Red Hat JBoss Enterprise Application Platform 8-beta

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: 2022-12-16

Red Hat JBoss Enterprise Application Platform 8-beta 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 RED HAT DOCUMENTATION

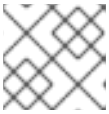
We appreciate your feedback on our documentation. To provide feedback, you can highlight the text in a document and add comments. Follow the steps in the procedure to learn about submitting feedback on Red Hat documentation.

Prerequisites

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

Procedure

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



NOTE

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

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

MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 1. 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.

1.1. 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 .

1.2. 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

1.3. 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
```

Additional resources

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

1.4. 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 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 server you generated. You can see 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=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

1.5. 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 server you generated. You can see 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=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 2. 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.

2.1. 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

2.2. 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

2.3. 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

2.4. 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 3. REFERENCE

3.1. KEY-MANAGER ATTRIBUTES

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

Table 3.1. 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.
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**.

3.2. KEY-STORE ATTRIBUTES

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

Table 3.2. key-store attributes

Attribute	Description

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.

3.3. SERVER-SSL-CONTEXT ATTRIBUTES

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

Table 3.3. server-ssl-context attributes

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.

Attribute	Description
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.
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.
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**.

Attribute	Description
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.
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**.

3.4. TRUST-MANAGER ATTRIBUTES

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

Table 3.4. trust-manager attributes

Attribute	Description

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.
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**.

Attribute	Description
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.
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**.