simple-ear-example.ear:add()
   ```

2. Add **org.jboss.ejb3.annotation.SecurityDomain** annotation in the Jakarta Enterprise Beans application referencing the virtual security domain resource to use to secure the application.

Syntax

```
@SecurityDomain("<deployment_name>")
```

Example

```
...
@SecurityDomain("simple-ear-example.ear")
@Remote(RemoteHello.class)
@Stateless
public class RemoteHelloBean implements RemoteHello {

  @Resource
  private SessionContext context;

  @Override
  public String whoAmI() {
    return context.getCallerPrincipal().getName();
  }

}
```

If you invoke this Jakarta Enterprise Beans from a Servlet secured with OIDC, the principal returned by **whoAmI ()** will match the principal the Servlet obtained from the OIDC provider.

3. Deploy the Jakarta Enterprise Beans.

Example

```
$ mvn wildfly:deploy
```

Additional resources

- **virtual-security-domain** attributes

3.3. PROPAGATING IDENTITY FROM VIRTUAL SECURITY DOMAIN TO A SECURITY DOMAIN

You can propagate the security identity from a virtual security domain, obtained from an OpenID Connect (OIDC) provider to a different security domain. You might want to do this if you want the security identity's roles to be determined by the security domain you propagate the identity to and not the virtual security domain.

The steps in the following procedure apply both when the Servlet invoking a Jakarta Enterprise Beans and the Jakarta Enterprise Beans are in the same deployment and when they are in separate deployments.

Prerequisites

- You have secured the application from which you invoke the Jakarta Enterprise Beans using an OIDC provider.
- You have created the Jakarta Enterprise Beans to secure.

- You secured the Jakarta Enterprise Beans with a security domain.

Procedure

1. Create a **virtual-security-domain** resource referencing the WAR that contains the Servlet secured with OIDC or the EAR that contains a subdeployment that is secured with OIDC.

   Syntax

   ```
   /subsystem=elytron/virtual-security-domain=<deployment_name>:add(outflow-security-domains=[<domain_to_propagate_to>])
   ```

   Example

   ```
   /subsystem=elytron/virtual-security-domain=simple-ear-example.ear:add(outflow-security-domains=[exampleEJBSecurityDomain])
   ```

2. Update the security domain configuration for the Jakarta Enterprise Beans to trust the **virtual-security-domain**.

   Syntax

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=trusted-virtual-security-domains,value=[<deployment_name>])
   ```

   Example

   ```
   /subsystem=elytron/security-domain=exampleEJBSecurityDomain:write-attribute(name=trusted-virtual-security-domains,value=[simple-ear-example.ear])
   ```

3. Reload the server.

   ```
   reload
   ```

4. Deploy the Jakarta Enterprise Beans.

   Example

   ```
   $ mvn wildfly:deploy
   ```

Additional resources

- **security-domain** attributes
- **virtual-security-domain** attributes

CHAPTER 4. SECURING THE JBOSS EAP MANAGEMENT CONSOLE WITH AN OPENID PROVIDER

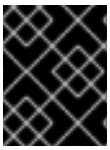
You can secure the JBoss EAP management console with an external identity provider, such as Red Hat build of Keycloak, using OIDC. By using an external identity provider, you can delegate authentication to the identity provider.

To secure the JBoss EAP management console using OIDC, follow these procedures:

- Configuring Red Hat build of Keycloak to secure JBoss EAP management console
- Securing the JBoss EAP management console using OpenID Connect

4.1. JBOSS EAP MANAGEMENT CONSOLE SECURITY WITH OIDC

You can secure the JBoss EAP management console with OpenID Connect (OIDC) by configuring an OIDC provider, such as Red Hat build of Keycloak, and the **elytron-oidc-client** subsystem.



IMPORTANT

Securing the management console of JBoss EAP running as a managed domain with OIDC is not supported.

JBoss EAP management console security with OIDC works as follows:

- When you configure a **secure-server** resource in the **elytron-oidc-client** subsystem, the JBoss EAP management console redirects to the OIDC provider login page for login.
- JBoss EAP then uses the **secure-deployment** resource configuration to secure the management interface with bearer token authentication.

NOTE

OIDC relies on accessing a web application in a browser. Therefore, the JBoss EAP management CLI can't be secured with OIDC.

RBAC support

You can configure and assign roles in the OIDC provider to implement role-based access control (RBAC) to the JBoss EAP management console. JBoss EAP includes or excludes the users roles for RBAC as defined in the JBoss EAP RBAC configuration. For more information about RBAC, see Role-Based Access Control in the JBoss EAP 7.4 *Security Architecture* guide.

Additional resources

- Configuring Red Hat build of Keycloak to secure JBoss EAP management console
- Securing the JBoss EAP management console using OpenID Connect

4.2. CONFIGURING RED HAT BUILD OF KEYCLOAK TO SECURE JBOSS EAP MANAGEMENT CONSOLE

Configure the required users, roles, and clients in the OpenID Connect (OIDC) provider to secure the JBoss EAP management console.

Two clients are required to secure the management console with OIDC. The clients must be configured as follows:

- A client configured for standard flow.
- A client configured as bearer-only client.

The following procedure outlines the minimum steps required to get started with securing the JBoss EAP management console using OIDC for testing purposes. For detailed configurations, see the Red Hat build of Keycloak documentation.

Prerequisites

- You have administrator access to Red Hat build of Keycloak.
- Red Hat build of Keycloak is running.

Procedure

1. Create a realm in Red Hat build of Keycloak using the Red Hat build of Keycloak admin console; for example, **example_jboss_infra**. You will use this realm to create the required users, roles, and clients.
   For more information, see Creating a realm.
2. Create a user. For example, **user1**.
   For more information, see Creating users.
3. Create a password for the user. For example, **passwordUser1**.
   For more information, see Setting a password for a user .
4. Create a role. For example, **Administrator**.
   To enable role-based access control (RBAC) in JBoss EAP, the name should be one of the standard RBAC roles like **Administrator**. For more information about RBAC in JBoss EAP, see Role-Based Access Control in the JBoss EAP 7.4 *Security Architecture* guide.

   For more information about creating roles in Red Hat build of Keycloak, see Creating a realm role.
5. Assign roles to users.
   For more information, see Assigning role mappings.
6. Create an OpenID Connect client, for example, **jboss-console**.
   - Ensure that the following capability configuration values are checked:
     - Standard flow
     - Direct access grants
   - Set the following attributes at the minimum on the **Login settings** page:
     - Set **Valid Redirect URIs** to the management console URI. For example, http://localhost:9990.

- Set **Web Origins** to the management console URI. For example, http://localhost:9990.

7. Create another OpenID Connect client, for example, **jboss-management**, as a bearer-only client.
   - In capability configuration, uncheck the following options:
     - Standard flow
     - Direct access grants
   - You do not need to specify any fields on the **Login settings** page.

You can now secure the JBoss EAP management console by using the clients you defined. For more information, see Securing the JBoss EAP management console using OpenID Connect.

Additional resources

- JBoss EAP management console security with OIDC

4.3. SECURING THE JBOSS EAP MANAGEMENT CONSOLE USING OPENID CONNECT

When you secure the JBoss EAP management console using OpenID Connect (OIDC), JBoss EAP redirects to the OIDC provider for users to log in to the management console.

Prerequisites

- You have configured the required clients in the OIDC provider.
  For more information, see Configuring Red Hat build of Keycloak to secure JBoss EAP management console.

Procedure

1. Configure the OIDC provider in the **elytron-oidc-client** subsystem.

   Syntax

   ```
   /subsystem=elytron-oidc-client/provider=keycloak:add(provider-url=<OIDC_provider_URL>)
   ```

   Example

   ```
   /subsystem=elytron-oidc-client/provider=keycloak:add(provider-
   url=http://localhost:8180/realms/example_jboss_infra)
   ```

2. Create a **secure-deployment** resource called **wildfly-management** to protect the management interface.

   Syntax

   ```
   /subsystem=elytron-oidc-client/secure-deployment=wildfly-
   management:add(provider=<OIDC_provider_name>,client-
   id=<OIDC_client_name>,principal-attribute=<attribute_to_use_as_principal>,bearer-
   only=true,ssl-required=<internal_or_external>)
   ```

-

Example

```
/subsystem=elytron-oidc-client/secure-deployment=wildfly-management:add(provider=keycloak,client-id=jboss-management,principal-attribute=preferred_username,bearer-only=true,ssl-required=EXTERNAL)
```

3. OPTIONAL: You can enable role-based access control (RBAC) using the following commands.

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

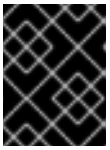
4. Create a **secure-server** resource called **wildfly-console** that references the **jboss-console** client.

Syntax

```
/subsystem=elytron-oidc-client/secure-server=wildfly-console:add(provider=<OIDC_provider_name>,client-id=<OIDC_client_name>,public-client=true)
```

Example

```
/subsystem=elytron-oidc-client/secure-server=wildfly-console:add(provider=keycloak,client-id=jboss-console,public-client=true)
```



IMPORTANT

The JBoss EAP management console requires that the **secure-server** resource be specifically named **wildfly-console**.

Verification

1. Access the management console. By default, the management console is available at http://localhost:9990.
   You are redirected to the OIDC provider.
2. Log in with the credentials of the user you created in the OIDC provider.

The JBoss EAP management console is now secured with OIDC.

Additional resources

- JBoss EAP management console security with OIDC
- **elytron-oidc-client** subsystem attributes

CHAPTER 5. REFERENCE

5.1. ELYTRON-OIDC-CLIENT SUBSYSTEM ATTRIBUTES

The **elytron-oidc-client** subsystem provides attributes to configure its behavior.

Table 5.1. elytron-oidc-client subsystem attributes

Attribute	Description
provider	Configuration for an OpenID Connect provider.
secure-deployment	A deployment secured by an OpenID Connect provider.
realm	Configuration for a Red Hat build of Keycloak realm. This is provided for convenience. You can copy the configuration in the keycloak client adapter and use it here. Using the **provider** is recommended instead.

Use the three **elytron-oidc-client** attributes for the following purposes:

- **provider**: For configuring the OpenID Connect provider. For more information, see **provider** attributes.
- **secure-deployment**: For configuring the deployment secured by an OpenID Connect. For more information, see **secure-deployment** attributes
- **realm**: For configuring Red Hat build of Keycloak. For more information, see **realm** attributes. The use of **realm** is not recommended. It is provided for convenience. You can copy the configuration in the keycloak client adapter and use it here. Using the **provider** attribute is recommended instead.

Table 5.2. provider attributes

Attribute	Default value	Description
allow-any-hostname	**false**	If you set the value to **true**, hostname verification is skipped when communicating with the OpenID provider. This is useful when testing. Do not set this to **true** in a production environment.
always-refresh-token		If set to **true**, the subsystem refreshes the token every time your application receives a web request, and sends a new request to the OpenID provider to obtain a new access token.

Attribute	Default value	Description
auth-server-url		The base URL of the Red Hat build of Keycloak realm authorization server. If you use this attribute, you must also define the **realm** attribute. You can alternatively use the **provider-url** attribute to provide both base URL and the realm in a single attribute.
autodetect-bearer-only	**false**	Set whether to automatically detect bearer-only requests. When a bearer-only request is received and **autodetect-bearer-only** is set to **true**, the application cannot participate in browser logins. Use this attribute to automatically detect Simple Object Access Protocol (SOAP) or REST clients based on headers like **X-Requested-With**, **SOAPAction** or **Accept**.
client-id		The client-id of JBoss EAP registered with the OpenID provider.
client-key-password		If you specify **client-keystore**, specify it's password in this attribute.
client-keystore		If your application communicates with the OpenID provider over HTTPS, set the path to the client keystore in this attribute.
client-keystore-password		If you specify the **client keystore**, provide the password for accessing it in this attribute.
confidential-port	**8443**	Specify the confidential port (SSL/TLS) used by the OpenID provider.
connection-pool-size		Specify the connection pool size to be used when communicating with the OpenID provider.
connection-timeout-millis	**-1L**	Specify the timeout for establishing a connection with the remote host in milliseconds. The minimum is **-1L**, and the maximum **2147483647L**.**-1L** indicates that the value is undefined, which is the default.
connection-ttl-millis	**-1L**	Specify the amount of time in milliseconds for the connection to be kept alive. The minimum is **-1L**, and the maximum **2147483647L**. **-1L** indicates that the value is undefined, which is the default.
cors-allowed-headers		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Allow-Headers** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.

Attribute	Default value	Description
cors-allowed-methods		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the Access-Control-Allow-Methods header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-exposed-headers		If CORS is enabled, this sets the value of the Access-Control-Expose-Headers header. This should be a comma-separated string. This is optinal. If not set, this header is not returned in CORS responses.
cors-max-age		Set the value for Cross-Origin Resource Sharing (CORS) Max-Age header. The value can be between **-1L** and **2147483647L**. This attribute only takes effect if **enable-cors** is set to **true**.
disable-trust-manager		Specify whether or not to make use of a trust manager when communicating with the OpenID provider over HTTPS.
enable-cors	**false**	Enable Red Hat build of Keycloak Cross-Origin Resource Sharing (CORS) support.
expose-token	**false**	If set to **true**, an authenticated browser client can obtain the signed access token, through a Javascript HTTP invocation, via the URL **root/k_query_bearer_token**. This is optional. This is specific to Red Hat build of Keycloak.
ignore-oauth-query-parameter	**false**	Disable query parameter parsing for **access_token**.
principal-attribute		Specify which claim value from the ID token to use as the principal for the identity
provider-url		Specify the OpenID provider URL.
proxy-url		Specify the URL for the HTTP proxy if you use one.
realm-public-key		Specify the public key of the realm.
register-node-at-startup	**false**	If set to **true**, a registration request is sent to Red Hat build of Keycloak. This attribute is useful only when your application is clustered.
register-node-period		Specify how often to re-register the node.
socket-timeout-millis		Specify the timeout for socket waiting for data in milliseconds.

Attribute	Default value	Description
ssl-required	**external**	Specify whether communication with the OpenID provider should be over HTTPS. The value can be one of the following: • **all** - all communication happens over HTTPS. • **external** - Only the communication with external clients happens over HTTPs. • **none** - HTTPs is not used.
token-signature-algorithm	**RS256**	Specify the token signature algorithm used by the OpenID provider. The supported algorithms are: • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**
token-store		Specify cookie or session storage for auth-session data.
truststore		Specify the truststore used for client HTTPS requests.
truststore-password		Specify the truststore password.
verify-token-audience	**false**	If set to **true**, then during bearer-only authentication,verify if token contains this client name (**resource**) as an audience.

Table 5.3. **secure-deployment** attributes

Attribute	Default value	Description
allow-any-hostname	**false**	If you set the value to **true**, hostname verification is skipped when communicating with the OpenID provider. This is useful when testing. Do not set this to **ture** in a production environment.
always-refresh-token		If set to **true**, JBoss EAP refreshes tokens on every web request.

Attribute	Default value	Description
auth-server-url		The base URL of the Red Hat build of Keycloak realm authorization server You can alternatively use the **provider-url** attribute.
autodetect-bearer-only	**false**	Set whether to automatically detect bearer-only requests. When a bearer-only request is received and **autodetect-bearer-only** is set to **true**, the application cannot participate in browser logins.
bearer-only	**false**	Set this to **true** to secure the application with Bearer Token authentication. When Bearer Token authentication is enabled, users are not redirected to the OpenID provider to log in; instead, the **elytron-oidc-client** subsystem attempts to verify the user's bearer token.
client-id		The unique identifier for a client registered in the OpenID provider.
client-key-password		If you specify **client-keystore**, specify its password in this attribute.
client-keystore		If your application communicates with the OpenID provider over HTTPS, set the path to the client keystore in this attribute.
client-keystore-password		If you specify the **client keystore**, provide the password for accessing it in this attribute.
confidential-port	**8443**	Specify the confidential port (SSL/TLS) used by OpenID provider.
connection-pool-size		Specify the connection pool size to be used when communicating with the OpenID provider.

Attribute	Default value	Description
connection-timeout-millis	**-1L**	Specify the timeout for establishing a connection with the remote host in milliseconds. The minimum is **-1L**, and the maximum **2147483647L**. **-1L** indicates that the value is undefined, which is the default.
connection-ttl-millis	**-1L**	Specify the amount of time in milliseconds for the connection to be kept alive. The minimum is **-1L**, and the maximum **2147483647L**. -1L indicates that the value is undefined, which is the default.
cors-allowed-headers		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Allow-Headers** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-allowed-methods		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Allow-Methods** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-exposed-headers		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Expose-Headers** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-max-age		Set the value for Cross-Origin Resource Sharing (CORS) Max-Age header. The value can be between **-1L** and **2147483647L**. This attribute only takes effect if **enable-cors** is set to **true**.

Attribute	Default value	Description
credential		Specify the credential to use to communicate with the OpenID provider.
disable-trust-manager		Specify whether or not to make use of a trust manager when communicating with the OpenID provider over HTTPS.
enable-cors	**false**	Enable Red Hat build of Keycloak Cross-Origin Resource Sharing (CORS) support.
enable-basic-auth	**false**	Enable Basic Authentication to specify the credentials to be used to obtain a bearer token.
expose-token	**false**	If set to **true**, an authenticated browser client can obtain the signed access token, through a Javascript HTTP invocation, via the URL **root/k_query_bearer_token**. This is optional.This is specific to Red Hat build of Keycloak.
ignore-oauth-query-parameter	**false**	Disable query parameter parsing for **access_token**.
min-time-between-jwks-requests	If the subsystem detects a token signed by an unknown public key, JBoss EAP tries to download new public key from the **elytron-oidc-client** server. The attribute specifies the interval, in seconds, that JBoss EAP waits before subsequent download attempts.The value can be between **-1L** and **2147483647L**.	principal-attribute
	Specify which claim value from the ID token to use as the principal for the identity	provider
	Specify the OpenID provider.	provider-url
	Specify the OpenID provider URL.	proxy-url

Attribute	Default value	Description
	Specify the URL for the HTTP proxy if you use one.	public-client
false	If set to **true**, no client credentials are sent when communicating with the OpenID provider. This is optional.	realm
	The realm with which to connect in Red Hat build of Keycloak.	realm-public-key
	Specify the public key of the OpenID provider in PEM format.	redirect-rewrite-rule
	Specify the rewrite rule to apply to the redirect URI.	register-node-at-startup
false	If set to **true**, a registration request is sent to Red Hat build of Keycloak. This attribute is useful only when your application is clustered.	register-node-period
	Specify how often to re-register the node in seconds.	resource
	Specify the name of the application you are securing with OIDC. Alternatively, you can specify the **client-id**.	socket-timeout-millis
	Specify the timeout for socket waiting for data in milliseconds.	ssl-required

Attribute	Default value	Description
external	Specify whether communication with the OpenID provider should be over HTTPS. The value can be one of the following: • **all** - all communication happens over HTTPS. • **external** - Only the communication with external clients happens over HTTPs. • **none** - HTTPs is not used.	token-minimum-time-to-live
	The adapter refreshes the token if the current token is expired or is to expire within the amount of time you set in seconds.	token-signature-algorithm
RS256	Specify the token signature algorithm used by the OpenID provider. The supported algorithms are: • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**	token-store
	Specify cookie or session storage for auth-session data.	truststore
	Specify the truststore used for adapter client HTTPS requests.	truststore-password
	Specify the truststore password.	turn-off-change-session-id-on-login
false	The session id is changed by default on a successful login. Set the value to **true** to turn this off.	use-resource-role-mappings

Attribute	Default value	Description
false	Use resource-level permissions obtained from token.	verify-token-audience

Table 5.4. **secure-server** attributes

Attribute	Default value	Description
adapter-state-cookie-path		If set, this defines the path used in cookies set by the subsystem. If not set,**""** is used as the path.
allow-any-hostname	**false**	If you set the value to **true**, hostname verification is skipped when communicating with the OpenID provider. This is useful when testing. Do not set this to **true** in a production environment.
always-refresh-token		If set to **true**, the subsystem refreshes the token every time your application receives a web request, and sends a new request to the OpenID provider to obtain a new access token.
auth-server-url-for-backend-requests		Specifies the URL to use only for backend requests to invoke OpenID provider directly without having to go through a load balancer or a reverse proxy.
auth-server-url		The base URL of the Red Hat build of Keycloak realm authorization server You can alternatively use the **provider-url** attribute.

Attribute	Default value	Description
autodetect-bearer-only	**false**	Set whether to automatically detect bearer-only requests. When a bearer-only request is received and **autodetect-bearer-only** is set to **true**, the application cannot participate in browser logins. Use this attribute to automatically detect Simple Object Access Protocol (SOAP) or REST clients based on headers like **X-Requested-With**, **SOAPAction** or **Accept**.
bearer-only	**false**	Set this to **true** to secure the application with Bearer Token authentication. When Bearer Token authentication is enabled, users are not redirected to the OpenID provider to log in; instead, the **elytron-oidc-client** subsystem attempts to verify the user's bearer token.
client-id		The unique identifier for a client registered in the OpenID provider.
client-key-password		If you specify **client-keystore**, specify its password in this attribute.
client-keystore-password		If you specify the **client keystore**, provide the password for accessing it in this attribute.
client-keystore		When communicating with the OpenID provider over HTTPS, set the path to the client keystore in this attribute.
confidential-port	**8443**	Specify the confidential port (SSL/TLS) used by OpenID provider.

Attribute	Default value	Description
connection-pool-size		Specify the connection pool size to be used when communicating with the OpenID provider.
connection-timeout-millis	**-1L**	Specify the timeout for establishing a connection with the remote host in milliseconds. The minimum is **-1L**, and the maximum **2147483647L**. **-1L** indicates that the value is undefined, which is the default.
connection-ttl-millis	**-1L**	Specify the amount of time in milliseconds for the connection to be kept alive. The minimum is **-1L**, and the maximum **2147483647L**. -1L indicates that the value is undefined, which is the default.
cors-allowed-headers		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Allow-Headers** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-allowed-methods		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Allow-Methods** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-exposed-headers		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Expose-Headers** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.

Attribute	Default value	Description
cors-max-age		Set the value for Cross-Origin Resource Sharing (CORS) Max-Age header. The value can be between **-1L** and **2147483647L**. This attribute only takes effect if **enable-cors** is set to **true**.
credential		Specify the credential to use to communicate with the OpenID provider.
disable-trust-manager		Specify whether or not to make use of a trust manager when communicating with the OpenID provider over HTTPS.
enable-basic-auth	**false**	Enable Basic Authentication to specify the credentials to be used to obtain a bearer token.
enable-cors	**false**	Enable Red Hat build of Keycloak Cross-Origin Resource Sharing (CORS) support.
expose-token	**false**	If set to **true**, an authenticated browser client can obtain the signed access token, through a Javascript HTTP invocation, via the URL **root/k_query_bearer_token**. This is optional.This is specific to Red Hat build of Keycloak.
ignore-oauth-query-parameter	**false**	Disable query parameter parsing for **access_token**.
min-time-between-jwks-requests		If the subsystem detects a token signed by an unknown public key, JBoss EAP tries to download new public key from the **elytron-oidc-client** server. However, JBoss EAP deosn't try to download new public key if it has already tried it in less than the value, in seconds, that you set for this attribute. The value can be between **-1L** and **2147483647L**.

Attribute	Default value	Description
principal-attribute		Specify which claim value from the ID token to use as the principal for the identity
principal-attribute		Specify which claim value from the ID token to use as the principal for the identity.
provider		Specify the OpenID provider.
provider-url		Specify the OpenID provider URL.
proxy-url		Specify the URL for the HTTP proxy if you use one.
public-client	**false**	If set to **true**, no client credentials are sent when communicating with the OpenID provider. This is optional.
public-key-cache-ttl		The maximum interval between two requests to retrieve new public keys in seconds.
realm-public-key		Specify the public key of the OpenID provider in PEM format.
realm		The realm with which to connect in Red Hat build of Keycloak.
redirect-rewrite-rule		Specify the rewrite rule to apply to the redirect URI.
register-node-at-startup	**false**	If set to **true**, a registration request is sent to Red Hat build of Keycloak. This attribute is useful only when your application is clustered.
register-node-period		Specify how often to re-register the node in seconds.
resource		Specify the name of the application you are securing with OIDC. Alternatively, you can specify the **client-id**.

Attribute	Default value	Description
socket-timeout-millis		Specify the timeout for socket waiting for data in milliseconds.
ssl-required	**external**	Specify whether communication with the OpenID provider should be over HTTPS. The value can be one of the following: • **all** - all communication happens over HTTPS. • **external** - Only the communication with external clients happens over HTTPs. • **none** - HTTPs is not used.
token-minimum-time-to-live		The adapter refreshes the token if the current token is expired or is to expire within the amount of time you set in seconds.
token-signature-algorithm	**RS256**	Specify the token signature algorithm used by the OpenID provider. The supported algorithms are: • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**
token-store		Specify cookie or session storage for auth-session data.
truststore-password		Specify the truststore password.
truststore		Specify the truststore used for adapter client HTTPS requests.
turn-off-change-session-id-on-login	**false**	The session id is changed by default on a successful login. Set the value to **true** to turn this off.

Attribute	Default value	Description
use-resource-role-mappings	**false**	Use resource-level permissions obtained from token.
verify-token-audience	**false**	If set to **true**, then during bearer-only authentication, the adapter verifies if token contains this client name (**resource**) as an audience.

Table 5.5. **realm** attributes

Attribute	Default value	Description
allow-any-hostname	**false**	If you set the value to **true**, hostname verification is skipped when communicating with the OpenID provider. This is useful when testing. Do not set this to **ture** in a production environment.
always-refresh-token		If set to **true**, the subsystem refreshes the token every time your application receives a web request, and sends a new request to the OpenID provider to obtain a new access token.
auth-server-url		The base URL of the Red Hat build of Keycloak realm authorization server You can alternatively use the **provider-url** attribute.
autodetect-bearer-only	**false**	Set whether to automatically detect bearer-only requests. When a bearer-only request is received and **autodetect-bearer-only** is set to **true**, the application cannot participate in browser logins.
client-key-password		If you specify **client-keystore**, specify it's password in this attribute.
client-keystore		If your application communicates with the OpenID provider over HTTPS, set the path to the client keystore in this attribute.

Attribute	Default value	Description
client-keystore-password		If you specify the **client keystore**, provide the password for accessing it in this attribute.
confidential-port	**8443**	Specify the confidential port (SSL/TLS) used by Red Hat build of Keycloak.
connection-pool-size		Specify the connection pool size to be used when communicating with Red Hat build of Keycloak.
connection-timeout-millis	**-1L**	Specify the timeout for establishing a connection with the remote host in milliseconds. The minimum is **-1L**, and the maximum **2147483647L**. **-1L** indicates that the value is undefined, which is the default.
connection-ttl-millis	**-1L**	Specify the amount of time in milliseconds for the connection to be kept alive. The minimum is **-1L**, and the maximum **2147483647L**. **-1L** indicates that the value is undefined, which is the default.
cors-allowed-headers		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the Access-Control-Allow-Headers header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-allowed-methods		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Allow-Methods header**. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.

Attribute	Default value	Description
cors-exposed-headers		If Cross-Origin Resource Sharing (CORS) is enabled, this sets the value of the **Access-Control-Expose-Headers** header. This should be a comma-separated string. This is optional. If not set, this header is not returned in CORS responses.
cors-max-age		Set the value for Cross-Origin Resource Sharing (CORS) Max-Age header. The value can be between **-1L** and **2147483647L**. This attribute only takes effect if **enable-cors** is set to **true**.
disable-trust-manager		Specify whether or not to make use of a trust manager when communicating with the OpenID provider over HTTPS._
enable-cors	**false**	Enable Red Hat build of Keycloak Cross-Origin Resource Sharing (CORS) support.
expose-token	**false**	If set to **true**, an authenticated browser client can obtain the signed access token, through a Javascript HTTP invocation, via the URL **root/k_query_bearer_token**. This is optional.
ignore-oauth-query-parameter	**false**	Disable query parameter parsing for **access_token**.
principal-attribute		Specify which claim value from the ID token to use as the principal for the identity
provider-url		Specify the OpenID provider URL.
proxy-url		Specify the URL for the HTTP proxy if you use one.
realm-public-key		Specify the public key of the realm.

Attribute	Default value	Description
register-node-at-startup	**false**	If set to **true**, a registration request is sent to Red Hat build of Keycloak. This attribute is useful only when your application is clustered.
register-node-period		Specify how often to re-register the node.
socket-timeout-millis		Specify the timeout for socket waiting for data in milliseconds.
ssl-required	**external**	Specify whether communication with the OpenID provider should be over HTTPS. The value can be one of the following: ● **all** - all communication happens over HTTPS. ● **external** - Only the communication with external clients happens over HTTPs. ● **none** - HTTPs is not used.
token-signature-algorithm	**RS256**	Specify the token signature algorithm used by the OpenID provider. The supported algorithms are: ● **RS256** ● **RS384** ● **RS512** ● **ES256** ● **ES384** ● **ES512**
token-store		Specify cookie or session storage for auth-session data.
truststore		Specify the truststore used for client HTTPS requests.

Attribute	Default value	Description
truststore-password		Specify the truststore password.
verify-token-audience	**false**	If set to **true**, then during bearer-only authentication, the adapter verifies if token contains this client name (resource) as an audience.

Additional resources

- Application security with OpenID Connect in JBoss EAP
- Securing a web application using OpenID Connect

5.2. 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, which happens in the following scenarios: • The domain to outflow to does not trust this domain. • The identity being outflowed to a domain does not exist in that domain Outflowing anonymous identity clears any previously established identity 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.
trusted-virtual-security-domains	The list of virtual security domains that are trusted by this security domain.

5.3. VIRTUAL-SECURITY-DOMAIN ATTRIBUTES

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

Table 5.6. virtual-security-domain attributes

Attribute	Description
outflow-anonymous	Set this attribute to **true** to outflow anonymous identity if outflowing the security identity to a security domain is not possible, which happens in the following scenarios: • The domain to outflow to does not trust this virtual domain. • The identity being outflowed to a domain does not exist in that domain Outflowing anonymous identity has the effect of clearing any identity already established for that domain. The default value is **false**.
outflow-security-domains	The list of security domains that the security identity from this virtual domain should automatically outflow to.