Red Hat JBoss Enterprise Application Platform 8.0

在 JBoss EAP 中配置 SSL/TLS

在 JBoss EAP 中启用 SSL/TLS 的指南来保护 JBoss EAP 管理界面和部署的应用程序

Last Updated: 2024-02-09

Red Hat JBoss Enterprise Application Platform 8.0 在 JBoss EAP 中配置 SSL/TLS

在 JBoss EAP 中启用 SSL/TLS 的指南来保护 JBoss EAP 管理界面和部署的应用程序

法律通告

摘要

在 JBoss EAP 中启用 SSL/TLS 以确保 JBoss EAP 管理接口和部署的应用程序的指南。

目录

提供有关 JBOSS EAP 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 为管理界面和应用程序启用单向 SSL/TLS

SSL/TLS 或传输层安全(TLS)是基于证书的安全协议，用于保护通过网络通信的两个实体之间的数据传输。

您可以为 JBoss EAP 管理界面和 JBoss EAP 上部署的应用程序启用单向 SSL/TLS。如需更多信息，请参阅以下步骤：

- 为管理界面启用单向 SSL/TLS。
- 为在 JBoss EAP 上部署的应用程序启用单向 SSL/TLS。

1.1. 为管理界面启用单向 SSL/TLS

为管理界面启用单向 SSL/TLS，以便 JBoss EAP 管理接口和连接接口的客户端之间的通信是安全的。

要为管理界面启用单向 SSL/TLS，您可以使用以下步骤：

- 使用向导为管理接口启用单向 SSL/TLS ：使用此流程使用基于 CLI 的向导快速设置 SSL/TLS。Elytron 根据您的向导的输入创建所需资源。
- 使用子系统命令为管理接口启用单向 SSL/TLS ：使用此流程配置必要的资源，以手动启用 SSL/TLS。手动配置资源可让您更好地控制服务器配置。

另外，您可以使用 安全命令禁用 SSL/TLS 管理接口的步骤，为管理接口禁用 SSL/TLS。

1.1.1. 使用向导为管理接口启用单向 SSL/TLS

Elytron 提供了一个向导来快速设置 SSL/TLS。您可以使用包含证书的现有密钥存储，或者使用向导生成的密钥存储和自签名证书来启用 SSL/TLS。您还可以使用 **--lets-encrypt** 选项从 Let 的 Encrypt 证书颁发机构获取和使用证书。有关 Let's Encrypt 的详情，请查看 Let 的加密文档。

使用向导生成的自签名证书来启用 SSL/TLS，仅用于测试和开发目的。对于生产环境，始终使用证书颁发机构(CA)签名证书。

重要

不要在生产环境中使用自签名证书。仅使用证书颁发机构(CA)签名的证书。

向导配置为管理接口启用 SSL/TLS 所需的以下资源：

- **key-store**
- **key-manager**
- **server-ssl-context**
- 然后，**server-ssl-context** 应用到 **http-interface**。

Elytron 将每个资源命名为 *resource-type-UUID*。例如，key-store-9e35a3be-62bb-4fff-afc2-2d8d141b82bc。通用唯一标识符(UUID)有助于避免资源的名称冲突。

先决条件

- JBoss EAP 正在运行。

流程

- 在管理 CLI 中输入以下命令来启动向导，为管理接口配置单向 SSL/TLS。

  语法

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

  提示时输入所需信息。

  使用 **--lets-encrypt** 选项从 Let 的 Encrypt 证书颁发机构获取和使用证书。

  如果已经为管理界面启用了 SSL/TLS，向导会退出，并显示以下信息：

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

  要更改现有配置，首先禁用对管理接口的 SSL/TLS，然后创建新配置。有关为管理接口禁用 SSL/TLS 的详情，请参考使用 向导为管理接口禁用 SSL/TLS。

  注意

  要启用单向 SSL/TLS，请在提示启用 SSL 相互身份验证时输入 **n** 或 blank。设置 mutual 身份验证可启用双向 SSL/TLS。

  以互动方式使用向导的示例

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

  向导提示输入示例

  ```
  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
```

输入 **y** 后，服务器将重新加载。如果您配置了自签名证书，请使用向导来生成自签名证书或配置了 Java 虚拟机(JVM)信任的证书，管理 CLI 会提示您接受服务器提供的证书。

```
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 :
```

输入 **T** 或 **P** 以继续连接。

您将获得以下输出：

```
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
```

验证

- 使用管理 CLI 客户端连接来验证 SSL/TLS。
  您可以通过将 Elytron 客户端 SSL 上下文放在配置文件中，然后使用管理 CLI 连接到服务器并引用配置文件来测试 SSL/TLS。

  a. 前往包含密钥存储文件的目录。在本例中，密钥存储文件 **exampleKeystore.pkcs12** 在服务器的 **standalone/configuration** 目录中生成。

  Example

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

  b. 使用服务器证书创建客户端 **trust-store**。

  语法

  ```
  $ 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
```

如果您使用了自签名证书，系统会提示您信任证书。

c. 在文件中定义客户端 SSL 上下文，如 **example-security.xml**。

语法

```
<?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. 连接到服务器并发出命令。

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
```

预期输出

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

- 使用浏览器验证 SSL/TLS。

a. 进入 https://localhost:9993。
如果您使用了自签名证书，浏览器会显示一个警告，表示服务器出示的证书未知。

b. 检查证书，并验证浏览器中显示的指纹是否与密钥存储中的证书的指纹匹配。您可以使用以下命令查看生成的证书：

语法

```
/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")
```

您可以从向导的输出中获取密钥存储名称，例如 "key-store 为 key-store-a18ba30e-6a26-4ed6-87c5-feb7f3e4dff1"。

输出示例

```
...
"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",
...
```

接受服务器证书后，会提示您输入登录凭证。您可以使用现有 JBoss EAP 用户的用户凭证登录。

现在为 JBoss EAP 管理界面启用了 SSL/TLS。

其他资源

- **key-manager** 属性
- **key-store** 属性

- **server-ssl-context** 属性

1.1.2. 使用 subsystem 命令为管理接口启用单向 SSL/TLS

使用 **elytron** 子系统命令通过 SSL/TLS 保护 JBoss EAP 管理接口。

出于测试和开发目的，您可以使用自签名证书。您可以使用包含证书的现有密钥存储，或者在创建 **key-store** 资源时使用 Elytron 生成的密钥存储。对于生产环境，始终使用证书颁发机构(CA)签名证书。

重要

不要在生产环境中使用自签名证书。仅使用证书颁发机构(CA)签名的证书。

先决条件

- JBoss EAP 正在运行。

流程

1. 配置密钥存储以存储证书。
   您可以提供到现有密钥存储的路径，例如，包含 CA 签名证书的路径，或者提供要创建的密钥存储的路径。

   ```
   /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. 如果密钥存储不包含任何证书，或者您使用上面的步骤来创建密钥存储，则必须生成证书并将证书存储在文件中。

   a. 在密钥存储中生成密钥对。

   语法

   ```
   /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. 将证书存储在文件中。

   语法

   -

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

Example

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

3. 配置引用 **key-store** 的 **key-manager**。

语法

```
/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})
```

重要

红帽没有指定 algorithm 属性，因为 **elytron** 子系统使用 **KeyManagerFactory.getDefaultAlgorithm （）** 来默认确定算法。但是，您可以指定 algorithm 属性。

要指定 algorithm 属性，您需要知道您使用的 Java Development Kit (JDK)提供哪些密钥管理器算法。例如，使用 Java 安全套接字扩展(SunJSSE)的 JDK 提供了 PKIX 和 SunX509 算法。

在命令中，您可以指定 SunX509 作为 **key-manager** 算法属性。

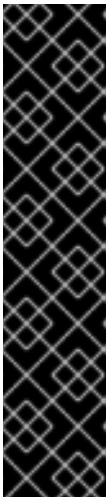
4. 配置引用 **key-manager** 的 **server-ssl-context**。

语法

```
/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"])
```



重要

您需要确定您要支持的 SSL/TLS 协议。示例命令使用 TLSv1.2。

- 对于 TLSv1.2 及更早版本，请使用 **cipher-suite-filter** 参数指定允许哪些密码套件。
- 对于 TLSv1.3，使用 **cipher-suite-names** 参数指定允许哪些密码套件。TLSv1.3 默认禁用。如果您没有使用 protocol 属性指定 **协议**，或者指定的集合包含 TLSv1.3，请配置 **cipher-suite-names** 启用 TLSv1.3。

使用 **use-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 security 子系统行为不同，它默认为遵循客户端密码套件顺序。

5. 更新管理界面，以使用配置的 **server-ssl-context**。

语法

```
/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. 重新加载服务器。

```
reload
```

如果您使用自签名证书启用 SSL/TLS，管理 CLI 会提示您接受服务器提供的证书。这是您通过配置密钥存储的证书。

输出示例

```
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 :
```

输入 **T** 或 **P** 以继续连接。

验证

- 通过客户端连接来验证 SSL/TLS。
  您可以通过在配置文件中放置 Elytron 客户端 SSL 上下文来测试 SSL/TLS，然后使用引用配置文件的管理 CLI 连接到服务器。

  a. 前往包含密钥存储文件的目录。在本例中，密钥存储文件 **exampleserver.keystore.pkcs12** 在服务器的 **standalone/configuration** 目录中生成。

  Example

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

  b. 导出服务器证书，以便将其导入到客户端信任存储中。

  ```
  $ 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. 使用服务器证书创建客户端 **trust-store**。

  语法

  ```
  $ 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
  ```

  如果您使用了自签名证书，系统会提示您信任证书。

  d. 在文件中定义客户端 SSL 上下文，如 **example-security.xml**。

  语法

  ```
  <?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. 连接到服务器并发出命令。

Example

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

预期输出

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

- 使用浏览器验证 SSL/TLS。

a. 进入 https://localhost:9993。
如果您使用了自签名证书，浏览器会显示一个警告，表示服务器出示的证书未知。

b. 检查证书，并验证浏览器中显示的指纹是否与密钥存储中的证书的指纹匹配。您可以使用以下命令查看生成的证书：

语法

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

Example

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

输出示例

```
...
"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",
...
```

接受服务器证书后，会提示您输入登录凭证。您可以使用现有 JBoss EAP 用户的用户凭证登录。

现在为 JBoss EAP 管理界面启用了 SSL/TLS。

其他资源

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

1.1.3. 使用安全 命令为管理接口禁用 SSL/TLS

使用 **security** 命令为管理界面禁用 SSL/TLS。您可能希望进行此操作，将不同的 SSL/TLS 配置用于配置的不同 SSL/TLS 配置。

使用 命令禁用 SSL/TLS 不会删除 Elytron 资源。命令只是取消定义 **http-interface management-interface** 资源的 **secure-socket-binding** 和 **ssl-context** 属性。

先决条件

- JBoss EAP 正在运行。

流程

- 在管理 CLI 中使用 **disable-ssl-management** 命令。

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

服务器使用以下输出重新加载：

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

您可以使用以下方法之一为服务器管理接口启用 SSL/TLS：

- 使用向导为管理接口启用单向 SSL/TLS ：使用此流程使用基于 CLI 的向导快速设置 SSL/TLS。Elytron 根据您的向导的输入创建所需资源。
- 使用子系统命令为管理接口启用单向 SSL/TLS ：使用此流程配置必要的资源，以手动启用 SSL/TLS。手动配置资源可让您更好地控制服务器配置。

1.2. 为在 JBOSS EAP 上部署的应用程序启用单向 SSL/TLS

为 JBoss EAP 上部署的应用启用单向 SSL/TLS，因此 Web 浏览器等应用与客户端之间的通信是安全的。

要为在 JBoss EAP 上部署的应用程序启用单向 SSL/TLS，您可以使用以下步骤：

- 使用自动 生成的自签名证书为应用程序启用 SSL/TLS ：仅在开发或测试环境中使用此流程。此流程可帮助您快速为应用程序启用 SSL/TLS，而无需进行任何配置。
- 使用向导为 JBoss EAP 上部署的应用程序启用单向 SSL/TLS ：使用此流程使用基于 CLI 的向导来快速设置 SSL/TLS。Elytron 根据您的向导的输入创建所需资源。
- 使用子系统命令为应用程序启用单向 SSL/TLS ：使用此方法配置所需资源以手动启用 SSL/TLS。手动配置资源可让您更好地控制服务器配置。

此外，您可以使用 安全命令禁用 SSL/TLS 的步骤，为 JBoss EAP 上部署的应用程序禁用 SSL/TLS 。

1.2.1. Elytron 中的默认 SSL 上下文

为了帮助开发人员为应用程序快速设置单向 SSL/TLS，**elytron** 子系统包含启用单向 SSL/TLS 所需的资源，默认准备在开发或测试环境中使用。

默认提供以下资源：

- 名为 **applicationKS** 的 **key-store**。
- **key-manager**，名为 **applicationKM**，引用 **key-store**。
- 名为 **applicationSSC** 的 **server-ssl-context**，引用 **key-manager**。

默认 TLS 配置

```
...
<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>
  ...
```

默认 **key-manager applicationKM** 包含一个 **generate-self-signed-certificate-host** 属性，值为 **localhost**。**generate-self-signed-certificate-host** 属性表示当使用此 **key-manager** 获取服务器证书时，如果支持其 **key-store** 的文件不存在，则 key-manager 应该自动生成自签名证书，并将 **localhost** 用作 **通用名称**。生成的自签名证书存储在支持 **key-store** 的文件中。

由于安装服务器时支持默认 key-store 的文件，只需向服务器发送 https 请求会生成自签名证书，并为应用程序启用单向 SSL/TLS。如需更多信息，请参阅使用自动生成的自签名证书为应用程序启用 SSL/TLS。

其他资源

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

1.2.2. 使用自动生成的自签名证书为应用程序启用 SSL/TLS

当服务器第一次收到 HTTPS 请求时，JBoss EAP 会自动生成自签名证书。**elytron** 子系统还包含可在开发或测试环境中使用的 **key-store**、**key-manager** 和 **server-ssl-context** 资源。因此，当 JBoss EAP 生成自签名证书后，应用程序就会使用证书进行保护。

重要

不要在生产环境中使用自签名证书。仅使用证书颁发机构(CA)签名的证书。

先决条件

- JBoss EAP 正在运行。

流程

- 导航到端口 **8443** 上的服务器 URL，例如 https://localhost:8443。
  JBoss EAP 在收到此请求时生成自签名证书。有关此证书的详情，您可以查看服务器日志。

  浏览器会将连接标记为不安全，因为生成的证书是自签名的。

验证

1. 将显示的证书 JBoss EAP 与浏览器与服务器日志中的证书进行比较。

   服务器日志示例

   ```
   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
```

向浏览器显示的证书示例

```
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. 如果指纹匹配，如示例中，您可以继续进入该页面。

为应用程序启用 SSL/TLS。

其他资源

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

1.2.3. 使用向导为 JBoss EAP 上部署的应用程序启用单向 SSL/TLS

Elytron 提供了一个向导来快速设置 SSL/TLS。您可以使用包含证书的现有密钥存储，或者使用向导生成的密钥存储和自签名证书来启用 SSL/TLS。您还可以使用 **--lets-encrypt** 选项从 Let 的 Encrypt 证书颁发机构获取和使用证书。有关 Let's Encrypt 的详情，请查看 Let 的加密文档。

使用向导生成的自签名证书来启用 SSL/TLS，仅用于测试和开发目的。对于生产环境，始终使用证书颁发机构(CA)签名证书。

重要

不要在生产环境中使用自签名证书。仅使用证书颁发机构(CA)签名的证书。

向导配置为应用程序启用 SSL/TLS 所需的以下资源：

- **key-store**
- **key-manager**
- **server-ssl-context**
- 然后，**server-ssl-context** 应用到 Undertow **https-listener**。

Elytron 将每个资源命名为 *resource-type-UUID*。例如，key-store-9e35a3be-62bb-4fff-afc2-2d8d141b82bc。通用唯一标识符(UUID)有助于避免资源的名称冲突。

先决条件

- JBoss EAP 正在运行。

流程

- 在管理 CLI 中输入以下命令来启动向导来为应用程序配置单向 SSL/TLS：

  语法

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

  提示时输入所需信息。

  使用 **--lets-encrypt** 选项从 Let 的 Encrypt 证书颁发机构获取和使用证书。

  如果 **server-ssl-context** 已存在，向导会退出，并显示以下信息：

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

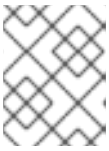
  

  注意

  elytron 子系统默认包含一个已配置的 **server-ssl-context** 资源。因此，您必须在全新安装后第一次启动向导时使用 **--override-ssl-context** 选项。

  如需更多信息，请参阅 Elytron 中的默认 SSL 上下文。

  如果您覆盖现有的 **server-ssl-context**，Elytron 将使用向导创建的 **server-ssl-context** 来启用 SSL。

  注意

  要启用单向 SSL/TLS，请在提示启用 SSL 相互身份验证时输入 **n** 或 blank。设置 mutual 身份验证可启用双向 SSL/TLS。

  启动向导示例

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

  向导提示输入示例

  ```
  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
```

输入 **y** 后，服务器会使用以下输出重新载入：

```
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
```

验证

1. 进入 https://localhost:8443。
   如果您使用了自签名证书，浏览器会显示一个警告，表示服务器出示的证书未知。
2. 检查证书，并验证浏览器中显示的指纹是否与密钥存储中的证书的指纹匹配。您可以使用以下命令查看生成的证书：

   语法

   ```
   /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")
   ```

   您可以从向导的输出中获取密钥存储名称，例如 "key-store 为 key-store-4cba6678-c464-4dcc-90ff-9295312ac395"。

   输出示例

   ```
   ...
   "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",
   ...
   ```

现在，在 JBoss EAP 上部署的应用程序启用了 SSL/TLS。

其他资源

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

1.2.4. 使用 subsystem 命令为应用程序启用单向 SSL/TLS

使用 **elytron** 子系统命令，使用 SSL/TLS 保护在 JBoss EAP 上部署的应用。

出于测试和开发目的，您可以使用自签名证书。您可以使用包含证书的现有密钥存储，或者在创建 **key-store** 资源时使用 Elytron 生成的密钥存储。对于生产环境，始终使用证书颁发机构(CA)签名证书。

重要

不要在生产环境中使用自签名证书。仅使用证书颁发机构(CA)签名的证书。

先决条件

- JBoss EAP 正在运行。

流程

1. 配置密钥存储以存储证书。
   您可以提供到现有密钥存储的路径，例如，包含 CA 签名证书的路径，或者提供要创建的密钥存储的路径。

   ```
   /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. 如果密钥存储不包含任何证书，或者您使用上面的步骤来创建密钥存储，则必须生成证书并将证书存储在文件中。

   a. 在密钥存储中生成密钥对。

   语法

   ```
   /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. 将证书存储在文件中。

语法

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

Example

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

3. 配置引用 **key-store** 的 **key-manager**。

语法

```
/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})
```

重要

红帽没有指定 algorithm 属性，因为 **elytron** 子系统使用 **KeyManagerFactory.getDefaultAlgorithm （）** 来默认确定算法。但是，您可以指定 algorithm 属性。

要指定 algorithm 属性，您需要知道您使用的 Java Development Kit (JDK)提供哪些密钥管理器算法。例如，使用 Java 安全套接字扩展(SunJSSE)的 JDK 提供了 PKIX 和 SunX509 算法。

在命令中，您可以指定 SunX509 作为 **key-manager** 算法属性。

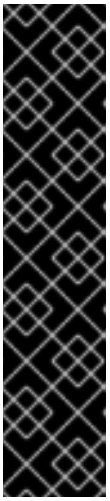
4. 配置引用 **key-manager** 的 **server-ssl-context**。

语法

```
/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"])
```



重要

您需要确定您要支持的 SSL/TLS 协议。示例命令使用 TLSv1.2。

- 对于 TLSv1.2 及更早版本，请使用 **cipher-suite-filter** 参数指定允许哪些密码套件。
- 对于 TLSv1.3，使用 **cipher-suite-names** 参数指定允许哪些密码套件。TLSv1.3 默认禁用。如果您没有使用 protocol 属性指定 协议，或者指定的集合包含 TLSv1.3，请配置 **cipher-suite-names** 启用 TLSv1.3。

使用 **use-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 security 子系统行为不同，它默认为遵循客户端密码套件顺序。

5. 更新 Undertow，以使用配置的 **server-ssl-context**。

语法

```
/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. 重新加载服务器。

```
reload
```

验证

1. 进入 https://localhost:8443。
如果您使用了自签名证书，浏览器会显示一个警告，表示服务器出示的证书未知。

2. 检查证书，并验证浏览器中显示的指纹是否与密钥存储中的证书的指纹匹配。您可以使用以下命令查看生成的证书：

语法

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

Example

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

输出示例

```
...
"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",
...
```

现在，在 JBoss EAP 上部署的应用程序启用了 SSL/TLS。

其他资源

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

1.2.5. 使用 security 命令为应用程序禁用 SSL/TLS

使用 **security** 命令为 JBoss EAP 上部署的应用程序禁用 SSL/TLS。使用 命令禁用 SSL/TLS 不会删除 Elytron 资源。命令仅将服务器的 **ssl-context** 设置为其默认值 **applicationSSC**。

先决条件

- JBoss EAP 正在运行。

流程

- 在管理 CLI 中使用 **security disable-ssl-http-server** 命令。

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

服务器使用以下输出重新加载：

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

您可以使用以下流程之一为在 JBoss EAP 上部署的应用程序启用 SSL/TLS：

- 使用自动 生成的自签名证书为应用程序启用 SSL/TLS ：仅在开发或测试环境中使用此流程。此流程可帮助您快速为应用程序启用 SSL/TLS，而无需进行任何配置。
- 使用向导为 JBoss EAP 上部署的应用程序启用单向 SSL/TLS ：使用此流程使用基于 CLI 的向导来快速设置 SSL/TLS。Elytron 根据您的向导的输入创建所需资源。
- 使用子系统命令为应用程序启用单向 SSL/TLS ：使用此方法配置所需资源以手动启用 SSL/TLS。手动配置资源可让您更好地控制服务器配置。

其他资源

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

第 2 章 为管理界面和应用程序启用双向 SSL/TLS

SSL/TLS 或传输层安全(TLS)是基于证书的安全协议，用于保护通过网络通信的两个实体之间的数据传输。如果您希望服务器只与可信客户端连接时，请使用双向 SSL/TLS。

双向 SSL/TLS 提供以下安全功能：

身份验证

在单向 SSL/TLS 中，服务器向客户端提供证书以验证其自身。在双向 SSL/TLS 中，客户端还向服务器提供证书来验证客户端。因此，双向 SSL/TLS 也称为 mutual 身份验证。

保密性

在客户端和服务器间传输的数据会被加密。

数据完整性

TLS 协议提供具有安全哈希功能的数据完整性，用于消息身份验证代码(MAC)计算。您可以使用 SSL 上下文资源的 cipher-suite- **filter 和 cipher-suite -names** 属性为连接强制实施特定的算法和哈希功能。

如需更多信息，请参阅 **server-ssl-context** 属性。

您可以使用双向 SSL/TLS 保护 JBoss EAP 管理接口和部署的应用程序。

要使用双向 SSL/TLS 保护管理接口，请使用以下流程：

- 从客户端的证书颁发机构(CA)获取证书。另外，对于非生产环境，您可以按照以下步骤生成自签名证书： 生成客户端证书。
- 为客户端证书配置信任存储和信任管理器
- 为双向 SSL/TLS 配置服务器证书。
- 配置 SSL 上下文以使用 SSL/TLS 保护 JBoss EAP 管理接口

要使用双向 SSL/TLS 保护在 JBoss EAP 上部署的应用程序，请使用以下流程：

- 从客户端的证书颁发机构(CA)获取证书。另外，对于非生产环境，您可以按照以下步骤生成自签名证书： 生成客户端证书。
- 为客户端证书配置信任存储和信任管理器
- 为双向 SSL/TLS 配置服务器证书
- 配置 SSL 上下文，以使用 SSL/TLS 保护在 JBoss EAP 上部署的应用程序

您可以按照在 Elytron 中配置证书撤销检查中的步骤来配置证书撤销检查。

2.1. 生成客户端证书

在 CLI 中使用 **keytool** 命令生成自签名客户端证书，以测试和开发双向 SSL/TLS 配置。

重要

不要在生产环境中使用自签名证书。仅使用证书颁发机构(CA)签名的证书。

流程

1. 生成客户端证书。

   语法

   ```
   $ 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. 将客户端证书导出到文件。

   语法

   ```
   $ 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>
   ```

现在，您可以使用生成的客户端证书在服务器中配置服务器信任存储和信任管理器。如需更多信息，请参阅为客户端证书配置信任存储和信任管理器。

2.2. 为客户端证书配置信任存储和信任管理器

使用客户端证书和信任管理器配置信任存储，并引用信任存储，以便在 TLS 握手期间验证客户端证书。

先决条件

- 您已获取或生成了客户端证书。
  如需更多信息，请参阅生成客户端证书。
- JBoss EAP 正在运行。

流程

1. 使用管理 CLI 使用客户端证书配置信任存储。

   a. 创建服务器信任存储，以存储要信任的客户端证书。

   语法

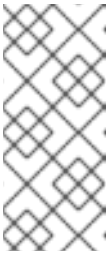
   ```
   /subsystem=elytron/key-store=<server_trust_store_name>:add(path=<path_to_server_trust_store_file>,credential-reference={<password>})
   ```

-

Example

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

b. 通过指定客户端证书别名，将客户端证书导入到服务器信任存储中。只有提供服务器信任存储信任的证书的客户端才能连接到服务器。



注意

如果您使用自签名证书配置双向 SSL/TLS，请将 **validate** 设置为 **false**，因为证书没有信任链。

如果您要使用 CA 签名的证书在生产环境中配置双向 SSL/TLS，请将 **validate** 设置为 **true**。

语法

```
/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. 将客户端证书导出到信任存储文件中。

语法

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

Example

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

2. 配置信任管理器，以在 TLS 握手过程中验证客户端证书。

语法

```
/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"}
```

配置的信任存储中的客户端证书用于验证客户端在与服务器的 TLS 握手期间显示的证书。

其他资源

- 使用证书撤销列表配置证书撤销检查
- 在 Elytron 中使用 OCSP 配置证书撤销检查
- **key-store** 属性
- **trust-manager** 属性

2.3. 为双向 SSL/TLS 配置服务器证书

配置服务器证书，它会在 TLS 握手期间提供给客户端。

先决条件

- JBoss EAP 正在运行。

流程

1. 生成用于测试和开发目的的自签名服务器证书。如果您已从证书颁发机构(CA)获取证书，请跳过这一步。

重要

不要在生产环境中使用自签名证书。仅使用证书颁发机构(CA)签名的证书。

   a. 创建用于存储服务器证书的密钥存储。

语法

```
/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. 在密钥存储中生成服务器证书。

语法

-

```
/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. 将密钥存储存储到文件中。

语法

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

Example

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

d. 导出服务器证书。

语法

```
/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. 创建引用服务器密钥存储的密钥管理器。

语法

```
/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"}
```

当启用 SSL/TLS 时，服务器会向客户端显示此证书。

3. 将服务器证书导入到客户端的信任存储中，以便客户端可以在 SSL 握手期间验证服务器证书。

语法

```
$ 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]:
```

输入 **yes**。您将获得以下输出：

```
Certificate was added to keystore
```

后续步骤

- 要使用双向 SSL/TLS 保护管理接口，请按照以下步骤执行：
- 配置 SSL 上下文以使用 SSL/TLS 保护 JBoss EAP 接口
- 要使用 SSL/TLS 保护部署到 JBoss EAP 的应用程序，请按照以下步骤执行：

配置 SSL 上下文以使用 SSL/TLS 保护在 JBoss EAP 上部署的应用程序

其他资源

- **key-store** 属性
- **key-manager** 属性

2.4. 配置 SSL 上下文以使用 SSL/TLS 保护 JBOSS EAP 管理接口

使用双向 SSL/TLS 保护 JBoss EAP 管理接口，以便只有提供服务器信任的证书的客户端才能连接到服务器的管理界面。

先决条件

- JBoss EAP 正在运行。
- 您已为客户端证书配置了服务器信任存储和信任管理器。
  如需更多信息，请参阅为客户端证书配置信任存储和信任管理器。
- 您已配置了服务器证书。
  如需更多信息，请参阅为 SSL/TLS 配置服务器证书

流程

1. 配置服务器 SSL 上下文以启用双向 SSL。

   语法

   ```
   /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"}
   ```

   默认情况下，SSL 上下文使用 TLSv1.2。您可以将 protocol 属性 **配置为使用** TLSv1.3，如下所示：

   语法

   ```
   /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. 添加对用于 http 管理界面的 SSLContext 的引用。

   语法

```
/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. 定义用于 HTTPS 管理接口套接字的套接字绑定配置。

语法

```
/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. 重新加载服务器。

```
reload


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

输入 **T** 或 **P** 以接受由服务器临时或永久提供的证书。

管理 CLI 断开连接，因为它需要提供客户端证书。

验证

- 验证管理控制台是否受到保护。
  a. 使用 CLI 验证：

语法

```
$ 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. 使用浏览器进行验证。

i. 将客户端证书导入到您的浏览器中。生成客户端证书过程中创建的示例证书 称为 **exampleclient.keystore.pkcs12**，以及用于导入它的示例密码是 **secret**。
有关将证书导入到浏览器的信息，请参阅浏览器的文档。

ii. 在浏览器中打开 **https://localhost:9993**。
浏览器提示您显示与服务器相关的证书。

iii. 选择您导入到浏览器的证书。例如： **exampleclient.keystore.pkcs12**。
如果您使用自签名证书，浏览器会显示一个警告，表示服务器出示的证书未知。

iv. 检查证书，并验证浏览器中显示的指纹是否与密钥存储中的证书的指纹匹配。您可以使用以下命令在密钥存储中查看证书：

语法

```
/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",
```

接受服务器证书后，会提示您输入登录凭证。您可以使用现有 JBoss EAP 用户的用户凭证登录。

- 验证管理 CLI 是否受到保护。
  - 使用以下内容创建文件 **wildfly-config.xml** ：

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>
```

注意

您可以在 **key-store-clear-password** 元素中使用屏蔽的密码来代替明文处理。

- 通过提供客户端证书来访问管理 CLI。

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

两种客户端：客户端的 Web 浏览器和管理 CLI，信任服务器证书，并且服务器信任这两个客户端。客户端和服务器之间的通信通过 SSL/TLS。

其他资源

- **server-ssl-context** 属性

2.5. 配置 SERVER-SSL-CONTEXT 以使用 SSL/TLS 保护在 JBOSS EAP 上部署的应用程序

Elytron 提供名为 **applicationSSC** 的默认 **server-ssl-context**，可用于配置 SSL/TLS。另外，您可以在 Elytron 中创建自己的 SSL 上下文。以下流程演示了使用默认的 SSL 上下文 - **applicationSSC** 来为应用程序配置 SSL/TLS。

先决条件

- JBoss EAP 正在运行。

- 您已为客户端证书配置了服务器信任存储和信任管理器。
  如需更多信息，请参阅为客户端证书配置信任存储和信任管理器。
- 您已配置了服务器证书。
  如需更多信息，请参阅为 SSL/TLS 配置服务器证书

流程

1. 配置默认服务器 SSL 上下文以启用双向 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"
    }
}
```

   默认情况下，SSL 上下文使用 TLSv1.2。您可以将 protocol 属性**配置为使用** TLSv1.3，如下所示：

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

2. 为服务器 SSL 上下文配置信任管理器。

   语法

```
/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. 配置服务器 SSL 上下文的密钥管理器。

   语法

```
/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. 重新加载服务器。

```
reload
```

验证

- 验证您可以访问 JBoss EAP 欢迎页面。
  a. 使用 CLI 验证：

  语法

  ```
  $ 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. 使用浏览器进行验证。

  i. 将客户端证书导入到您的浏览器中。生成客户端证书过程中创建的示例证书 称为 **exampleclient.keystore.pkcs12**，以及用于导入它的示例密码是 **secret**。
  有关将证书导入到浏览器的信息，请参阅浏览器的文档。

  ii. 在浏览器中打开 **https://localhost:8443**。
  浏览器提示您显示与服务器相关的证书。

  iii. 选择您导入到浏览器的证书。例如： **exampleclient.keystore.pkcs12**。
  如果您使用自签名证书，浏览器会显示一个警告，表示服务器出示的证书未知。

  iv. 检查证书，并验证浏览器中显示的指纹是否与密钥存储中的证书的指纹匹配。您可以使用以下命令在密钥存储中查看证书：

  语法

  ```
  /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",
```

接受服务器证书后，您可以访问 JBoss EAP 欢迎页面。

现在为应用程序配置了双向 SSL/TLS。

其他资源

- **server-ssl-context** 属性

第 3 章 在 ELYTRON 中配置证书撤销检查

为确保在 Elytron 或 Elytron 客户端不信任其过期日期之前发出证书颁发机构(CA)撤销的证书，请配置证书撤销检查。您可以使用证书撤销列表(CRL)或在线证书状态协议(OCSP)响应程序进行证书撤销检查。如果您不想下载整个 CRL，请使用 OCSP。

3.1. 使用证书撤销列表配置证书撤销检查

使用 Elytron 信任管理器中的证书撤销列表(CRL)配置证书撤销检查，用于启用双向 SSL/TLS，以便在 Elytron 不信任证书颁发机构(CA)之前通过发出证书颁发机构(CA)撤销的证书。

先决条件

- JBoss EAP 正在运行。
- 配置了信任管理器。
  如需更多信息，请参阅为客户端证书配置信任存储和信任管理器。

流程

1. 使用以下步骤之一将信任管理器配置为使用 CRL：
   - 配置信任管理器，以使用从证书中引用的分布点获取的 CRL。

     语法

     ```
     /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=[])
     ```

   - 覆盖从证书中引用的分布点获取的 CRL。

     语法

     ```
     /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. 配置信任管理器，以使用 CRL 进行证书撤销检查。
   - 如果还为证书撤销检查配置了 OCSP 响应程序，请在信任管理器中添加值为 **true** 的属性 **ocsp.prefer-crls**，以使用 CRL 进行证书撤销检查：

     语法

```
/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")
```

- 如果没有为证书撤销检查配置 OCSP 响应程序，则配置已完成。

其他资源

- **trust-manager** 属性

3.2. 在 ELYTRON 中使用 OCSP 配置证书撤销检查

配置用于启用双向 SSL/TLS 的信任管理器，以使用在线证书状态协议(OCSP)响应程序进行证书撤销检查。OCSP 在 RFC6960 中定义。

当为证书撤销检查配置 OCSP 响应程序和 CRL 时，默认会调用 OCSP 响应程序。

先决条件

- JBoss EAP 正在运行。
- 配置了信任管理器。
  如需更多信息，请参阅为客户端证书配置信任存储和信任管理器。

流程

- 使用以下任一步骤，使用 OCSP 配置信任管理器：
  - 将信任管理器配置为使用证书中定义的 OCSP 响应程序进行证书撤销检查。

    语法

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

    示例

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

  - 覆盖证书中定义的 OCSP 响应程序。

    语法

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

    示例

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

其他资源

- **trust-manager** 属性

3.3. 在 ELYTRON 客户端中使用 CRL 配置证书撤销检查

使用 Elytron 客户端中的证书撤销列表(CRL)配置证书撤销检查，以便在客户端不信任其过期日期前由发出证书颁发机构(CA)撤销的证书。

先决条件

- 您已为 Elytron 客户端创建了 **wildfly-config.xml** 文件。

流程

- 在 **wildfly-config.xml** 文件中的 &lt ;**ssl-context** > 元素中添加以下内容：

  语法

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

  示例

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

其他资源

- **trust-manager** 属性

3.4. 在 ELYTRON 客户端中使用 OCSP 配置证书撤销检查

在 Elytron 客户端中使用在线证书状态协议(OCSP)配置证书撤销检查，以便在客户端不信任其过期日期前由发出证书颁发机构(CA)撤销的证书。当使用 OCSP 响应程序时，您不必下载整个 CRL。

先决条件

- 您已为 Elytron 客户端创建了 **wildfly-config.xml** 文件。

流程

- 在 wildfly-config.xml 中的 < **ssl-context** > 元素中添加以下内容：

  语法

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

示例

```
<ocsp />
```

其他资源

- **trust-manager** 属性

第 4 章 在 JBOSS EAP 客户端中使用 ELYTRON 客户端默认 SSLCONTEXT 安全供应商

要使 Java 虚拟机(JVM)使用 Elytron 客户端配置来提供默认的 **SSLcontext**，您可以使用 **WildFlyElytronClientDefaultSSLContextProvider**。使用此提供程序使客户端库在请求默认的 **SSLContext** 时自动使用 Elytron 客户端配置。

4.1. ELYTRON 客户端默认 SSL 上下文安全供应商

Elytron 客户端提供了一个 Java 安全供应商 **org.wildfly.security.auth.client.WildFlyElytronClientlytronClientDefaultSSLContextProvider**，您可以使用它来注册 Java 虚拟机(JVM)范围的默认 SSL 上下文。

WildFlyElytronClientDefaultSSLContextProvider 提供程序的工作方式如下：

- 当调用 **SSLContext.getDefault （）** 方法时，提供程序会实例化 SSLContext。**SSLContext** 从从以下位置之一获取的身份验证上下文启动：
  - Elytron 客户端配置文件作为参数传递给提供程序。
  - 文件系统中自动发现 **wildfly-config.xml** 文件。如需更多信息，请参阅 默认配置方法。作为参数传递给提供程序的客户端配置文件具有优先权。
- 当调用 **SSLContext.getDefault （）** 方法时，JVM 会返回提供程序实例化的 **SSLContext**。由于 Elytron 客户端可以配置多个 SSL 上下文，因此规则用于为连接选择一个 SSL 上下文。默认 SSL 上下文是匹配所有规则的 SSL 上下文。提供程序返回此默认 SSL 上下文。

注意

如果没有配置默认 **SSLContext**，或者没有配置，则忽略该提供程序。

当您注册 **WildFlyElytronClientDefaultSSLContextProvider** 提供程序时，所有使用 **SSLContext.getDefault （）** 方法的客户端库都使用 Elytron 客户端配置，而无需在代码中使用 Elytron 客户端 API。

要注册供应商，您必须对以下工件添加运行时依赖项：

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

您可以在客户端代码中以编程方式、或以静态方式在 **java.security** 文件中注册提供程序。当您要动态决定使用哪些供应商时，请使用编程注册。

以编程方式注册提供程序

您可以以编程方式在客户端代码中注册供应商，如下所示：

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

以静态方式注册供应商

您可以在 **java.security** 文件中注册供应商，如下所示：

-

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

其他资源

- 创建加载默认 SSLContext 的客户端示例

4.2. 创建加载默认 SSL 上下文的客户端示例

以下示例演示了以编程方式注册 **WildFlyElytronClientDefaultSSLContextProvider** 提供程序，并使用 **SSLContext.getDefault （）** 方法获取 Elytron 客户端初始化的 SSLContext。该示例使用静态客户端配置作为提供程序的参数提供。

4.2.1. 为 JBoss EAP 客户端创建 Maven 项目

要为部署到 JBoss EAP 的应用创建客户端，请创建一个具有所需依赖项和目录结构的 Maven 项目。

先决条件

- 您已安装了 Maven。如需更多信息，请参阅 下载 Apache Maven。

流程

1. 使用 **mvn** 命令设置 Maven 项目。该命令创建项目的目录结构以及 **pom.xml** 配置文件。

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

2. 导航到应用程序根目录。

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

3. 将生成的 **pom.xml** 文件的内容替换为以下文本：

```
<?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. **wildfly-elytron-client** 的依赖项。
2. **wildfly-client-config** 的依赖项。

4. 删除 **src/test** 目录。

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

验证

- 在应用程序根目录中，输入以下命令：

```
$ mvn install
```

您会看到类似如下的输出：

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

后续步骤

- 创建载入默认 SSLContext 的客户端

4.2.2. 创建载入默认 SSLContext的客户端

为部署到 JBoss EAP 的应用创建一个客户端，以使用 **SSLContext.getDefault （）** 方法加载 SSLContext。

在此过程中，*<application_home>* 指向包含应用程序 **pom.xml** 配置文件的目录。

先决条件

- 通过双向 TLS，您已保护部署到 JBoss EAP 的应用。
  要做到这一点，请按照以下步骤执行：
  - 生成客户端证书。
  - 为客户端证书配置信任存储和信任管理器
  - 为双向 SSL/TLS 配置服务器证书

- 配置 SSL 上下文，以使用 SSL/TLS 保护在 JBoss EAP 上部署的应用程序

- 您已创建了一个 Maven 项目。
  如需更多信息，请参阅为 JBoss EAP 客户端创建 Maven 项目 。

- JBoss EAP 正在运行。

流程

1. 创建一个用于存储 Java 文件的目录。

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

2. 前往新目录。

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

3. 使用以下内容创建 Java 文件 **App.java** ：

```
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. 定义 JBoss EAP 主页 URL。
2. 注册安全供应商。**1** 定义此提供程序的优先级.要静态注册提供程序，您可以在 **java.security** 文件中添加提供程序：security.provider **.1=org.wildfly.security.auth.client.WildFlyElytronClientDefaultSSLContextProvider <** ***PATH>*** **/** ***<TO>*****/wildfly-config-two-way-tls.xml**
3. 获取默认的 SSL 上下文。

4. 在 < ***application_home>*** **/src** 目录中创建名为 "wildfly-config-two-way-tls.xml" 的客户端配置文件。

```
<?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>
```

将 holder 值替换为实际路径：

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

验证

1. 进入 < *application_home*> 目录。

2. 运行应用程序。

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

输出示例

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

SSL Default SSLContext is: WildFlyElytronClientDefaultSSLContextProvider
```

第 5 章 参考

5.1. KEY-MANAGER 属性

您可以通过设置其属性来配置 **key-manager**。

表 5.1. key-manager 属性

属性	描述
algorithm	用于创建底层 **KeyManagerFactory** 的算法名称。这是 JDK 提供的。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。如需更多信息，请参阅 Oracle 网站上的支持类和接口。
alias-filter	应用到从密钥存储返回的别名的过滤器。这可以是以逗号分隔的别名列表，可以返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**
credential-reference	用于解密密钥存储项目的凭据引用。这可以以明文形式指定，或作为对存储在 **credential-store** 中的凭据的引用。这不是密钥存储的密码。
generate-self-signed-certificate-host	如果支持密钥存储的文件不存在，并且设置了此属性，则会为指定主机名生成一个自签名证书。不要在生产环境中设置此属性。
key-store	对用于初始化底层 **KeyManageronnectionFactory** 的 **key-store** 的引用。
provider-name	用于创建底层 **KeyManagerFactory** 的供应商名称。
providers	获取在创建底层 **KeyManagerFactory** 时要使用的 **Provider[]** 的引用。

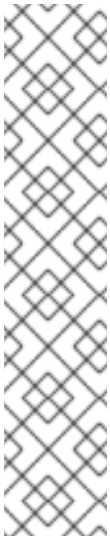
5.2. KEY-STORE 属性

您可以通过设置其属性来配置 **key-store**。

表 5.2. key-store 属性

属性	描述

属性	描述
alias-filter	要应用到从密钥存储返回的别名的过滤器，可以是逗号分隔的别名列表，可以返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**  **注意** **alias-filter** 属性区分大小写。由于使用混合情况或大写字母别名（如 **elytronAppServer** ）可能无法被某些密钥存储供应商识别，因此建议使用小写别名，如 **elytronappserver**。
credential-reference	用于访问密钥存储的密码。这可以以明文形式指定，或作为对存储在 **credential-store** 中的凭据的引用。
path	密钥存储文件的路径。
provider-name	用于加载密钥存储的提供程序名称。设置此属性时，搜索可以创建指定类型的密钥存储的第一个供应商被禁用。
providers	对用于获取要搜索的供应商实例列表的提供程序的引用。如果没有指定，则使用全局供应商列表。
relative-to	这个存储相对于的基本路径。这可以是完整路径或预定义路径，如 **jboss.server.config.dir**。
required	如果设置为 **true**，则引用的密钥存储文件必须在密钥存储服务启动时存在。默认值为 **false**。

属性	描述
type	密钥存储的类型，如 **JKS**。  **注意** 以下密钥存储类型会被自动检测到： • **JKS** • **JCEKS** • **PKCS12** • **BKS** • **BCFKS** • **UBER** 您必须手动指定其他密钥存储类型。 有关密钥存储类型的完整列表，请参阅 Oracle JDK 文档中的 JDK 11 的 Java Cryptography Architecture Standard Algorithm Name 文档。

5.3. SERVER-SSL-CONTEXT 属性

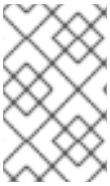
您可以通过设置其属性来配置服务器 SSL 上下文 **server-ssl-context**。

表 5.3. server-ssl-context 属性

属性	描述
authentication-optional	如果为 **true**，则安全域拒绝客户端证书不会阻止连接。当安全域拒绝客户端证书时，这允许使用其他身份验证机制，如表单登录。这仅在设置安全域时才生效。默认值为 **false**。
cipher-suite-filter	用于指定启用的密码套件的过滤器。此过滤器取以冒号、逗号或空格分隔的项目列表。每个项目可以是 OpenSSL 样式的密码套件名称、标准 SSL/TLS 密码套件名称，也可以是关键字，如 **TLSv1.2** 或 **DES**。完整的关键字列表以及创建过滤器的详情，请参考 **CipherSuiteSelector** 类的 Javadoc 中。默认值为 **DEFAULT**，它对应于所有没有 **NULL** 加密并排除任何没有身份验证的密码套件的已知密码套件。
cipher-suite-names	用于为 TLSv1.3 指定启用的密码套件的过滤器。
final-principal-transformer	应用于此机制域的最后一个主体转换器。
key-manager	引用 **SSLContext** 中使用的密钥管理器。

属性	描述
maximum-session-cache-size	要缓存的最大 SSL/TLS 会话数量。
need-client-auth	如果设置为 **true**，则 SSL 握手中需要一个客户端证书。没有可信客户端证书的连接将被拒绝。默认值为 **false**。
post-realm-principal-transformer	选择域之后要应用的主体转换器。
pre-realm-principal-transformer	在选择域之前要应用的主体转换器。
protocols	启用的协议。允许的选项有 • **SSLv2** • **SSLv3** • **TLSv1** • **TLSv1.1** • **TLSv1.2** • **TLSv1.3** 默认为启用 **TLSv1**, **TLSv1.1**, **TLSv1.2**, 和 **TLSv1.3**。 **警告** 使用 TLSv1.2 或 TLSv1.3 而不是 SSLv2、SSLv3 和 TLSv1.0。使用 SSLv2、SSLv3 或 TLSv1.0 会导致安全风险，因此您必须明确禁用它们。 如果没有指定协议，配置 **cipher-suite-names** 将 **协议** 的值设置为 **TLSv1.3**。
provider-name	要使用的供应商的名称。如果未指定，则来自提供程序的所有 **提供程序** 都将传递到 **SSLContext**。
providers	要获取用于加载 **SSLContext** 的 **Provider[]** 的提供程序名称。
realm-mapper	用于 SSL/TLS 身份验证的域映射程序。
security-domain	在 SSL/TLS 会话建立过程中用于身份验证的安全域。

属性	描述
session-timeout	SSL 会话的超时时间（以秒为单位）。 值 **-1** 指示 Elytron 使用 Java 虚拟机(JVM)默认值。 值 **0** 表示存在超时。 默认值为 **-1**。
trust-manager	引用在 SSLContext 中使用的 **trust-manager**。
use-cipher-suites-order	如果设置为 **true**，则使用服务器中定义的密码套件顺序。如果设置为 **false**，则使用客户端提供的密码套件顺序。默认值为 **true**。
want-client-auth	如果设置为 **true**，则需要在 SSL 握手中请求客户端证书，但不需要。如果引用安全域并支持 X509 证据，则 **want-client-auth** 会自动设置为 **true**。当设置了 **need-client-auth** 时会忽略它。默认值为 **false**。
wrap	如果为 **true**，则返回的 **SSLEngine**、**SSLSocket** 和 **SSLServerSocket** 实例会被嵌套，以防止进一步修改。默认值为 **false**。



注意

server-ssl-context 的 **realm-mapper** 和 **principal-transformer** 属性只适用于 SASL EXTERNAL 机制，其中证书由信任管理器验证。HTTP CLIENT-CERT 身份验证设置在 **http-authentication-factory** 中配置。

5.4. TRUST-MANAGER 属性

您可以通过设置其属性来配置信任管理器 **trust-manager**。

表 5.4. trust-manager 属性

属性	描述
algorithm	用于创建底层 **TrustManagerFactory** 的算法名称。这是 JDK 提供的。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。有关 SunJSSE 的详细信息，请参阅 Oracle 文档中的 Java 安全套接字扩展(JSSE)参考指南中的 支持类和接口。

属性	描述
alias-filter	应用到从密钥存储返回的别名的过滤器。这可以是以逗号分隔的别名列表，可以返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**
certificate-revocation-list	在信任管理器中启用证书撤销列表检查。您只能使用此属性定义单个 CRL 路径。要定义多个 CRL 路径，请使用 **certificate-revocation-lists**。**certificate-revocation-list** 的属性是： • **maximum-cert-path** - 认证路径中可存在的最大非签发的中间证书。默认值为 **5**。此属性已弃用。在 **trust-manager** 中使用 **maximum-cert-path**。 • **path** - 证书撤销列表的路径。 • **relative-to** - 证书撤销列表文件的基本路径。
certificate-revocation-lists	使用多个证书撤销列表列表在信任管理器中启用证书撤销列表。**certificate-revocation-list** 的属性是： • **path** - 证书撤销列表的路径。 • **relative-to** - 证书撤销列表文件的基本路径。
key-store	对用于初始化底层 **TrustManagerFactory** 的 **key-store** 的引用。
maximum-cert-path	认证路径中可存在的最大非签发的中间证书。默认值为 **5**。 此属性已从 JBoss EAP 7.3 中的 **trust-manager** 内的 **certificate-revocation-list** 移到 **trust-manager** 中。为了向后兼容，属性也会出现在 **certificate-revocation-list** 中。下一步，使用 **trust-manager** 中的 **maximum-cert-path**。  **注意** 在 **trust-manager** 或 **certificate-revocation-list** 中没有定义两者中的 **maximum-cert-path**。

属性	描述
OCSP	在信任管理器中启用在线证书状态协议(OCSP)检查。**ocsp** 的属性是： • **responder** - 覆盖从证书解析的 OCSP Responder URI。 • **responder-certificate** - 如果未定义 **responder-keystore** 或 **trust-manager** 键存储中的响应器证书的 Alias。 • **responder-keystore** - 响应器证书的备用密钥存储。必须定义 **responder-certificate**。 • **prefer-crls** - 当同时配置了 OCSP 和 CRL 机制时，会首先调用 OCSP 机制。当 **prefer-crls** 设为 **true** 时，首先调用 CRL 机制。
only-leaf-cert	只检查叶证书的撤销状态。这是一个可选属性。默认值为 **false**。
provider-name	用于创建底层 **TrustManagerFactory** 的供应商名称。
providers	获取创建底层 **TrustManagerFactory** 时要使用的提供程序的引用。