Red Hat

Red Hat JBoss Enterprise Application Platform 8.0

Configuring SSL/TLS in JBoss EAP

Guide to enabling SSL/TLS in JBoss EAP to secure JBoss EAP management interfaces and deployed applications

Last Updated: 2024-02-21

Red Hat JBoss Enterprise Application Platform 8.0 Configuring SSL/TLS in JBoss EAP

Guide to enabling SSL/TLS in JBoss EAP to secure JBoss EAP management interfaces and deployed applications

Legal Notice

Abstract

Guide to enabling SSL/TLS in JBoss EAP to secure JBoss EAP management interfaces and deployed applications.

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. ENABLING ONE-WAY SSL/TLS FOR MANAGEMENT INTERFACES AND APPLICATIONS

SSL/TLS, or transport layer security (TLS), is a certificates-based security protocol that is used to secure the data transfer between two entities communicating over a network.

You can enable one-way SSL/TLS both for the JBoss EAP management interfaces and the applications deployed on JBoss EAP. For more information, see the following procedures:

- Enabling one-way SSL/TLS for management interfaces .
- Enabling one-way SSL/TLS for applications deployed on JBoss EAP .

1.1. ENABLING ONE-WAY SSL/TLS FOR MANAGEMENT INTERFACES

Enable one-way SSL/TLS for management interfaces so that the communication between JBoss EAP management interfaces and the clients connecting to the interfaces is secure.

To enable one-way SSL/TLS for management interfaces, you can use the following procedures:

- Enabling one-way SSL/TLS for management interfaces by using the wizard : Use this procedure to quickly set up SSL/TLS using a CLI-based wizard. Elytron creates the required resources for you based on your inputs to the wizard.
- Enable one-way SSL/TLS for management interfaces by using the subsystem commands : Use this procedure to configure the required resource for enabling SSL/TLS manually. Manually configuring the resources gives you more control over the server configuration.

Additionally, you can disable SSL/TLS for management interfaces using the procedure Disabling SSL/TLS for management interfaces by using the security command.

1.1.1. Enabling one-way SSL/TLS for management interfaces by using the wizard

Elytron provides a wizard to quickly set up SSL/TLS. You can either use an existing keystore containing certificates or use the keystore and self-signed certificates that the wizard generates to enable SSL/TLS. You can also obtain and use certificates from the Let’s Encrypt certificate authority by using the **--lets-encrypt** option. For information about Let’s Encrypt, see the Let’s Encrypt documentation.

Use the self-signed certificates the wizard generates to enable SSL/TLS for testing and development purposes only. For production environments always use certificate authority (CA)-signed certificates.

IMPORTANT

Do not use self-signed certificates in a production environment. Use only the certificates signed by a certificate authority (CA).

The wizard configures the following resources that are required to enable SSL/TLS for for management interfaces:

- **key-store**
- **key-manager**
- **server-ssl-context**

- The **server-ssl-context** is then applied to **http-interface**.

Elytron names each resource as *resource-type-UUID*. For example, key-store-9e35a3be-62bb-4fff-afc2-2d8d141b82bc. The universally unique identifier (UUID) helps avoid name collisions for the resources.

Prerequisites

- JBoss EAP is running.

Procedure

- Launch the wizard to configure one-way SSL/TLS for management interfaces by entering the following command in the management CLI.

  Syntax

  ```
  security enable-ssl-management --interactive
  ```

  Enter the required information when prompted.

  Use the **--lets-encrypt** option to obtain and use certificates from the Let's Encrypt certificate authority.

  If SSL/TLS is already enabled for management interfaces the wizard exits with the following message:

  ```
  SSL is already enabled for http-interface
  ```

  To change the existing configuration, first disable SSL/TLS for management interfaces and then create a new configuration. For information about disabling SSL/TLS for management interfaces, see Disabling SSL/TLS for management interfaces by using the wizard .

  NOTE

  To enable one-way SSL/TLS, enter **n** or blank when prompted to enable SSL mutual authentication. Setting mutual authentication enables two-way SSL/TLS.

  Example of using the wizard interactively

  ```
  security enable-ssl-management --interactive
  ```

  Example inputs to the wizard prompts

  ```
  Please provide required pieces of information to enable SSL:

  Certificate info:
  Key-store file name (default management.keystore): exampleKeystore.pkcs12
  Password (blank generated): secret
  What is your first and last name? [Unknown]: localhost
  What is the name of your organizational unit? [Unknown]:
  What is the name of your organization? [Unknown]:
  What is the name of your City or Locality? [Unknown]:
  ```

```
What is the name of your State or Province? [Unknown]:
What is the two-letter country code for this unit? [Unknown]:
Is CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
correct y/n [y]?y
Validity (in days, blank default): 365
Alias (blank generated): localhost
Enable SSL Mutual Authentication y/n (blank n):n //For one way SSL/TLS enter blank or n
here

SSL options:
keystore file: exampleKeystore.pkcs12
distinguished name: CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown,
C=Unknown
password: secret
validity: 365
alias: localhost
Server keystore file exampleKeystore.pkcs12, certificate file exampleKeystore.pem and
exampleKeystore.csr file will be generated in server configuration directory.

Do you confirm y/n :y
```

After you enter **y**, the server reloads. If you configured a self-signed certificate, used the wizard to generate self-signed certificate or configured a certificate that is not trusted by the Java virtual machine (JVM), the management CLI prompts you to accept the certificate that the server presents.

```
Unable to connect due to unrecognised server certificate
Subject    - CN=localhost,OU=Unknown,O=Unknown,L=Unknown,ST=Unknown,C=Unknown
Issuer     - CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown,
C=Unknown
Valid From - Mon Jan 30 23:32:20 IST 2023
Valid To   - Tue Jan 30 23:32:20 IST 2024
MD5 : b6:e7:f0:57:59:9e:bf:b8:20:99:10:fc:e2:0b:0f:d0
SHA1 : 9c:f0:92:de:c1:11:df:71:0b:d7:16:02:c8:7e:c9:83:ab:e3:0c:2e


Accept certificate? [N]o, [T]emporarily, [P]ermanently :
```

Enter **T** or **P** to proceed with the connection.

You get the following output:

```
Server reloaded.
SSL enabled for http-interface
ssl-context is ssl-context-a18ba30e-6a26-4ed6-87c5-feb7f3e4dff1
key-manager is key-manager-a18ba30e-6a26-4ed6-87c5-feb7f3e4dff1
key-store   is key-store-a18ba30e-6a26-4ed6-87c5-feb7f3e4dff1
```

Verification

- Verify SSL/TLS by connecting with the management CLI client.
  You can test SSL/TLS by placing an Elytron client SSL context in a configuration file and then connecting to the server using the management CLI and referencing the configuration file.

a. Navigate to the directory containing the keystore file. In this example, the keystore file **exampleKeystore.pkcs12** was generated in the server's **standalone/configuration** directory.

Example

```
$ cd JBOSS_HOME/standalone/configuration
```

b. Create a client **trust-store** with server certificates.

Syntax

```
$ keytool -importcert -keystore <trust_store_name> -storepass <password> -alias
<alias> -trustcacerts -file <file_containing_server_certificate>
```

Example

```
$ keytool -importcert -keystore client.truststore.pkcs12 -storepass secret -alias localhost
-trustcacerts -file exampleKeystore.pem
```

If you used a self-signed certificate, you are prompted to trust the certificate.

c. Define the client-side SSL context in a file, for example **example-security.xml**.

Syntax

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

<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <key-stores>
      <key-store name="${key-store_name}" type="PKCS12" >
        <file name="${path_to_truststore}"/>
        <key-store-clear-password password="${keystore_password}" />
      </key-store>
    </key-stores>
    <ssl-contexts>
      <ssl-context name="${ssl_context_name}">
        <trust-store key-store-name="${trust_store_name}" />
      </ssl-context>
    </ssl-contexts>
    <ssl-context-rules>
      <rule use-ssl-context="${ssl_context_name}" />
    </ssl-context-rules>
  </authentication-client>
</configuration>
```

Example

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

<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <key-stores>
```

```
            <key-store name="clientStore" type="PKCS12" >
                <file
name="JBOSS_HOME/standalone/configuration/client.truststore.pkcs12"/>
                <key-store-clear-password password="secret" />
            </key-store>
        </key-stores>
        <ssl-contexts>
            <ssl-context name="client-SSL-context">
                <trust-store key-store-name="clientStore" />
            </ssl-context>
        </ssl-contexts>
        <ssl-context-rules>
            <rule use-ssl-context="client-SSL-context" />
        </ssl-context-rules>
    </authentication-client>
</configuration>
```

d. Connect to the server and issue a command.

Example

```
$ EAP_HOME/bin/jboss-cli.sh -c --controller=remote+https://127.0.0.1:9993 -
Dwildfly.config.url=<path_to_the_configuration_file>/example-security.xml :whoami
```

Expected output

```
{
    "outcome" => "success",
    "result" => {"identity" => {"username" => "$local"}}
}
```

- Verify SSL/TLS by using a browser.

a. Navigate to https://localhost:9993.
If you used a self-signed certificate, the browser presents a warning that the certificate presented by the server is unknown.

b. Inspect the certificate and verify that the fingerprints shown in your browser match the fingerprints of the certificate in your keystore. You can view the certificate you generated with the following command:

Syntax

```
/subsystem=elytron/key-store=<server_keystore_name>:read-alias(alias=<alias>)
```

Example

```
/subsystem=elytron/key-store=key-store-a18ba30e-6a26-4ed6-87c5-feb7f3e4dff1:read-
alias(alias="localhost")
```

You can get the keystore name from the wizard’s output, for example, "key-store is key-store-a18ba30e-6a26-4ed6-87c5-feb7f3e4dff1".

Example output

```
...
"sha-1-digest" => "48:e3:6f:16:d1:af:4b:31:8f:9b:0b:7f:33:94:58:af:69:85:c
0:ea",
"sha-256-digest" => "8f:3e:6b:b5:56:e0:d1:97:81:bc:f1:8d:c8:66:75:06:db:7d
:4d:b6:b1:d3:34:dd:f5:6c:85:ca:c7:2b:5b:c7",
...
```

After you accept the server certificate, you are prompted for login credentials. You can login using user credentials of existing JBoss EAP users.

SSL/TLS is now enabled for JBoss EAP management interfaces.

Additional resources

- **key-manager** attributes
- **key-store** attributes
- **server-ssl-context** attributes

1.1.2. Enabling one-way SSL/TLS for management interfaces by using the subsystem commands

Use the **elytron** subsystem commands to secure the JBoss EAP management interfaces with SSL/TLS.

For testing and development purposes, you can use self-signed certificates. You can either use an existing keystore containing certificates or use the keystore that Elytron generates when you create the **key-store** resource. For production environments always use certificate authority (CA)-signed certificates.

IMPORTANT

Do not use self-signed certificates in a production environment. Use only the certificates signed by a certificate authority (CA).

Prerequisites

- JBoss EAP is running.

Procedure

1. Configure a keystore to store certificates.
   You can either provide a path to an existing keystore, for example, the one that contains CA-signed certificates, or provide a path to the keystore to create.

   ```
   /subsystem=elytron/key-store=<keystore_name>:add(path=<path_to_keystore>, credential-reference=<credential_reference>, type=<keystore_type>)
   ```

   Example

   ```
   /subsystem=elytron/key-store=exampleKeyStore:add(path=exampleserver.keystore.pkcs12,
   relative-to=jboss.server.config.dir,credential-reference={clear-text=secret},type=PKCS12)
   ```

2. If the keystore doesn't contain any certificates, or you used the step above to create the keystore, you must generate a certificate and store the certificate in a file.

   a. Generate a key pair in the keystore.

      Syntax

      ```
      /subsystem=elytron/key-store=<keystore_name>:generate-key-
      pair(alias=<keystore_alias>,algorithm=<algorithm>,key-
      size=<key_size>,validity=<validity_in_days>,credential-
      reference=<credential_reference>,distinguished-name="<distinguished_name>")
      ```

      Example

      ```
      /subsystem=elytron/key-store=exampleKeyStore:generate-key-
      pair(alias=localhost,algorithm=RSA,key-size=2048,validity=365,credential-reference=
      {clear-text=secret},distinguished-name="CN=localhost")
      ```

   b. Store the certificate in a file.

      Syntax

      ```
      /subsystem=elytron/key-store=<keystore_name>:store()
      ```

      Example

      ```
      /subsystem=elytron/key-store=exampleKeyStore:store()
      ```

3. Configure a **key-manager** referencing the **key-store**.

   Syntax

   ```
   /subsystem=elytron/key-manager=<key-manager_name>:add(key-store=<key-
   store_name>,credential-reference=<credential_reference>)
   ```

   Example

   ```
   /subsystem=elytron/key-manager=exampleKeyManager:add(key-
   store=exampleKeyStore,credential-reference={clear-text=secret})
   ```

IMPORTANT

Red Hat did not specify the algorithm attribute because the **elytron** subsystem uses **KeyManagerFactory.getDefaultAlgorithm()** to determine an algorithm by default. However, you can specify the algorithm attribute.

To specify the algorithm attribute, you need to know what key manager algorithms are provided by the Java Development Kit (JDK) you are using. For example, a JDK that uses Java Secure Socket Extension (SunJSSE) provides the PKIX and SunX509 algorithms.

In the command you could specify SunX509 as the **key-manager** algorithm attribute.

4. Configure a **server-ssl-context** referencing the **key-manager**.

Syntax

```
/subsystem=elytron/server-ssl-context=<server-ssl-context_name>:add(key-manager=<key-manager_name>, protocols=<list_of_protocols>)
```

Example

```
/subsystem=elytron/server-ssl-context=examplehttpsSSC:add(key-manager=exampleKeyManager, protocols=["TLSv1.2"])
```

IMPORTANT

You need to determine what SSL/TLS protocols you want to support. The example command uses TLSv1.2.

- For TLSv1.2 and earlier, use the **cipher-suite-filter** argument to specify which cipher suites are allowed.
- For TLSv1.3, use the **cipher-suite-names** argument to specify which cipher suites are allowed. TLSv1.3 is disabled by default. If you do not specify a protocol with the **protocols** attribute or the specified set contains TLSv1.3, configuring **cipher-suite-names** enables TLSv1.3.

Use the **use-cipher-suites-order** argument to honor server cipher suite order. The **use-cipher-suites-order** attribute is set to **true** by default. This differs from the legacy security subsystem behavior, which defaults to honoring client cipher suite order.

5. Update the management interfaces to use the configured **server-ssl-context**.

Syntax

```
/core-service=management/management-interface=http-interface:write-attribute(name=ssl-context, value=<server-ssl-context_name>)
/core-service=management/management-interface=http-interface:write-attribute(name=secure-socket-binding, value=management-https)
```

Example

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

6. Reload the server.

   ```
   reload
   ```

   If you used self-signed certificates for enabling SSL/TLS, the management CLI prompts you to accept the certificate that the server presents. This is the certificate you configured the keystore with.

Example output

```
Unable to connect due to unrecognised server certificate
Subject    - CN=localhost
Issuer     - CN=localhost
Valid From - Mon Jan 30 23:47:21 IST 2023
Valid To   - Tue Jan 30 23:47:21 IST 2024
MD5 : a1:00:84:78:a6:46:a4:78:4d:44:c8:6d:ba:1f:30:6a
SHA1 : a4:e5:c1:34:ad:e0:91:18:6f:f6:57:09:91:ae:17:8d:70:f0:1a:7d


Accept certificate? [N]o, [T]emporarily, [P]ermanently :
```

Enter **T** or **P** to proceed with the connection.

Verification

- Verify SSL/TLS by connecting through a client.
  You can test SSL/TLS by placing an Elytron client SSL context in a configuration file and then connecting to the server by using the management CLI referencing the configuration file.

  a. Navigate to the directory containing the keystore file. In this example, the keystore file **exampleserver.keystore.pkcs12** was generated in the server's **standalone/configuration** directory.

  #### Example

  ```
  $ cd JBOSS_HOME/standalone/configuration
  ```

  b. Export the server certificate so that it can be imported into a client trust store.

  ```
  $ keytool -export -alias <alias> -keystore <key_store> -storepass <keystore_password> -file <file_name>
  ```

  #### Example

  ```
  $ keytool -export -alias localhost -keystore exampleserver.keystore.pkcs12 -file -storepass secret server.cer
  ```

c. Create a client **trust-store** with the server certificates.

Syntax

```
$ keytool -importcert -keystore <trust_store_name> -storepass <password> -alias
<alias> -trustcacerts -file <file_containing_server_certificate>
```

Example

```
$ keytool -importcert -keystore client.truststore.pkcs12 -storepass secret -alias localhost
-trustcacerts -file server.cer
```

If you used a self-signed certificate, you are prompted to trust the certificate.

d. Define the client-side SSL context in a file, for example **example-security.xml**.

Syntax

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

<configuration>
   <authentication-client xmlns="urn:elytron:client:1.2">
      <key-stores>
         <key-store name="${key-store_name}" type="PKCS12" >
            <file name="${path_to_truststore}"/>
            <key-store-clear-password password="${keystore_password}" />
         </key-store>
      </key-stores>
      <ssl-contexts>
         <ssl-context name="${ssl_context_name}">
            <trust-store key-store-name="${trust_store_name}" />
         </ssl-context>
      </ssl-contexts>
      <ssl-context-rules>
         <rule use-ssl-context="${ssl_context_name}" />
      </ssl-context-rules>
   </authentication-client>
</configuration>
```

Example

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

<configuration>
   <authentication-client xmlns="urn:elytron:client:1.2">
      <key-stores>
         <key-store name="clientStore" type="PKCS12" >
            <file
name="JBOSS_HOME/standalone/configuration/client.truststore.pkcs12"/>
            <key-store-clear-password password="secret" />
         </key-store>
      </key-stores>
      <ssl-contexts>
         <ssl-context name="client-SSL-context">
```

```
                <trust-store key-store-name="clientStore" />
            </ssl-context>
        </ssl-contexts>
        <ssl-context-rules>
            <rule use-ssl-context="client-SSL-context" />
        </ssl-context-rules>
    </authentication-client>
</configuration>
```

e. Connect to the server and issue a command.

Example

```
$ EAP_HOME/bin/jboss-cli.sh -c --controller=remote+https://127.0.0.1:9993 -
Dwildfly.config.url=example-security.xml :whoami
```

Expected output

```
{
    "outcome" => "success",
    "result" => {"identity" => {"username" => "$local"}}
}
```

- Verify SSL/TLS by using a browser.

a. Navigate to https://localhost:9993.
If you used a self-signed certificate, the browser presents a warning that the certificate presented by the server is unknown.

b. Inspect the certificate and verify that the fingerprints shown in your browser match the fingerprints of the certificate in your keystore. You can view the certificate you generated with the following command:

Syntax

```
/subsystem=elytron/key-store=<server_keystore_name>:read-alias(alias=<alias>)
```

Example

```
/subsystem=elytron/key-store=exampleKeyStore:read-alias(alias="localhost")
```

Example output

```
...
"sha-1-digest" => "48:e3:6f:16:d1:af:4b:31:8f:9b:0b:7f:33:94:58:af:69:85:c
0:ea",
"sha-256-digest" => "8f:3e:6b:b5:56:e0:d1:97:81:bc:f1:8d:c8:66:75:06:db:7d
:4d:b6:b1:d3:34:dd:f5:6c:85:ca:c7:2b:5b:c7",
...
```

After you accept the server certificate, you are prompted for login credentials. You can login using user credentials of existing JBoss EAP users.

SSL/TLS is now enabled for JBoss EAP management interfaces.

Additional resources

- **key-manager** attributes
- **key-store** attributes
- **server-ssl-context** attributes

1.1.3. Disabling SSL/TLS for management interfaces by using the security command

Use the **security** command to disable SSL/TLS for management interfaces. You might want to do this to use a different SSL/TLS configuration to the one that is configured.

Disabling SSL/TLS using the command does not delete the Elytron resources. The command just undefines the **secure-socket-binding** and the **ssl-context** attributes of the **http-interface management-interface** resource.

Prerequisites

- JBoss EAP is running.

Procedure

- Use the **disable-ssl-management** command in the management CLI.

  ```
  security disable-ssl-management
  ```

  The server reloads with the following output:

  ```
  ...
  Server reloaded.
  Reconnected to server.
  SSL disabled for http-interface
  ```

You can enable SSL/TLS for server management interfaces using one of the following methods:

- Enable one-way SSL/TLS for management interfaces by using the wizard : Use this procedure to quickly set up SSL/TLS using a CLI-based wizard. Elytron creates the required resources for you based on your inputs to the wizard.
- Enable one-way SSL/TLS for management interfaces by using the subsystem commands : Use this procedure to configure the required resource for enabling SSL/TLS manually. Manually configuring the resources gives you more control over the server configuration.

1.2. ENABLING ONE-WAY SSL/TLS FOR APPLICATIONS DEPLOYED ON JBOSS EAP

Enable one-way SSL/TLS for applications deployed on JBoss EAP so that the communication between the applications and clients, such as web browsers, is secure.

To enable one-way SSL/TLS for applications deployed on JBoss EAP, you can use the following procedures:

- Enabling SSL/TLS for applications by using the automatically generated self-signed certificate : Use this procedure in development or testing environments only. This procedure helps you to quickly enable SSL/TLS for applications without having to do any configurations.
- Enable one-way SSL/TLS for applications deployed on JBoss EAP by using the wizard : Use this procedure to quickly set up SSL/TLS using a CLI-based wizard. Elytron creates the required resources for you based on your inputs to the wizard.
- Enabling one-way SSL/TLS for applications by using the subsystem commands : Use this method to configure the required resource for enabling SSL/TLS manually. Manually configuring the resources gives you more control over the server configuration.

Additionally, you can disable SSL/TLS for applications deployed on JBoss EAP by using the procedure Disabling SSL/TLS for applications by using the security command .

1.2.1. The default SSL context in Elytron

To help developers quickly set up one-way SSL/TLS for applications, the **elytron** subsystem contains the required resources to enable one-way SSL/TLS, ready to use in a development or testing environment by default.

The following resources are provided by default:

- A **key-store** named **applicationKS**.
- A **key-manager**, named **applicationKM**, referencing the **key-store**.
- A **server-ssl-context**, named **applicationSSC**, referencing the **key-manager**.

Default TLS configuration

```
...
<tls>
    <key-stores>
        <key-store name="applicationKS">
            <credential-reference clear-text="password"/>
            <implementation type="JKS"/>
            <file path="application.keystore" relative-to="jboss.server.config.dir"/>
        </key-store>
    </key-stores>
    <key-managers>
        <key-manager name="applicationKM" key-store="applicationKS" generate-self-signed-
certificate-host="localhost">
            <credential-reference clear-text="password"/>
        </key-manager>
    </key-managers>
    <server-ssl-contexts>
        <server-ssl-context name="applicationSSC" key-manager="applicationKM"/>
    </server-ssl-contexts>
</tls>
...
```

The default **key-manager**, **applicationKM**, contains a **generate-self-signed-certificate-host** attribute with the value **localhost**. The **generate-self-signed-certificate-host** attribute indicates that when this **key-manager** is used to obtain the server's certificate, if the file that backs its **key-store** doesn't already exist, then the key-manager should automatically generate a self-signed certificate with **localhost** as the **Common Name**. This generated self-signed certificate is stored in the file that backs the **key-store**.

As the file that backs the default key-store doesn't exist when the server is installed, just sending an https request to the server generates a self-signed certificate and enables one-way SSL/TLS for application. For more information, see Enabling SSL/TLS for applications by using the automatically generated self-signed certificate.

Additional resources

- **key-manager** attributes
- **key-store** attributes
- **server-ssl-context** attributes

1.2.2. Enabling SSL/TLS for applications by using the automatically generated self-signed certificate

JBoss EAP automatically generates a self-signed certificate the first time the server receives an HTTPS request. The **elytron** subsystem also contains **key-store**, **key-manager**, and **server-ssl-context** resources that are ready to use in a development or testing environment by default. Therefore, as soon as JBoss EAP generates a self-signed certificate, the applications are secured using the certificate.

IMPORTANT

Do not use self-signed certificates in a production environment. Use only the certificates signed by a certificate authority (CA).

Prerequisites

- JBoss EAP is running.

Procedure

- Navigate to the server URL at the port **8443**, for example, https://localhost:8443.
  JBoss EAP generates a self-signed certificate when it receives this request. You can see the server logs for details about this certificate.

  The browser flags the connection as insecure because the generated certificate is self-signed.

Verification

1. Compare the certificate JBoss EAP presented to the browser with the certificate in the server log.

   Example server log

   ```
   17:50:24,086 WARN  [org.wildfly.extension.elytron] (default task-1) WFLYELY01085:
   Generated self-signed certificate at /home/user1/Downloads/wildflies/wildfly-
   27.0.1.Final/standalone/configuration/application.keystore. Please note that self-signed
   certificates are not secure and should only be used for testing purposes. Do not use this self-
   signed certificate in production.
   SHA-1 fingerprint of the generated key is
   ```

```
11:2f:e7:8c:18:b7:2c:c1:b0:5a:ad:ea:83:e0:32:59:ba:73:91:e2
SHA-256 fingerprint of the generated key is
b2:a4:ed:b0:5c:c2:a1:4c:ca:39:03:e8:3a:11:e4:c5:c4:81:9d:46:97:7c:e6:6f:0c:45:f6:5d:64:3f:0d:
64
```

Example certificate presented to the browser

```
SHA-256 Fingerprint B2 A4 ED B0 5C C2 A1 4C CA 39 03 E8 3A 11 E4 C5
C4 81 9D 46 97 7C E6 6F 0C 45 F6 5D 64 3F 0D 64
SHA-1 Fingerprint 11 2F E7 8C 18 B7 2C C1 B0 5A AD EA 83 E0 32 59
BA 73 91 E2
```

2. If the fingerprints match, like in the example, you can proceed to the page.

SSL/TLS is enabled for applications.

Additional resources

- **key-manager** attributes
- **key-store** attributes
- **server-ssl-context** attributes

1.2.3. Enabling one-way SSL/TLS for applications deployed on JBoss EAP by using the wizard

Elytron provides a wizard to quickly set up SSL/TLS. You can either use an existing keystore containing certificates or use the keystore and self-signed certificates that the wizard generates to enable SSL/TLS. You can also obtain and use certificates from the Let's Encrypt certificate authority by using the **--lets-encrypt** option. For information about Let's Encrypt, see the Let's Encrypt documentation.

Use the self-signed certificates the wizard generates to enable SSL/TLS for testing and development purposes only. For production environments always use certificate authority (CA)-signed certificates.

IMPORTANT

Do not use self-signed certificates in a production environment. Use only the certificates signed by a certificate authority (CA).

The wizard configures the following resources that are required to enable SSL/TLS for applications:

- **key-store**
- **key-manager**
- **server-ssl-context**
- The **server-ssl-context** is then applied to Undertow **https-listener**.

Elytron names each resource as *resource-type-UUID*. For example, key-store-9e35a3be-62bb-4fff-afc2-2d8d141b82bc. The universally unique identifier (UUID) helps avoid name collisions for the resources.

Prerequisites

- JBoss EAP is running.

Procedure

- Launch the wizard to configure one-way SSL/TLS for applications by entering the following command in the management CLI:

  Syntax

  ```
  security enable-ssl-http-server --interactive
  ```

  Enter the required information when prompted.

  Use the **--lets-encrypt** option to obtain and use certificates from the Let’s Encrypt certificate authority.

  If a **server-ssl-context** already exists, the wizard exits with the following message:

  ```
  An SSL server context already exists on the HTTPS listener, use --override-ssl-context option
  to overwrite the existing SSL context
  ```

  NOTE

  The **elytron** subsystem contains an already configured **server-ssl-context** resource by default. Therefore, you must use the **--override-ssl-context** option the first time you launch the wizard after a fresh installation.

  For more information, see The default SSL context in Elytron .

  If you override the existing **server-ssl-context**, Elytron will use the **server-ssl-context** created by the wizard to enable SSL.

  NOTE

  To enable one-way SSL/TLS, enter **n** or blank when prompted to enable SSL mutual authentication. Setting mutual authentication enables two-way SSL/TLS.

  Example of starting the wizard

  ```
  security enable-ssl-http-server --interactive --override-ssl-context
  ```

  Example inputs to the wizard prompts

  ```
  Please provide required pieces of information to enable SSL:

  Certificate info:
  Key-store file name (default default-server.keystore): exampleKeystore.pkcs12
  Password (blank generated): secret
  What is your first and last name? [Unknown]: localhost
  What is the name of your organizational unit? [Unknown]:
  What is the name of your organization? [Unknown]:
  ```

```
What is the name of your City or Locality? [Unknown]:
What is the name of your State or Province? [Unknown]:
What is the two-letter country code for this unit? [Unknown]:
Is CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
correct y/n [y]?y
Validity (in days, blank default): 365
Alias (blank generated): localhost
Enable SSL Mutual Authentication y/n (blank n):n //For one way SSL/TLS enter blank or n
here

SSL options:
keystore file: exampleKeystore.pkcs12
distinguished name: CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown,
C=Unknown
password: secret
validity: 365
alias: localhost
Server keystore file exampleKeystore.pkcs12, certificate file exampleKeystore.pem and
exampleKeystore.csr file will be generated in server configuration directory.

Do you confirm y/n :y
```

After you enter **y**, the server reloads with the following output:

```
Server reloaded.
SSL enabled for default-server
ssl-context is ssl-context-4cba6678-c464-4dcc-90ff-9295312ac395
key-manager is key-manager-4cba6678-c464-4dcc-90ff-9295312ac395
key-store   is key-store-4cba6678-c464-4dcc-90ff-9295312ac395
```

Verification

1. Navigate to https://localhost:8443.
   If you used a self-signed certificate, the browser presents a warning that the certificate presented by the server is unknown.

2. Inspect the certificate and verify that the fingerprints shown in your browser match the fingerprints of the certificate in your keystore. You can view the certificate you generated with the following command:

   Syntax

   ```
   /subsystem=elytron/key-store=<server_keystore_name>:read-alias(alias=<alias>)
   ```

   Example

   ```
   /subsystem=elytron/key-store=key-store-4cba6678-c464-4dcc-90ff-9295312ac395:read-
   alias(alias="localhost")
   ```

   You can get the keystore name from the wizard’s output, for example, "key-store is key-store-4cba6678-c464-4dcc-90ff-9295312ac395".

   Example output

```
...
"sha-1-digest" => "48:e3:6f:16:d1:af:4b:31:8f:9b:0b:7f:33:94:58:af:69:85:c
0:ea",
"sha-256-digest" => "8f:3e:6b:b5:56:e0:d1:97:81:bc:f1:8d:c8:66:75:06:db:7d
:4d:b6:b1:d3:34:dd:f5:6c:85:ca:c7:2b:5b:c7",
...
```

SSL/TLS is now enabled for applications deployed on JBoss EAP.

Additional resources

- **key-manager** attributes
- **key-store** attributes
- **server-ssl-context** attributes

1.2.4. Enabling one-way SSL/TLS for applications by using the subsystem commands

Use the **elytron** subsystem commands to secure the applications deployed on JBoss EAP with SSL/TLS.

For testing and development purposes, you can use self-signed certificates. You can either use an existing keystore containing certificates or use the keystore that Elytron generates when you create the **key-store** resource. For production environments always use certificate authority (CA)-signed certificates.

IMPORTANT

Do not use self-signed certificates in a production environment. Use only the certificates signed by a certificate authority (CA).

Prerequisites

- JBoss EAP is running.

Procedure

1. Configure a keystore to store certificates.
   You can either provide a path to an existing keystore, for example, the one that contains CA-signed certificates, or provide a path to the keystore to create.

   ```
   /subsystem=elytron/key-store=<keystore_name>:add(path=<path_to_keystore>, credential-
   reference=<credential_reference>, type=<keystore_type>)
   ```

   Example

   ```
   /subsystem=elytron/key-store=exampleKeyStore:add(path=exampleserver.keystore.pkcs12,
   relative-to=jboss.server.config.dir,credential-reference={clear-text=secret},type=PKCS12)
   ```

2. If the keystore doesn’t contain any certificates, or you used the step above to create the keystore, you must generate a certificate and store the certificate in a file.

a. Generate a key pair in the keystore.

Syntax

```
/subsystem=elytron/key-store=<keystore_name>:generate-key-
pair(alias=<keystore_alias>,algorithm=<algorithm>,key-
size=<key_size>,validity=<validity_in_days>,credential-
reference=<credential_reference>,distinguished-name="<distinguished_name>")
```

Example

```
/subsystem=elytron/key-store=exampleKeyStore:generate-key-
pair(alias=localhost,algorithm=RSA,key-size=2048,validity=365,credential-reference=
{clear-text=secret},distinguished-name="CN=localhost")
```

b. Store the certificate in a file.

Syntax

```
/subsystem=elytron/key-store=<keystore_name>:store()
```

Example

```
/subsystem=elytron/key-store=exampleKeyStore:store()
```

3. Configure a **key-manager** referencing the **key-store**.

Syntax

```
/subsystem=elytron/key-manager=<key-manager_name>:add(key-store=<key-
store_name>,credential-reference=<credential_reference>)
```

Example

```
/subsystem=elytron/key-manager=exampleKeyManager:add(key-
store=exampleKeyStore,credential-reference={clear-text=secret})
```

IMPORTANT

Red Hat did not specify the algorithm attribute because the **elytron** subsystem uses **KeyManagerFactory.getDefaultAlgorithm()** to determine an algorithm by default. However, you can specify the algorithm attribute.

To specify the algorithm attribute, you need to know what key manager algorithms are provided by the Java Development Kit (JDK) you are using. For example, a JDK that uses Java Secure Socket Extension (SunJSSE) provides the PKIX and SunX509 algorithms.

In the command you could specify SunX509 as the **key-manager** algorithm attribute.

4. Configure a **server-ssl-context** referencing the **key-manager**.

Syntax

```
/subsystem=elytron/server-ssl-context=<server-ssl-context_name>:add(key-manager=<key-manager_name>, protocols=<list_of_protocols>)
```

Example

```
/subsystem=elytron/server-ssl-context=examplehttpsSSC:add(key-manager=exampleKeyManager, protocols=["TLSv1.2"])
```

IMPORTANT

You need to determine what SSL/TLS protocols you want to support. The example command uses TLSv1.2.

- For TLSv1.2 and earlier, use the **cipher-suite-filter** argument to specify which cipher suites are allowed.
- For TLSv1.3, use the **cipher-suite-names** argument to specify which cipher suites are allowed. TLSv1.3 is disabled by default. If you do not specify a protocol with the **protocols** attribute or the specified set contains TLSv1.3, configuring **cipher-suite-names** enables TLSv1.3.

Use the **use-cipher-suites-order** argument to honor server cipher suite order. The **use-cipher-suites-order** attribute is set to **true** by default. This differs from the legacy security subsystem behavior, which defaults to honoring client cipher suite order.

5. Update Undertow to use the configured **server-ssl-context**.

Syntax

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-context, value=<server-ssl-context_name>)
```

Example

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-context, value=examplehttpsSSC)
```

6. Reload the server.

```
reload
```

Verification

1. Navigate to https://localhost:8443.
   If you used a self-signed certificate, the browser presents a warning that the certificate presented by the server is unknown.

2. Inspect the certificate and verify that the fingerprints shown in your browser match the fingerprints of the certificate in your keystore. You can view the certificate you generated with the following command:

   Syntax

   ```
   /subsystem=elytron/key-store=<server_keystore_name>:read-alias(alias=<alias>)
   ```

   Example

   ```
   /subsystem=elytron/key-store=exampleKeyStore:read-alias(alias=localhost)
   ```

   Example output

   ```
   ...
   "sha-1-digest" => "cc:f1:82:59:c7:0d:f6:91:bc:3e:69:0a:38:fb:48:be:ec:7f:d
   4:bd",
   "sha-256-digest" => "c0:f3:f9:8b:3c:f1:72:17:64:54:35:a6:bb:82:7e:51:b0:78
   :30:cb:68:ef:04:0e:f5:2b:9d:62:ca:a7:f6:35",
   ...
   ```

SSL/TLS is now enabled for applications deployed on JBoss EAP.

Additional resources

- **key-manager** attributes
- **key-store** attributes
- **server-ssl-context** attributes

1.2.5. Disabling SSL/TLS for applications by using the security command

Use the **security** command to disable SSL/TLS for applications deployed on JBoss EAP. Disabling SSL/TLS using the command does not delete the Elytron resources. The command just sets the **ssl-context** for the server to its default value **applicationSSC**.

Prerequisites

- JBoss EAP is running.

Procedure

- Use the **security disable-ssl-http-server** command in the management CLI.

  ```
  security disable-ssl-http-server
  ```

  The server reloads with the following output:

  ```
  ...
  Server reloaded.
  SSL disabled for default-server
  ```

You can enable SSL/TLS for applications deployed on JBoss EAP using one of the following procedure:

- Enabling SSL/TLS for applications by using the automatically generated self-signed certificate : Use this procedure in development or testing environments only. This procedure helps you to quickly enable SSL/TLS for applications without having to do any configurations.
- Enable one-way SSL/TLS for applications deployed on JBoss EAP by using the wizard : Use this procedure to quickly set up SSL/TLS using a CLI-based wizard. Elytron creates the required resources for you based on your inputs to the wizard.
- Enabling one-way SSL/TLS for applications by using the subsystem commands : Use this method to configure the required resource for enabling SSL/TLS manually. Manually configuring the resources gives you more control over the server configuration.

Additional resources

- **key-manager** attributes
- **key-store** attributes
- **server-ssl-context** attributes

CHAPTER 2. ENABLING TWO-WAY SSL/TLS FOR MANAGEMENT INTERFACES AND APPLICATIONS

SSL/TLS, or transport layer security (TLS), is a certificates-based security protocol that is used to secure the data transfer between two entities communicating over a network. Use two-way SSL/TLS when you want the server to connect only with trusted clients.

Two-way SSL/TLS provides the following security functions:

Authentication

In one-way SSL/TLS, the server presents its certificate to a client to authenticate itself. In two-way SSL/TLS, the client also presents its certificate to the server for the server to authenticate the client. Two-way SSL/TLS, therefore, is also called mutual authentication.

Confidentiality

The data transferred between the client and the server is encrypted.

Data integrity

The TLS protocol provides data integrity with secure hash functions, which are used for message authentication code (MAC) computations. You can enforce specific algorithms and hash functions for the connections using the **cipher-suite-filter** and **cipher-suite-names** attributes of the SSL context resources.

For more information, see **server-ssl-context** attributes.

You can secure both JBoss EAP management interfaces and deployed applications by using two-way SSL/TLS.

To secure management interfaces with two-way SSL/TLS, use the following procedures:

- Obtain a certificate from a certificate authority (CA) for the client. Alternatively, for non-production environments, you can generate a self-signed certificate by following the procedure: Generate client certificates.
- Configure a trust store and a trust manager for client certificate
- Configure a server certificate for two-way SSL/TLS.
- Configure SSL context to secure JBoss EAP management interfaces with SSL/TLS

To secure applications deployed on JBoss EAP with two-way SSL/TLS, use the following procedures:

- Obtain a certificate from a certificate authority (CA) for the client. Alternatively, for non-production environments, you can generate a self-signed certificate by following the procedure: Generate client certificates.
- Configure a trust store and a trust manager for client certificate
- Configure a server certificate for two-way SSL/TLS
- Configure SSL context to secure applications deployed on JBoss EAP with SSL/TLS

You can configure certificate revocation checks by following the procedures in Configuring certificate revocation checks in Elytron.

2.1. GENERATING CLIENT CERTIFICATES

Generate self-signed client certificates using the **keytool** command, in the CLI, for the purpose of testing and development of a two-way SSL/TLS configuration.

IMPORTANT

Do not use self-signed certificates in a production environment. Use only the certificates signed by a certificate authority (CA).

Procedure

1. Generate a client certificate.

   Syntax

   ```
   $ keytool -genkeypair -alias <keystore_alias> -keyalg <algorithm> -keysize <key_size> -validity <validity_in_days> -keystore <keystore_name> -dname "<distinguished_name>" -keypass <private_key_password> -storepass <keystore_password>
   ```

   Example

   ```
   $ keytool -genkeypair -alias exampleClientKeyStore -keyalg RSA -keysize 2048 -validity 365 -keystore exampleclient.keystore.pkcs12 -dname "CN=client" -keypass secret -storepass secret
   ```

2. Export the client certificate to a file.

   Syntax

   ```
   $ keytool -exportcert  -keystore <keystore_name> -alias <keystore_alias> -keypass <private_key_password> -storepass <keystore_password> -file <file_path>
   ```

   Example

   ```
   $ keytool -exportcert  -keystore exampleclient.keystore.pkcs12 -alias exampleClientKeyStore -keypass secret -storepass secret -file EAP_HOME/standalone/configuration/client.cer

   Certificate stored in file <EAP_HOME/standalone/configuration/client.cer>
   ```

You can now use the generated client certificate to configure a server trust store and a trust manager in a server. For more information, see Configuring a trust store and a trust manager for client certificates .

2.2. CONFIGURING A TRUST STORE AND A TRUST MANAGER FOR CLIENT CERTIFICATES

Configure a trust store with the client certificate and a trust manager with a reference to the trust store to verify the client certificate during the TLS handshake.

Prerequisites

- You have obtained or generated a client certificate.
  For more information, see Generating client certificates.

- JBoss EAP is running.

Procedure

1. Configure a trust store with a client certificate by using the management CLI.

    a. Create a server trust store to store the client certificate to trust.

    Syntax

    ```
    /subsystem=elytron/key-
    store=<server_trust_store_name>:add(path=<path_to_server_trust_store_file>,credential
    -reference={<password>})
    ```

    Example

    ```
    /subsystem=elytron/key-
    store=exampleServerTrustStore:add(path=exampleTLSServer.truststore,relative-
    to=jboss.server.config.dir,credential-reference={clear-text=secret})
    {"outcome" => "success"}
    ```

    b. Import the client certificate to the server trust store by specifying the client certificate alias. Only the clients that present a certificate that the server's trust store trusts can connect to the server.

    NOTE

    If you are configuring two-way SSL/TLS by using a self-signed certificate, set **validate** to **false** because no chain of trust exists for the certificate.

    If you are configuring two-way SSL/TLS in a production environment by using certificates signed by a CA, set **validate** to **true**.

    Syntax

    ```
    /subsystem=elytron/key-store=<server_trust_store_name>:import-
    certificate(alias=<alias>,path=<certificate_file>,credential-reference={<password>},trust-
    cacerts=<true_or_false>,validate=<true_false>)
    ```

    Example

    ```
    /subsystem=elytron/key-store=exampleServerTrustStore:import-
    certificate(alias=client,path=client.cer,relative-to=jboss.server.config.dir,credential-
    reference={clear-text=serverTrustSecret},trust-cacerts=true,validate=false)
    {"outcome" => "success"}
    ```

    c. Export the client certificate to a trust store file.

    Syntax

    ```
    /subsystem=elytron/key-store=<server_trust_store_name>:store()
    ```

Example

```
/subsystem=elytron/key-store=exampleServerTrustStore:store()
{
    "outcome" => "success",
    "result" => undefined
}
```

2. Configure a trust manager to verify the client certificate during the TLS handshake.

   Syntax

   ```
   /subsystem=elytron/trust-manager=<trust_manager_name>:add(key-
   store=<server_trust_store_name>)
   ```

   Example

   ```
   /subsystem=elytron/trust-manager=exampleTLSTrustManager:add(key-
   store=exampleServerTrustStore)
   {"outcome" => "success"}
   ```

The client certificate in the configured trust store is used to verify the certificate that a client presents during TLS handshake with the server.

Additional resources

- Configuring certificate revocation checks using certificate revocation lists
- Configuring certificate revocation checks using OCSP in Elytron
- **key-store** attributes
- **trust-manager** attributes

2.3. CONFIGURING A SERVER CERTIFICATE FOR TWO-WAY SSL/TLS

Configure a server certificate, which will be presented to clients during the TLS handshake.

Prerequisites

- JBoss EAP is running.

Procedure

1. Generate a self-signed server certificate to use for testing and development purposes. If you have obtained a certificate from a certificate authority (CA), skip this step.

   IMPORTANT

   Do not use self-signed certificates in a production environment. Use only the certificates signed by a certificate authority (CA).

a. Create a key store to store server certificate.

Syntax

```
/subsystem=elytron/key-store=<key_store_name>:add(path=<path>,credential-
reference={<password>},type=<key_store_type>)
```

Example

```
/subsystem=elytron/key-
store=exampleServerKeyStore:add(path=server.keystore.pkcs12,relative-
to=jboss.server.config.dir,credential-reference={clear-text=secret},type=PKCS12)
{"outcome" => "success"}
```

b. Generate a server certificate in the key store.

Syntax

```
/subsystem=elytron/key-store=<key_store_name>:generate-key-
pair(alias=<alias>,algorithm=<algorithm>,key-
size=<key_size>,validity=<validaity_in_days>,credential-reference=
{<password>},distinguished-name="<distinguished_name_in_certificate>")
```

Example

```
/subsystem=elytron/key-store=exampleServerKeyStore:generate-key-
pair(alias=localhost,algorithm=RSA,key-size=2048,validity=365,credential-reference=
{clear-text=secret},distinguished-name="CN=localhost")
{"outcome" => "success"}
```

c. Store the key store to a file.

Syntax

```
/subsystem=elytron/key-store=<key_store_name>:store()
```

Example

```
/subsystem=elytron/key-store=exampleServerKeyStore:store()
{
    "outcome" => "success",
    "result" => undefined
}
```

d. Export the server certificate.

Syntax

```
/subsystem=elytron/key-store=<key_store_name>:export-
certificate(alias=<alias>,path=<path_to_certificate>,pem=true)
```

Example

```
/subsystem=elytron/key-store=exampleServerKeyStore:export-
certificate(alias=localhost,path=server.cer,pem=true,relative-to=jboss.server.config.dir)
{"outcome" => "success"}
```

2. Create a key manager referencing the server key store.

   Syntax

   ```
   /subsystem=elytron/key-manager=<key_manager_name>:add(credential-reference=
   {<password>},key-store=<key_store_name>)
   ```

   Example

   ```
   /subsystem=elytron/key-manager=exampleServerKeyManager:add(credential-reference=
   {clear-text=secret},key-store=exampleServerKeyStore)
   {"outcome" => "success"}
   ```

   The server presents this certificate to the client when SSL/TLS is enabled.

3. Import the server certificate to the client's trust store so that the client can verify the server certificate during SSL handshake.

   Syntax

   ```
   $ keytool -import -file <server_certificate_file> -alias <alias> -keystore
   <client_trust_store_file> -storepass <password>
   ```

   Example

   ```
   $ keytool -import -file EAP_HOME/standalone/configuration/server.cer -alias server -
   keystore client.truststore.p12 -storepass secret

   Owner: CN=localhost
   Issuer: CN=localhost
   Serial number: 52679016fbb54f46
   Valid from: Fri Sep 30 18:25:29 IST 2022 until: Sat Sep 30 18:25:29 IST 2023
   Certificate fingerprints:
     SHA1: 4B:68:24:9E:2A:2D:01:4E:23:69:94:C8:9A:1C:8F:A5:D4:27:CB:98
     SHA256:
   C0:AF:74:12:90:66:25:B2:65:4E:6B:4B:89:81:2D:6B:D5:2A:F4:04:BC:85:DA:1C:AB:26:6D:57:9
   F:9F:EE:15
   Signature algorithm name: SHA256withRSA
   Subject Public Key Algorithm: 1024-bit RSA key (disabled)
   Version: 3

   Extensions:

   #1: ObjectId: 2.5.29.14 Criticality=false
   SubjectKeyIdentifier [
   KeyIdentifier [
   0000: 59 13 DC 6A 81 B9 27 18   6E 72 17 0E 67 FC 9F 8F  Y..j..'.nr..g...
   0010: 04 01 74 8F                                        ..t.
   ]
   ```

```
]


Warning:
The input uses a 1024-bit RSA key which is considered a security risk and is disabled.

Trust this certificate? [no]:
```

Enter **yes**. You get the following output:

```
Certificate was added to keystore
```

Next steps

- To secure the management interfaces with two-way SSL/TLS, follow this procedure:
- Configuring SSL context to secure JBoss EAP interfaces with SSL/TLS
- To secure applications deployed to JBoss EAP with SSL/TLS, follow this procedure: Configuring SSL context to secure applications deployed on JBoss EAP with SSL/TLS

Additional resources

- **key-store** attributes
- **key-manager** attributes

2.4. CONFIGURING SSL CONTEXT TO SECURE JBOSS EAP MANAGEMENT INTERFACES WITH SSL/TLS

Secure the JBoss EAP management interfaces with two-way SSL/TLS so that only the clients that present a certificate trusted by the server can connect to the server's management interfaces.

Prerequisites

- JBoss EAP is running.
- You have configured server trust store and a trust manager for client certificates. For more information, see Configuring a trust store and a trust manager for client certificates .
- You have configured the server certificate. For more information, see Configuring the server certificate for SSL/TLS

Procedure

1. Configure a server SSL context to enable two-way SSL.

   Syntax

   ```
   /subsystem=elytron/server-ssl-context=<server_ssl_context_name>:add(key-manager=<key_manager_name>,trust-manager=<trust_manager_name>,need-client-auth=true)
   ```

Example

```
/subsystem=elytron/server-ssl-context=exampleServerSSLContext:add(key-
manager=exampleServerKeyManager,trust-manager=exampleTLSTrustManager,need-
client-auth=true)
{"outcome" => "success"}
```

By default, the SSL context uses TLSv1.2. You can configure the **protocols** attribute to use TLSv1.3 as follows:

Syntax

```
/subsystem=elytron/server-ssl-context=<server-ssl-context-name>:add(key-
manager=<key_manager_name>,trust-manager=<trust_manager_name>,need-client-
auth=true,protocols=[TLSv1.3])
```

2. Add a reference to the SSLContext to use for the http management interface.

Syntax

```
/core-service=management/management-interface=http-interface:write-attribute(name=ssl-
context, value=<server_ssl_context_name>)
```

Example

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

3. Define the socket binding configuration to use for the HTTPS management interface's socket.

Syntax

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

Example

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

–

4. Reload the server.

```
reload

...
Accept certificate? [N]o, [T]emporarily, [P]ermanently :
```

Enter **T** or **P** to accept the certificate provided by the server either temporarily or permanently.

The management CLI disconnects because it expects a client certificate to be presented.

Verification

- Verify that management console is protected.
  a. Verify using the CLI:

     Syntax

     ```
     $ curl --verbose --location --cacert <server_certificate> --cert
     <client_keystore>:<password> --cert-type P12 https://localhost:9993
     ```

     Example

     ```
     $ curl --verbose --location --cacert server.cer --cert
     EAP_HOME/standalone/configuration/exampleclient.keystore.pkcs12:secret --cert-type
     P12 https://localhost:9993
     ...
     < HTTP/1.1 200 OK
     ...
     ```

  b. Verify using a browser.
     i. Import the client certificate into your browser. The example certificate created in the Generating client certificates procedure is called **exampleclient.keystore.pkcs12** and the example password to import it is **secret**.
        Refer to your browser's documentation for information about importing certificates to the browser.
     ii. Access **https://localhost:9993** in a browser.
        The browser prompts you to present a certificate to identify with the server.
     iii. Choose the certificate you imported to the browser. For example, **exampleclient.keystore.pkcs12**.
        If you use a self-signed certificate, the browser presents a warning that the certificate presented by the server is unknown.
     iv. Inspect the certificate and verify that the fingerprints shown in your browser match the fingerprints of the certificate in your keystore. You can view the certificate in a keystore with the following command:

        Syntax

        ```
        /subsystem=elytron/key-store=<server_keystore_name>:read-alias(alias=<alias>)
        ```

-

Example

```
/subsystem=elytron/key-store=exampleServerKeyStore:read-alias(alias=localhost)
...
"sha-1-digest" => "5e:3e:ad:c8:df:d7:f6:63:38:05:e2:a3:a7:31:07:82:c8:c8:94:47",
"sha-256-digest" =>
"11:b6:8f:00:42:e1:7f:6c:16:ef:db:08:5e:13:d9:b8:16:6e:a0:3c:2e:d4:e5:fd:cb:53:90:88:
d2:9c:b1:99",
```

After you accept the server certificate, you are prompted for login credentials. You can login using user credentials of existing JBoss EAP users.

- Verify that management CLI is protected.
    - Create the file **wildfly-config.xml** with the following content:

Example

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

<configuration>
    <authentication-client xmlns="urn:elytron:client:1.7">
        <key-stores>
            <key-store name="truststore" type="PKCS12">
                <file name="${path_to_client_truststore}/client.truststore.p12"/>
                <key-store-clear-password password="secret" />
            </key-store>
            <key-store name="keystore" type="PKCS12">
                <file name="${path_to_client_truststore}/exampleclient.keystore.pkcs12"/>
                <key-store-clear-password password="secret" />
            </key-store>
        </key-stores>
        <ssl-contexts>
            <ssl-context name="client-context">
                <trust-store key-store-name="truststore"/>
                <key-store-ssl-certificate key-store-name="keystore">
                    <key-store-clear-password password="secret" />
                </key-store-ssl-certificate>
                <providers>
                    <global/>
                </providers>
            </ssl-context>
        </ssl-contexts>
        <ssl-context-rules>
            <rule use-ssl-context="client-context" />
        </ssl-context-rules>
    </authentication-client>
</configuration>
```

NOTE

You can use masked passwords in the **key-store-clear-password** element, in place of clear text, for obfuscation.

- Access management CLI by presenting the client certificate.

```
$ ./jboss-cli.sh --controller=remote+https://127.0.0.1:9993 -
Dwildfly.config.url=/path/to/wildfly-config.xml --connect
```

Both the clients: the client's web browser, and management CLI, trust the server's certificate, and the server trusts both clients. The communication between the client and server is over SSL/TLS.

Additional resources

- **server-ssl-context** attributes

2.5. CONFIGURING SERVER-SSL-CONTEXT TO SECURE APPLICATIONS DEPLOYED ON JBOSS EAP WITH SSL/TLS

Elytron provides a default **server-ssl-context** called **applicationSSC**, which you can use to configure SSL/TLS. Alternately, you can create your own SSL context in Elytron. The following procedure demonstrates using the default SSL context - **applicationSSC**, to configure SSL/TLS for applications.

Prerequisites

- JBoss EAP is running.
- You have configured a server trust store and a trust manager for client certificates.
  For more information, see Configuring a trust store and a trust manager for client certificates .
- You have configured the server certificate.
  For more information, see Configuring the server certificate for SSL/TLS

Procedure

1. Configure the default server SSL context to enable two-way SSL.

   ```
   /subsystem=elytron/server-ssl-context=applicationSSC:write-attribute(name=need-client-
   auth,value=true)
   {
       "outcome" => "success",
       "response-headers" => {
           "operation-requires-reload" => true,
           "process-state" => "reload-required"
       }
   }
   ```

   By default, the SSL context uses TLSv1.2. You can configure the **protocols** attribute to use TLSv1.3 as follows:

   ```
   /subsystem=elytron/server-ssl-context=applicationSSC:write-
   attribute(name=protocols,value=[TLSv1.3])
   ```

2. Configure a trust manager for the server SSL context.

   Syntax

```
/subsystem=elytron/server-ssl-context=applicationSSC:write-attribute(name=trust-
manager,value=<server_trust_manager>)
```

Example

```
/subsystem=elytron/server-ssl-context=applicationSSC:write-attribute(name=trust-
manager,value=exampleTLSTrustManager)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

3. Configure a key manager of the server SSL context.

Syntax

```
/subsystem=elytron/server-ssl-context=applicationSSC:write-attribute(name=key-
manager,value=<key_manager_name>)
```

Example

```
/subsystem=elytron/server-ssl-context=applicationSSC:write-attribute(name=key-
manager,value=exampleServerKeyManager)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

4. Reload the server.

```
reload
```

Verification

- Verify that you can access the JBoss EAP welcome page.

    a. Verify using the CLI:

    Syntax

    ```
    $ curl --verbose --location --cacert <server_certificate> --cert
    <client_keystore>:<password> --cert-type P12 https://localhost:8443
    ```

    Example

    ```
    $ curl --verbose --location --cacert server.cer --cert exampleclient.keystore.pkcs12:secret
    ```

```
--cert-type P12 https://localhost:8443
...
<h3>Your Red Hat JBoss Enterprise Application Platform is running.</h3>
...
```

b. Verify using a browser.

   i. Import the client certificate into your browser. The example certificate created in the Generating client certificates procedure is called **exampleclient.keystore.pkcs12** and the example password to import it is **secret**.
      Refer to your browser's documentation for information about importing certificates to the browser.

   ii. Navigate to **https://localhost:8443** in a browser.
      The browser prompts you to present a certificate to identify with the server.

   iii. Choose the certificate you imported to the browser. For example, **exampleclient.keystore.pkcs12**.
      If you use a self-signed certificate, the browser presents a warning that the certificate presented by the server is unknown.

   iv. Inspect the certificate and verify that the fingerprints shown in your browser match the fingerprints of the certificate in your keystore. You can view the certificate in a keystore with the following command:

      Syntax

      ```
      /subsystem=elytron/key-store=<server_keystore_name>:read-alias(alias=<alias>)
      ```

      Example

      ```
      /subsystem=elytron/key-store=exampleServerKeyStore:read-alias(alias=localhost)
      ...
      "sha-1-digest" => "5e:3e:ad:c8:df:d7:f6:63:38:05:e2:a3:a7:31:07:82:c8:c8:94:47",
      "sha-256-digest" =>
      "11:b6:8f:00:42:e1:7f:6c:16:ef:db:08:5e:13:d9:b8:16:6e:a0:3c:2e:d4:e5:fd:cb:53:90:88:
      d2:9c:b1:99",
      ```

After you accept the server certificate, you can access JBoss EAP welcome page.

Two-way SSL/TLS is now configured for applications.

Additional resources

- **server-ssl-context** attributes

CHAPTER 3. CONFIGURING CERTIFICATE REVOCATION CHECKS IN ELYTRON

To ensure that certificates that are revoked by the issuing Certificate Authority (CA) before their expiration date are not trusted by Elytron or the Elytron client, configure certificate revocation checks. You can use either Certificate Revocation Lists (CRL) or an Online Certificate Status Protocol (OCSP) responder for certificate revocation checking. Use OCSP if you do not want to download the entire CRL.

3.1. CONFIGURING CERTIFICATE REVOCATION CHECKS USING CERTIFICATE REVOCATION LISTS

Configure certificate revocation checks using Certificate Revocation Lists (CRL) in the Elytron trust manager used for enabling two-way SSL/TLS, so that the certificates that are revoked by the issuing Certificate Authority (CA) before their expiration date are not trusted by Elytron.

Prerequisites

- JBoss EAP is running.
- A trust manager is configured.
  For more information, see Configuring a trust store and a trust manager for client certificates .

Procedure

1. Configure the trust manager to use the CRL using one of the following steps:
   - Configure the trust manager to use CRLs obtained from distribution points referenced in your certificates.

     Syntax

     ```
     /subsystem=elytron/trust-manager=<trust_manager_name>:write-attribute(name=certificate-revocation-lists,value=[])
     ```

     Example

     ```
     /subsystem=elytron/trust-manager=exampleTLSTrustManager:write-attribute(name=certificate-revocation-lists,value=[])
     ```

   - Override the CRL obtained from distribution points referenced in your certificates.

     Syntax

     ```
     /subsystem=elytron/trust-manager=<trust_manager_name>:write-attribute(name=certificate-revocation-lists,value=[{path="<CRL-file-1>"},{path="<CRL-file-2>"},...,{path="<CRL-file-N>"}])
     ```

     Example

     ```
     /subsystem=elytron/trust-manager=exampleTLSTrustManager:write-attribute(name=certificate-revocation-lists,value=[{path="intermediate.crl.pem"}])
     ```

2. Configure the trust manager to use CRL for certificate revocation checking.

    - If an OCSP responder is also configured for certificate revocation checks, add attribute **ocsp.prefer-crls** with the value **true** in the trust manager to use CRL for certificate revocation checking:

      Syntax

      ```
      /subsystem=elytron/trust-manager=<trust_manager_name>:write-
      attribute(name=ocsp.prefer-crls,value="true")
      ```

      Example

      ```
      /subsystem=elytron/trust-manager=exampleTLSTrustManager:write-
      attribute(name=ocsp.prefer-crls,value="true")
      ```

    - If no OCSP responder is configured for certificate revocation checks, the configuration is complete.

Additional resources

- **trust-manager** attributes

3.2. CONFIGURING CERTIFICATE REVOCATION CHECKS USING OCSP IN ELYTRON

Configure the trust manager used for enabling two-way SSL/TLS to use an Online Certificate Status Protocol (OCSP) responder for certificate revocation checking. OCSP is defined in RFC6960.

When both the OCSP responder and the CRL are configured for certificate revocation checks, the OCSP responder is invoked by default.

Prerequisites

- JBoss EAP is running.
- A trust manager is configured.
  For more information, see Configuring a trust store and a trust manager for client certificates .

Procedure

- Configure the trust manager for certification revocation using OCSP using either of the following steps:

    - Configure the trust manager to use the OCSP responder defined in the certificate for certificate revocation checking.

      Syntax

      ```
      /subsystem=elytron/trust-manager=<trust_manager_name>:write-
      attribute(name=ocsp,value={})
      ```

      Example

```
/subsystem=elytron/trust-manager=exampleTLSTrustManager:write-attribute(name=ocsp,value={})
```

- Override the OCSP responder defined in the certificate.

  Syntax

  ```
  /subsystem=elytron/trust-manager=<trust_manager_name>:write-attribute(name=ocsp.responder,value="<ocsp_responeder_url>")
  ```

  Example

  ```
  /subsystem=elytron/trust-manager=exampleTLSTrustManager:write-attribute(name=ocsp.responder,value="http://example.com/ocsp-responder")
  ```

Additional resources

- **trust-manager** attributes

3.3. CONFIGURING CERTIFICATE REVOCATION CHECKS USING CRL IN THE ELYTRON CLIENT

Configure certificate revocation checks using Certificate Revocation Lists (CRL) in the Elytron client, so that the certificates that are revoked by the issuing Certificate Authority (CA) before their expiration date are not trusted by the client.

Prerequisites

- You have created the **wildfly-config.xml** file for the Elytron client.

Procedure

- Add the following content in the **<ssl-context>** element in the **wildfly-config.xml** file:

  Syntax

  ```
  <certificate-revocation-lists>
      <certificate-revocation-list path="${path_to_crl}"/>
  </certificate-revocation-lists>
  ```

  Example

  ```
  <certificate-revocation-lists>
      <certificate-revocation-list path="/server/ca/crl/revoked.pem"/>
  </certificate-revocation-lists>
  ```

Additional resources

- **trust-manager** attributes

3.4. CONFIGURING CERTIFICATE REVOCATION CHECKS USING OCSP IN THE ELYTRON CLIENT

Configure certificate revocation checks using Online Certificate Status Protocol (OCSP) in the Elytron client, so that the certificates that are revoked by the issuing Certificate Authority (CA) before their expiration date are not trusted by the client. When you use an OCSP responder, you do not have to download the entire CRL.

Prerequisites

- You have created the **wildfly-config.xml** file for the Elytron client.

Procedure

- Add the following content in the **<ssl-context>** element in wildfly-config.xml:

  #### Syntax

  ```
  <ocsp responder="${ocsp_responder_uri}" responder-
  certificate="${alias_of_ocsp_responder_certificate}" responder-
  keystore="${keystore_for_ocsp_responder_certificate}" />
  ```

  #### Example

  ```
  <ocsp />
  ```

Additional resources

- **trust-manager** attributes

CHAPTER 4. USING ELYTRON CLIENT DEFAULT SSLCONTEXT SECURITY PROVIDER IN JBOSS EAP CLIENTS

To make a Java Virtual Machine (JVM) use the Elytron client configuration to provide a default **SSLcontext**, you can use the **WildFlyElytronClientDefaultSSLContextProvider**. Use this provider to make your client libraries automatically use the Elytron client configuration when requesting the default **SSLContext**.

4.1. ELYTRON CLIENT DEFAULT SSL CONTEXT SECURITY PROVIDER

Elytron client provides a Java security provider, **org.wildfly.security.auth.client.WildFlyElytronClientDefaultSSLContextProvider**, that you can use to register a Java virtual machine (JVM)-wide default SSL context.

The **WildFlyElytronClientDefaultSSLContextProvider** provider works as follows:

- The provider instantiates the **SSLContext** when the **SSLContext.getDefault()** method is called. The **SSLContext** is initiated from the authentication context obtained from one of the following places:
    - Elytron client configuration file passed as an argument to the provider.
    - Automatically discovered **wildfly-config.xml** file on the filesystem. For more information, see The Default Configuration Approach .
      A client configuration file passed as an argument to the provider has the precedence.
- When the **SSLContext.getDefault()** method is called, the JVM returns the **SSLContext** instantiated by the provider.
  As the Elytron client can have multiple SSL contexts configured, rules are used to choose a single SSL context for the connection. The default SSL Context is the one that matches all rules. The provider returns this default SSL context.

NOTE

If no default **SSLContext** is configured or no configuration is present, the provider will be ignored.

When you register the **WildFlyElytronClientDefaultSSLContextProvider** provider, all client libraries that use **SSLContext.getDefault()** method use the Elytron client configuration without having to use Elytron client APIs in their code.

To register the provider, you must add runtime dependency on the following artifacts:

- **wildfly-elytron-client**
- **wildfly-client-config**

You can register the provider either programmatically, in your client code, or statically in the **java.security** file. Use programmatic registration when you want to decide dynamically what providers to register and use.

Registering the provider programmatically

You can register the provider programmatically in your client code as shown below:

```
Security.insertProviderAt(new
WildFlyElytronClientDefaultSSLContextProvider(CONFIG_FILE_PATH), 1);
```

Registering the provider statically

You can register the provider in the **java.security** file as shown below:

```
security.provider.1=org.wildfly.security.auth.client.WildFlyElytronClientDefaultSSLContextProvider
<CONFIG_FILE_PATH>
```

Additional resources

- Example of creating a client that loads the default SSLContext

4.2. EXAMPLE OF CREATING A CLIENT THAT LOADS THE DEFAULT SSL CONTEXT

The following example demonstrates registering of the **WildFlyElytronClientDefaultSSLContextProvider** provider programmatically and using the **SSLContext.getDefault()** method to obtain the SSLContext initialized by an Elytron client. The example uses static client configuration supplied as an argument to the provider.

4.2.1. Creating a Maven project for JBoss EAP client

To create a client for applications deployed to JBoss EAP, create a Maven project with the required dependencies and the directory structure.

Prerequisites

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

Procedure

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

   ```
   $ mvn archetype:generate \
   -DgroupId=com.example.client \
   -DartifactId=client-ssl-context \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-quickstart \
   -DinteractiveMode=false
   ```

2. Navigate to the application root directory.

   ```
   $ cd client-ssl-context
   ```

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.client</groupId>
  <artifactId>client-ssl-context</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>client-ssl-context</name>

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

  <repositories>
    <repository>
      <id>jboss-public-maven-repository</id>
      <name>JBoss Public Maven Repository</name>
      <url>https://repository.jboss.org/nexus/content/groups/public/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </snapshots>
      <layout>default</layout>
    </repository>
    <repository>
      <id>redhat-ga-maven-repository</id>
      <name>Red Hat GA Maven Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>never</updatePolicy>
      </snapshots>
      <layout>default</layout>
    </repository>
  </repositories>

  <dependencies>
    <dependency>                              1
      <groupId>org.wildfly.security</groupId>
      <artifactId>wildfly-elytron-client</artifactId>
      <version>2.0.0.Final-redhat-00001</version>
    </dependency>
    <dependency>                              2
      <groupId>org.wildfly.client</groupId>
      <artifactId>wildfly-client-config</artifactId>
```

```
        <version>1.0.1.Final-redhat-00001</version>
      </dependency>
    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.4.0</version>
          <configuration>
            <mainClass>com.example.client.App</mainClass>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </project>
```

1 Dependency for **wildfly-elytron-client**.

2 Dependency for **wildfly-client-config**.

4. Delete the **src/test** directory.

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

Verification

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

```
$ mvn install
```

You get an output similar to the following:

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.682 s
[INFO] Finished at: 2023-10-31T01:32:17+05:30
[INFO] ------------------------------------------------------------------------
```

Next steps

- Creating a client that loads the default SSLContext

4.2.2. Creating a client that loads the default SSLContext

Create a client for applications deployed to JBoss EAP that loads the **SSLContext** by using the **SSLContext.getDefault()** method.

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

Prerequisites

- You have secured the applications deployed to JBoss EAP with two-way TLS.
  To do this, follow these procedures:
  - Generate client certificates.
  - Configure a trust store and a trust manager for client certificate
  - Configure a server certificate for two-way SSL/TLS
  - Configure SSL context to secure applications deployed onJBoss EAP with SSL/TLS
- You have created a Maven project.
  For more information, see Creating a Maven project for JBoss EAP client .
- JBoss EAP is running.

Procedure

1. Create a directory to store the Java files.

```
$ mkdir -p <application_home>/src/main/java/com/example/client
```

2. Navigate to the new directory.

```
$ cd <application_home>/src/main/java/com/example/client
```

3. Create the Java file **App.java** with the following content:

```
package com.example.client;

import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.net.http.HttpResponse.BodyHandlers;
import java.security.NoSuchAlgorithmException;
import java.security.Security;
import java.util.Properties;
import javax.net.ssl.SSLContext;
import org.wildfly.security.auth.client.WildFlyElytronClientDefaultSSLContextProvider;

public class App  {

  public static void main( String[] args )  {
  String url = "https://localhost:8443/";                                                                  1
  try {
    Security.insertProviderAt(new WildFlyElytronClientDefaultSSLContextProvider("src/wildfly-
config-two-way-tls.xml"), 1);  2
    HttpClient httpClient = HttpClient.newBuilder().sslContext(SSLContext.getDefault()).build();
    HttpRequest request = HttpRequest.newBuilder()
              .uri(URI.create(url))
              .GET()
```

```
            .build();
    HttpResponse<Void> httpRresponse = httpClient.send(request,
BodyHandlers.discarding());
    String sslContext = SSLContext.getDefault().getProvider().getName();
3
    System.out.println ("\nSSL Default SSLContext is: " + sslContext);

  } catch (NoSuchAlgorithmException | IOException | InterruptedException e) {
      e.printStackTrace();
    }

  System.exit(0);
  }
}
```

1 Defines the JBoss EAP home page URL.

2 Register the security provider. **1** defines the priority for this provider. To statically register the provider, you can instead add the provider in the **java.security** file as: **security.provider.1=org.wildfly.security.auth.client.WildFlyElytronClientDefaultSSLContextProvider *<PATH>*/*<TO>*/wildfly-config-two-way-tls.xml**

3 Obtain the default SSL context.

4. Create the client configuration file called "wildfly-config-two-way-tls.xml" in the ***<application_home>*/src** directory.

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

<configuration>
    <authentication-client xmlns="urn:elytron:client:1.7">
        <key-stores>
            <key-store name="truststore" type="PKCS12">
                <file name="${path_to_client_truststore}/client.truststore.p12"/>
                <key-store-clear-password password="secret"/>
            </key-store>
            <key-store name="keystore" type="PKCS12">
                <file name="${path_to_client_keystore}/exampleclient.keystore.pkcs12"/>
                <key-store-clear-password password="secret"/>
            </key-store>
        </key-stores>
        <ssl-contexts>
            <ssl-context name="client-context">
                <trust-store key-store-name="truststore"/>
                <key-store-ssl-certificate key-store-name="keystore"
alias="exampleclientkeystore">
                    <key-store-clear-password password="secret"/>
                </key-store-ssl-certificate>
            </ssl-context>
        </ssl-contexts>
        <ssl-context-rules>
            <rule use-ssl-context="client-context"/>
        </ssl-context-rules>
    </authentication-client>
</configuration>
```

Replace the following place holder values with actual paths:

- *${path_to_client_truststore}*
- *${path_to_client_keystore}*

Verification

1. Navigate to the *<application_home>* directory.
2. Run the application.

```
$ mvn compile exec:java
```

Example output

```
INFO: ELY00001: WildFly Elytron version 2.0.0.Final-redhat-00001

SSL Default SSLContext is: WildFlyElytronClientDefaultSSLContextProvider
```

CHAPTER 5. REFERENCE

5.1. KEY-MANAGER ATTRIBUTES

You can configure a **key-manager** by setting its attributes.

Table 5.1. key-manager attributes

Attribute	Description
algorithm	The name of the algorithm to use to create the underlying **KeyManagerFactory**. This is provided by the JDK. For example, a JDK that uses SunJSSE provides the **PKIX** and **SunX509** algorithms. For more information, see theSupport Classes and Interfaces on the Oracle website.
alias-filter	A filter to apply to the aliases returned from the keystore. This can either be a comma-separated list of aliases to return or one of the following formats: ● **ALL:-alias1:-alias2** ● **NONE:+alias1:+alias2**
credential-reference	The credential reference to decrypt keystore item. This can be specified in clear text or as a reference to a credential stored in a **credential-store**. This is not a password of the keystore.
generate-self-signed-certificate-host	If the file that backs the keystore does not exist and this attribute is set, then a self-signed certificate is generated for the specified host name. Do not set this attribute in a production environment.
key-store	Reference to the **key-store** to use to initialize the underlying **KeyManagerFactory**.
provider-name	The name of the provider to use to create the underlying **KeyManagerFactory**.
providers	Reference to obtain the **Provider[]** to use when creating the underlying **KeyManagerFactory**.

5.2. KEY-STORE ATTRIBUTES

You can configure a **key-store** by setting its attributes.

Table 5.2. key-store attributes

Attribute	Description
alias-filter	A filter to apply to the aliases returned from the keystore, can either be a comma separated list of aliases to return or one of the following formats: • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2** NOTE The **alias-filter** attribute is case sensitive. Because the use of mixed-case or uppercase aliases, such as **elytronAppServer**, might not be recognized by some keystore providers, it is recommended to use lowercase aliases, such as **elytronappserver**.
credential-reference	The password to use to access the keystore. This can be specified in clear text or as a reference to a credential stored in a **credential-store**.
path	The path to the keystore file.
provider-name	The name of the provider to use to load the keystore. When you set this attribute, the search for the first provider that can create a key store of the specified type is disabled.
providers	A reference to the providers that should be used to obtain the list of provider instances to search. If not specified, the global list of providers will be used instead.
relative-to	The base path this store is relative to. This can be a full path or a predefined path such as **jboss.server.config.dir**.
required	If set to **true**, the key store file referenced must exist at the time the key store service starts. The default value is **false**.

Attribute	Description
type	The type of the key store, for example, **JKS**. NOTE The following key store types are automatically detected: • **JKS** • **JCEKS** • **PKCS12** • **BKS** • **BCFKS** • **UBER** You must manually specify the other key store types. A full list of key store types can be found in Java Cryptography Architecture Standard Algorithm Name Documentation for JDK 11 in the Oracle JDK documentation.

5.3. SERVER-SSL-CONTEXT ATTRIBUTES

You can configure the server SSL context, **server-ssl-context**, by setting its attributes.

Table 5.3. **server-ssl-context** attributes

Attribute	Description

Attribute	Description
authentication-optional	If **true** rejecting of the client certificate by the security domain will not prevent the connection. This allows a fall through to use other authentication mechanisms, such as form login, when the client certificate is rejected by security domain. This has an effect only when the security domain is set. This defaults to **false**.
cipher-suite-filter	The filter to apply to specify the enabled cipher suites. This filter takes a list of items delimited by colons, commas, or spaces. Each item may be an OpenSSL-style cipher suite name, a standard SSL/TLS cipher suite name, or a keyword such as **TLSv1.2** or **DES**. A full list of keywords as well as additional details on creating a filter can be found in the Javadoc for the **CipherSuiteSelector** class. The default value is **DEFAULT**, which corresponds to all known cipher suites that do not have **NULL** encryption and excludes any cipher suites that have no authentication.
cipher-suite-names	The filter to apply to specify the enabled cipher suites for TLSv1.3.
final-principal-transformer	A final principal transformer to apply for this mechanism realm.
key-manager	Reference to the key managers to use within the **SSLContext**.
maximum-session-cache-size	The maximum number of SSL/TLS sessions to be cached.
need-client-auth	If set to **true**, a client certificate is required on SSL handshake. Connection without a trusted client certificate will be rejected. This defaults to **false**.
post-realm-principal-transformer	A principal transformer to apply after the realm is selected.
pre-realm-principal-transformer	A principal transformer to apply before the realm is selected.

Attribute	Description
protocols	The enabled protocols. Allowed options are ● **SSLv2** ● **SSLv3** ● **TLSv1** ● **TLSv1.1** ● **TLSv1.2** ● **TLSv1.3** This defaults to enabling **TLSv1**, **TLSv1.1**, **TLSv1.2**, and **TLSv1.3**. WARNING Use TLSv1.2, or TLSv1.3 instead of SSLv2, SSLv3, and TLSv1.0. Using SSLv2, SSLv3, or TLSv1.0 poses a security risk, therefore you must explicitly disable them. If you do not specify a protocol, configuring **cipher-suite-names** sets the value of**protocols** to **TLSv1.3**.
provider-name	The name of the provider to use. If not specified, all providers from **providers** will be passed to the**SSLContext**.
providers	The name of the providers to obtain the **Provider[]** to use to load the **SSLContext**.
realm-mapper	The realm mapper to be used for SSL/TLS authentication.
security-domain	The security domain to use for authentication during SSL/TLS session establishment.

Attribute	Description
session-timeout	The timeout for SSL sessions, in seconds. The value **-1** directs Elytron to use the Java Virtual Machine (JVM) default value. The value **0** indicates that there is timeout. The default value is **-1**.
trust-manager	Reference to the **trust-manager** to use within the SSLContext.
use-cipher-suites-order	If set to **true** the cipher suites order defined on the server is used. If set to **false** the cipher suites order presented by the client is used. Defaults to **true**.
want-client-auth	If set to **true** a client certificate is requested, but not required, on SSL handshake. If a security domain is referenced and supports X509 evidence, **want-client-auth** is set to **true** automatically. This is ignored when **need-client-auth** is set. This defaults to **false**.
wrap	If **true**, the returned **SSLEngine**, **SSLSocket**, and **SSLServerSocket** instances are wrapped to protect against further modification. This defaults to **false**.

NOTE

The **realm-mapper** and **principal-transformer** attributes for **server-ssl-context** apply only for the SASL EXTERNAL mechanism, where the certificate is verified by the trust manager. HTTP CLIENT-CERT authentication settings are configured in an **http-authentication-factory**.

5.4. TRUST-MANAGER ATTRIBUTES

You can configure the trust manager, **trust-manager**, by setting its attributes.

Table 5.4. trust-manager attributes

Attribute	Description
algorithm	The name of the algorithm to use to create the underlying **TrustManagerFactory**. This is provided by the JDK. For example, a JDK that uses SunJSSE provides the **PKIX** and **SunX509** algorithms. More details on SunJSSE can be found in the Support Classes and Interfaces in Java Secure Socket Extension (JSSE) Reference Guide in Oracle documentation.

Attribute	Description
alias-filter	A filter to apply to the aliases returned from the key store. This can either be a comma-separated list of aliases to return or one of the following formats: • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**
certificate-revocation-list	Enables certificate revocation list checks in a trust manager. You can only define a single CRL path using this attribute. To define multiple CRL paths, use **certificate-revocation-lists**. The attributes of **certificate-revocation-list** are: • **maximum-cert-path** - The maximum number of non-self-issued intermediate certificates that can exist in a certification path. The default value is **5**. This attribute has been deprecated. Use **maximum-cert-path** in **trust-manager** instead. • **path** - The path to the certificate revocation list. • **relative-to** - The base path of the certificate revocation list file.
certificate-revocation-lists	Enables certificate revocation list checks in a trust manager using multiple certificate revocation lists. The attributes of **certificate-revocation-list** are: • **path** - The path to the certificate revocation list. • **relative-to** - The base path of the certificate revocation list file.
key-store	Reference to the **key-store** to use to initialize the underlying **TrustManagerFactory**.
maximum-cert-path	The maximum number of non-self-issued intermediate certificates that can exist in a certification path. The default value is **5**. This attribute has been moved to **trust-manager** from **certificate-revocation-list** inside **trust-manager** in JBoss EAP 7.3. For backward compatibility, the attribute is also present in **certificate-revocation-list**. Going forward, use **maximum-cert-path** in **trust-manager**. NOTE Define **maximum-cert-path** in either **trust-manager** or in **certificate-revocation-list** not in both.

Attribute	Description
ocsp	Enables online certificate status protocol (OCSP) checks in a trust manager. The attributes of **ocsp** are: • **responder** - Overrides the OCSP Responder URI resolved from the certificate. • **responder-certificate** - Alias for responder certificate located in **responder-keystore** or **trust-manager** key store if**responder-keystore** is not defined. • **responder-keystore** - Alternative keystore for responder certificate. **responder-certificate** must be defined. • **prefer-crls** - When both OCSP and CRL mechanisms are configured, OCSP mechanism is called first. When **prefer-crls** is set to **true**, the CRL mechanism is called first.
only-leaf-cert	Check revocation status of only the leaf certificate. This is an optional attribute. The default values is **false**.
provider-name	The name of the provider to use to create the underlying **TrustManagerFactory**.
providers	Reference to obtain the providers to use when creating the underlying **TrustManagerFactory**.