Red Hat JBoss Enterprise Application Platform 8.0

JBoss EAP における SSL/TLS の設定

JBoss EAP で SSL/TLS を有効にして JBoss EAP 管理インターフェイスとデプロイされたアプリケーションをセキュアにする方法

Last Updated: 2024-02-09

Red Hat JBoss Enterprise Application Platform 8.0 JBoss EAP における SSL/TLS の設定

JBoss EAP で SSL/TLS を有効にして JBoss EAP 管理インターフェイスとデプロイされたアプリケーションをセキュアにする方法

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

JBoss EAP で SSL/TLS を有効にして JBoss EAP 管理インターフェイスとデプロイされたアプリケーションをセキュアにする方法。

目次

JBOSS EAP ドキュメントへのフィードバック (英語のみ)

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. **このリンクをクリック** してチケットを作成します。
2. **Summary** に課題の簡単な説明を入力します。
3. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
4. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 管理インターフェイスとアプリケーションの一方向 SSL/TLS の有効化

SSL/TLS、または Transport Layer Security (TLS) は、ネットワークを介して通信する 2 つのエンティティー間のデータ転送を保護するために使用される証明書ベースのセキュリティープロトコルです。

JBoss EAP 管理インターフェイスと JBoss EAP にデプロイされたアプリケーションの両方の一方向 SSL/TLS を有効化できます。詳細については、次の手順を参照してください。

- 管理インターフェイスの一方向 SSL/TLS の有効化
- JBoss EAP にデプロイされたアプリケーションの一方向 SSL/TLS の有効化

1.1. 管理インターフェイスの一方向 SSL/TLS の有効化

管理インターフェイスの一方向 SSL/TLS を有効にして、JBoss EAP 管理インターフェイスとインターフェイスに接続するクライアントの通信が保護されるようにします。

以下の手順を使用して、管理インターフェイスの一方向 SSL/TLS を有効化できます。

- ウィザードを使用した管理インターフェイスの一方向 SSL/TLS の有効化: この手順を使用して、CLI ベースのウィザードを使用した SSL/TLS を迅速にセットアップします。Elytron はウィザードへの入力内容に基づいて、必要なリソースを作成します。
- サブシステムコマンドを使用した管理インターフェイスの一方向 SSL/TLS の有効化: この手順を使用して、SSL/TLS を手動で有効化するために必要なリソースを設定します。リソースを手動で設定すると、サーバー設定をより詳細に制御できます。

さらに、security コマンドを使用した管理インターフェイスの SSL/TLS の無効化 の手順を使用して、管理インターフェイスの SSL/TLS を無効化することもできます。

1.1.1. ウィザードを使用した管理インターフェイスの一方向 SSL/TLS の有効化

Elytron は、SSL/TLS を迅速にセットアップするウィザードを提供します。証明書が含まれる既存のキーストアを使用するか、ウィザードが生成するキーストアと自己署名証明書を使用して、SSL/TLS を有効化できます。**--lets-encrypt** オプションを使用して、Let's Encrypt 認証局から証明書を取得して使用することもできます。Let's Encrypt の詳細は、Let's Encrypt のドキュメント を参照してください。

ウィザードが生成する自己署名証明書を使用して、テストおよび開発の目的でのみ SSL/TLS を有効化します。実稼働環境では、常に認証局 (CA) 署名の証明書を使用します。

重要

実稼働環境では自己署名証明書を使用しないでください。認証局 (CA) によって署名された証明書のみ使用してください。

ウィザードは、管理インターフェイスの SSL/TLS を有効化するために必要な以下のリソースを設定します。

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

- **server-ssl-context**
- 次に、**server-ssl-context** が **http-interface** に適用されます。

Elytron は各リソースに **resource-type-UUID** という名前を付けます。たとえば、key-store-9e35a3be-62bb-4fff-afc2-2d8d141b82bc などです。Universally Unique Identifier (UUID) は、リソース名の競合を回避する際に役立ちます。

前提条件

- JBoss EAP が実行されている。

手順

- ウィザードを起動し、管理 CLI で以下のコマンドを入力して管理インターフェイスの一方向 SSL/TLS を設定します。

  構文

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

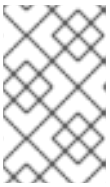
  プロンプトが表示されたら、必要な情報を入力します。

  --lets-encrypt オプションを使用して、Let's Encrypt 認証局から証明書を取得して使用します。

  管理インターフェイスの SSL/TLS がすでに有効化されている場合、ウィザードは終了し、次のメッセージが表示されます。

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

  既存の設定を変更するには、最初に管理インターフェイスの SSL/TLS を無効化してから、新しい設定を作成します。管理インターフェイスの SSL/TLS を無効にする方法の詳細は、ウィザードを使用した管理インターフェイスの SSL/TLS の無効化 を参照してください。

  注記

  一方向 SSL/TLS を有効化するには、SSL 相互認証を有効化するようにプロンプトが表示されたら、**n** を入力するか空白にします。相互認証を設定すると、双方向 SSL/TLS が有効になります。

  ウィザードの対話的な使用例

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

  ウィザードプロンプトへの入力の例

  ```
  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
```

y を入力すると、サーバーはリロードします。自己署名証明書を設定した場合、ウィザードを使用して自己署名証明書を生成した場合、または Java 仮想マシン (JVM) で信頼されない証明書を設定した場合、管理 CLI はサーバーが提示する証明書を受け入れるように要求します。

```
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 :
```

T または **P** を入力して接続を続行します。

次の出力が得られます。

```
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
```

検証

- 管理 CLI クライアントと接続して SSL/TLS を確認します。
  Elytron クライアント SSL コンテキストを設定ファイルに配置し、管理 CLI を使用してサーバーに接続して、設定ファイルを参照することで、SSL/TLS をテストできます。

a. キーストアファイルを含むディレクトリーに移動します。この例では、キーストアファイル **exampleKeystore.pkcs12** がサーバーの **standalone/configuration** ディレクトリーに生成されました。

例

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

b. サーバー証明書を使用して、クライアント **trust-store** を作成します。

構文

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

例

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

自己署名証明書を使用した場合は、証明書を信頼するように求められます。

c. ファイルでクライアント側の SSL コンテキストを定義します (例: **example-security.xml**)。

構文

```
<?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>
```

例

```
<?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. サーバーに接続してコマンドを発行します。

例

```
$ 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
```

予想される出力

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

- ブラウザーを使用して SSL/TLS を確認します。

a. https://localhost:9993 に移動します。
自己署名証明書を使用した場合、サーバーによって提示された証明書が不明であるという警告がブラウザーに表示されます。

b. 証明書を調べて、ブラウザーに表示されるフィンガープリントが、キーストアの証明書のフィンガープリントと一致することを確認します。次のコマンドを使用して、生成した証明書を表示できます。

構文

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

例

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

ウィザードの出力からキーストア名を取得できます (例: "key-store is key-store-a18ba30e-6a26-4ed6-87c5-feb7f3e4dff1")。

出力例

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

サーバー証明書を許可すると、ログイン認証情報の入力を求められます。既存 JBoss EAP ユーザーのユーザー認証情報を使用してログインできます。

これで、SSL/TLS が JBoss EAP 管理インターフェイスに対して有効化されました。

関連情報

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

1.1.2. サブシステムコマンドを使用した管理インターフェイスの一方向 SSL/TLS の有効化

elytron サブシステムコマンドを使用して、JBoss EAP 管理インターフェイスを SSL/TLS で保護します。

テストおよび開発の目的で、自己署名証明書を使用できます。証明書が含まれる既存のキーストアを使用するか、**key-store** リソースの作成時に Elytron が生成するキーストアを使用できます。実稼働環境では、常に認証局 (CA) 署名の証明書を使用します。

重要

実稼働環境では自己署名証明書を使用しないでください。認証局 (CA) によって署名された証明書のみ使用してください。

前提条件

- JBoss EAP が実行されている。

手順

1. 証明書を格納するようにキーストアを設定します。
   たとえば、CA 署名の証明書が含まれるキーストアなどの既存のキーストアへのパスを指定するか、作成するキーストアへのパスを指定できます。

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

例

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

2. キーストアに証明書が含まれていない場合や、上記の手順を使用してキーストアを作成した場合は、証明書を生成し、証明書をファイルに保存する必要があります。

   a. キーストアでキーペアを生成します。

      構文

      ```
      /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>")
      ```

      例

      ```
      /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. 証明書をファイルに保存します。

      構文

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

      例

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

3. **key-store** を参照する **key-manager** を設定します。

   構文

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

   例

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

重要

elytron サブシステムはデフォルトで **KeyManagerFactory.getDefaultAlgorithm()** を使用してアルゴリズムを決定するため、Red Hat はアルゴリズム属性を指定しませんでした。ただし、アルゴリズム属性は指定できます。

アルゴリズム属性を指定するには、使用している Java Development Kit (JDK) によって提供されるキーマネージャーアルゴリズムを知る必要があります。たとえば、Java Secure Socket Extension (SunJSSE) を使用する JDK は、PKIX および SunX509 アルゴリズムを提供します。

このコマンドでは、SunX509 を **key-manager** アルゴリズム属性として指定できます。

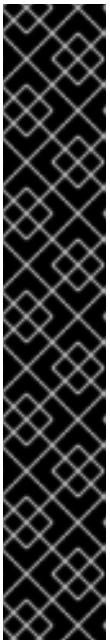
4. **key-manager** を参照する **server-ssl-context** を設定します。

構文

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

例

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



重要

利用できるようにする必要のある SSL/TLS プロトコルを決定する必要があります。コマンド例は TLSv1.2 を使用します。

- TLSv1.2 以前の場合は、**cipher-suite-filter** 引数を使用して、許可される暗号スイートを指定します。
- TLSv1.3 の場合、**cipher-suite-names** 引数を使用して、許可される暗号スイートを指定します。TLSv1.3 は、デフォルトでは無効になっています。**protocols** 属性でプロトコルを指定しない場合、または指定されたセットに TLSv1.3 が含まれている場合、**cipher-suite-names** を設定すると TLSv1.3 が有効になります。

use-cipher-suites-order 引数を使用して、サーバーの暗号スイートの順序に従います。**use-cipher-suites-order** 属性は、デフォルトで **true** に設定されています。これは、デフォルトでクライアント暗号スイートの順序に従うレガシーセキュリティーサブシステムの動作とは異なります。

5. 設定した **server-ssl-context** を使用するように管理インターフェイスを更新します。

構文

```
/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)
```

例

```
/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. サーバーをリロードします。

```
reload
```

SSL/TLS を有効化するために自己署名証明書を使用した場合、管理 CLI はサーバーが提示する証明書を受け入れるように要求します。これは、キーストアを設定した証明書です。

出力例

```
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 :
```

T または **P** を入力して接続を続行します。

検証

- クライアントを介して接続して SSL/TLS を確認します。
設定ファイル内に Elytron クライアント SSL コンテキストを配置し、設定ファイルを参照する管理 CLI を使用してサーバーに接続すると、SSL/TLS をテストできます。

  a. キーストアファイルを含むディレクトリーに移動します。この例では、キーストアファイル **exampleserver.keystore.pkcs12** がサーバーの **standalone/configuration** ディレクトリーに生成されました。

  例

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

  b. サーバー証明書をエクスポートして、クライアントトラストストアにインポートできるようにします。

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

  例

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

c. サーバー証明書を使用して、クライアント **trust-store** を作成します。

構文

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

例

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

自己署名証明書を使用した場合は、証明書を信頼するように求められます。

d. ファイルでクライアント側の SSL コンテキストを定義します (例: **example-security.xml**)。

構文

```
<?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>
```

例

```
<?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. サーバーに接続してコマンドを発行します。

例

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

予想される出力

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

- ブラウザーを使用して SSL/TLS を確認します。

a. https://localhost:9993 に移動します。
自己署名証明書を使用した場合、サーバーによって提示された証明書が不明であるという警告がブラウザーに表示されます。

b. 証明書を調べて、ブラウザーに表示されるフィンガープリントが、キーストアの証明書のフィンガープリントと一致することを確認します。次のコマンドを使用して、生成した証明書を表示できます。

構文

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

例

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

出力例

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

サーバー証明書を許可すると、ログイン認証情報の入力を求められます。既存 JBoss EAP ユーザーのユーザー認証情報を使用してログインできます。

これで、SSL/TLS が JBoss EAP 管理インターフェイスに対して有効化されました。

関連情報

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

1.1.3. security コマンドを使用した管理インターフェイスの SSL/TLS の無効化

security コマンドを使用して、管理インターフェイスの SSL/TLS を無効にします。これを実行して、設定されているものとは異なる SSL/TLS 設定を使用することを推奨します。

コマンドを使用して SSL/TLS を無効にしても、Elytron リソースは削除されません。このコマンドは単に、**http-interface management-interface** リソースの **secure-socket-binding** および **ssl-context** 属性の定義を解除します。

前提条件

- JBoss EAP が実行されている。

手順

- 管理 CLI で **disable-ssl-management** コマンドを使用します。

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

サーバーがリロードされ、以下の出力が表示されます。

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

以下のいずれかの方法で、サーバー管理インターフェイスに対して SSL/TLS を有効化できます。

- ウィザードを使用した管理インターフェイスの一方向 SSL/TLS の有効化: この手順を使用して、CLI ベースのウィザードを使用した SSL/TLS を迅速にセットアップします。Elytron はウィザードへの入力内容に基づいて、必要なリソースを作成します。
- サブシステムコマンドを使用した管理インターフェイスの一方向 SSL/TLS の有効化: この手順を使用して、SSL/TLS を手動で有効化するために必要なリソースを設定します。リソースを手動で設定すると、サーバー設定をより詳細に制御できます。

1.2. JBOSS EAP にデプロイされたアプリケーションの一方向 SSL/TLS の有効化

JBoss EAP にデプロイされたアプリケーションの一方向 SSL/TLS を有効化して、Web ブラウザーなどのアプリケーションとクライアント間の通信が保護されるようにします。

以下の手順を使用して、JBoss EAP にデプロイされたアプリケーションの一方向 SSL/TLS を有効化できます。

- 自動生成された自己署名証明書を使用したアプリケーションの SSL/TLS の有効化: この手順は、開発環境またはテスト環境でのみ使用してください。この手順は、設定を行わずにアプリケーションの SSL/TLS を迅速に有効化する上で役立ちます。
- ウィザードを使用して JBoss EAP にデプロイされたアプリケーションの一方向 SSL/TLS の有効化: この手順を使用し、CLI ベースのウィザードを使用して SSL/TLS を迅速にセットアップします。Elytron はウィザードへの入力内容に基づいて、必要なリソースを作成します。
- サブシステムコマンドを使用したアプリケーションの一方向 SSL/TLS の有効化: この方法を使用して、SSL/TLS を手動で有効化するために必要なリソースを設定します。リソースを手動で設定すると、サーバー設定をより詳細に制御できます。

さらに、security コマンドを使用したアプリケーションの SSL/TLS の無効化 の手順を使用して、JBoss EAP にデプロイされたアプリケーションの SSL/TLS を無効化できます。

1.2.1. Elytron のデフォルトの SSL コンテキスト

開発者がアプリケーションの一方向 SSL/TLS を迅速にセットアップできるようにするために、**elytron** サブシステムには一方向 SSL/TLS を有効化するために必要なリソースが含まれ、デフォルトで開発環境またはテスト環境ですぐに使用できます。

デフォルトで以下のリソースが提供されます。

- **applicationKS** という名前の **key-store**。
- **key-store** を参照する **applicationKM** という名前の **key-manager**。
- **key-manager** を参照する **applicationSSC** という名前の **server-ssl-context**。

デフォルトの TLS 設定

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

デフォルトの **key-manager**、**applicationKM** には、**localhost** の値を持つ **generate-self-signed-certificate-host** 属性が含まれます。**generate-self-signed-certificate-host** 属性は、この **key-manager** を使用してサーバーの証明書を取得する場合、その **key-store** をサポートするファイルがま

だ存在しない場合は、key-manager が **localhost** を **Common Name** として自己署名証明書を自動的に生成することを示します。この生成された自己署名証明書は、**key-store** をサポートするファイルに保存されます。

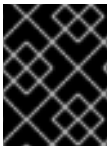
サーバーのインストール時に、デフォルトの key-store をサポートするファイルが存在しないため、サーバーに https 要求を送信するだけで自己署名証明書が生成され、アプリケーションの一方向 SSL/TLS が有効になります。詳細は、自動生成された自己署名証明書を使用したアプリケーションの SSL/TLS の有効化 を参照してください。

関連情報

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

1.2.2. 自動生成された自己署名証明書を使用したアプリケーションの SSL/TLS の有効化

JBoss EAP は、サーバーが HTTPS 要求を初めて受信したときに自己署名証明書を自動的に生成します。**elytron** サブシステムには、デフォルトで開発環境またはテスト環境ですぐに使用できる **key-store**、**key-manager**、および **server-ssl-context** リソースも含まれます。そのため、JBoss EAP が自己署名証明書を生成するとすぐに、アプリケーションはその証明書を使用して保護されます。



重要

実稼働環境では自己署名証明書を使用しないでください。認証局 (CA) によって署名された証明書のみ使用してください。

前提条件

- JBoss EAP が実行されている。

手順

- ポート **8443** のサーバー URL に移動します (例: https://localhost:8443)。
  JBoss EAP はこのリクエストを受信すると、自己署名証明書を生成します。この証明書の詳細は、サーバーログで確認できます。

  生成された証明書は自己署名されているため、ブラウザーは接続が安全ではないとしてフラグを立てます。

検証

1. JBoss EAP がブラウザーに提示した証明書とサーバーログの証明書を比較します。

   サーバーログの例

   ```
   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
```

ブラウザーに提示された証明書の例

```
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. この例のようにフィンガープリントが一致する場合は、そのページに進むことができます。

SSL/TLS がアプリケーション用に有効化されています。

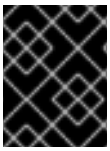
関連情報

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

1.2.3. ウィザードを使用した JBoss EAP にデプロイされたアプリケーションの一方向 SSL/TLS の有効化

Elytron は、SSL/TLS を迅速にセットアップするウィザードを提供します。証明書が含まれる既存のキーストアを使用するか、ウィザードが生成するキーストアと自己署名証明書を使用して、SSL/TLS を有効化できます。**--lets-encrypt** オプションを使用して、Let's Encrypt 認証局から証明書を取得して使用することもできます。Let's Encrypt の詳細は、Let's Encrypt のドキュメント を参照してください。

ウィザードが生成する自己署名証明書を使用して、テストおよび開発の目的でのみ SSL/TLS を有効化します。実稼働環境では、常に認証局 (CA) 署名の証明書を使用します。



重要

実稼働環境では自己署名証明書を使用しないでください。認証局 (CA) によって署名された証明書のみ使用してください。

ウィザードは、アプリケーションの SSL/TLS を有効にするために必要な以下のリソースを設定します。

- **key-store**
- **key-manager**
- **server-ssl-context**
- 次に、**server-ssl-context** は Undertow **https-listener** に適用されます。

Elytron は各リソースに **resource-type-UUID** という名前を付けます。たとえば、key-store-9e35a3be-62bb-4fff-afc2-2d8d141b82bc などです。Universally Unique Identifier (UUID) は、リソース名の競合を回避する際に役立ちます。

前提条件

- JBoss EAP が実行されている。

手順

- 管理 CLI で以下のコマンドを入力して、ウィザードを起動し、アプリケーションの一方向 SSL/TLS を設定します。

  構文

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

  プロンプトが表示されたら、必要な情報を入力します。

  --lets-encrypt オプションを使用して、Let's Encrypt 認証局から証明書を取得して使用します。

  server-ssl-context がすでに存在する場合、ウィザードは以下のメッセージで終了します。

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

  注記

  elytron サブシステムには、デフォルトで設定済みの **server-ssl-context** リソースが含まれています。したがって、新規インストール後にウィザードを初めて起動するときは、**--override-ssl-context** オプションを使用する必要があります。

  詳細は、Elytron のデフォルトの SSL コンテキスト を参照してください。

  既存の **server-ssl-context** をオーバーライドすると、Elytron はウィザードによって作成された **server-ssl-context** を使用して SSL を有効化します。

  注記

  一方向 SSL/TLS を有効化するには、SSL 相互認証を有効化するようにプロンプトが表示されたら、**n** を入力するか空白にします。相互認証を設定すると、双方向 SSL/TLS が有効になります。

  ウィザードの起動例

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

  ウィザードプロンプトへの入力の例

  ```
  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
```

y を入力すると、サーバーは以下の出力で再読み込みされます。

```
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
```

検証

1. https://localhost:8443 に移動します。
   自己署名証明書を使用した場合、サーバーによって提示された証明書が不明であるという警告がブラウザーに表示されます。

2. 証明書を調べて、ブラウザーに表示されるフィンガープリントが、キーストアの証明書のフィンガープリントと一致することを確認します。次のコマンドを使用して、生成した証明書を表示できます。

   構文

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

   例

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

ウィザードの出力からキーストア名を取得できます (例: "key-store is key-store-4cba6678-c464-4dcc-90ff-9295312ac395")。

出力例

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

JBoss EAP にデプロイされたアプリケーションの SSL/TLS が有効になりました。

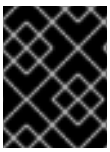
関連情報

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

1.2.4. サブシステムコマンドを使用したアプリケーションの一方向 SSL/TLS の有効化

elytron サブシステムコマンドを使用して、JBoss EAP にデプロイされたアプリケーションを SSL/TLS で保護します。

テストおよび開発の目的で、自己署名証明書を使用できます。証明書が含まれる既存のキーストアを使用するか、**key-store** リソースの作成時に Elytron が生成するキーストアを使用できます。実稼働環境では、常に認証局 (CA) 署名の証明書を使用します。



重要

実稼働環境では自己署名証明書を使用しないでください。認証局 (CA) によって署名された証明書のみ使用してください。

前提条件

- JBoss EAP が実行されている。

手順

1. 証明書を格納するようにキーストアを設定します。
   たとえば、CA 署名の証明書が含まれるキーストアなどの既存のキーストアへのパスを指定するか、作成するキーストアへのパスを指定できます。

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

例

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

2. キーストアに証明書が含まれていない場合や、上記の手順を使用してキーストアを作成した場合は、証明書を生成し、証明書をファイルに保存する必要があります。

   a. キーストアでキーペアを生成します。

      構文

      ```
      /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>")
      ```

      例

      ```
      /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. 証明書をファイルに保存します。

      構文

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

      例

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

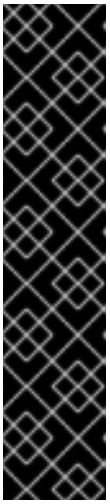
3. **key-store** を参照する **key-manager** を設定します。

   構文

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

   例

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



重要

elytron サブシステムはデフォルトで **KeyManagerFactory.getDefaultAlgorithm()** を使用してアルゴリズムを決定するため、Red Hat はアルゴリズム属性を指定しませんでした。ただし、アルゴリズム属性は指定できます。

アルゴリズム属性を指定するには、使用している Java Development Kit (JDK) によって提供されるキーマネージャーアルゴリズムを知る必要があります。たとえば、Java Secure Socket Extension (SunJSSE) を使用する JDK は、PKIX および SunX509 アルゴリズムを提供します。

このコマンドでは、SunX509 を **key-manager** アルゴリズム属性として指定できます。

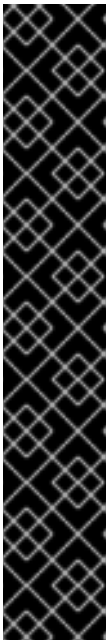
4. **key-manager** を参照する **server-ssl-context** を設定します。

構文

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

例

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



重要

利用できるようにする必要のある SSL/TLS プロトコルを決定する必要があります。コマンド例は TLSv1.2 を使用します。

- TLSv1.2 以前の場合は、**cipher-suite-filter** 引数を使用して、許可される暗号スイートを指定します。
- TLSv1.3 の場合、**cipher-suite-names** 引数を使用して、許可される暗号スイートを指定します。TLSv1.3 は、デフォルトでは無効になっています。**protocols** 属性でプロトコルを指定しない場合、または指定されたセットに TLSv1.3 が含まれている場合、**cipher-suite-names** を設定すると TLSv1.3 が有効になります。

use-cipher-suites-order 引数を使用して、サーバーの暗号スイートの順序に従います。**use-cipher-suites-order** 属性は、デフォルトで **true** に設定されています。これは、デフォルトでクライアント暗号スイートの順序に従うレガシーセキュリティーサブシステムの動作とは異なります。

5. 設定された **server-ssl-context** を使用するように Undertow を更新します。

構文

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

例

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

6. サーバーをリロードします。

```
reload
```

検証

1. https://localhost:8443 に移動します。
自己署名証明書を使用した場合、サーバーによって提示された証明書が不明であるという警告がブラウザーに表示されます。

2. 証明書を調べて、ブラウザーに表示されるフィンガープリントが、キーストアの証明書のフィンガープリントと一致することを確認します。次のコマンドを使用して、生成した証明書を表示できます。

構文

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

例

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

出力例

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

JBoss EAP にデプロイされたアプリケーションの SSL/TLS が有効になりました。

関連情報

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

1.2.5. security コマンドを使用したアプリケーションの SSL/TLS の無効化

security コマンドを使用して、JBoss EAP にデプロイされたアプリケーションの SSL/TLS を無効化します。コマンドを使用して SSL/TLS を無効にしても、Elytron リソースは削除されません。このコマンドは、サーバーの **ssl-context** をデフォルト値の **applicationSSC** に設定します。

前提条件

前提条件

- JBoss EAP が実行されている。

手順

- 管理 CLI で **security disable-ssl-http-server** コマンドを使用します。

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

サーバーがリロードされ、以下の出力が表示されます。

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

以下の手順の 1 つを使用すると、JBoss EAP にデプロイされたアプリケーションの SSL/TLS を有効化できます。

- 自動生成された自己署名証明書を使用したアプリケーションの SSL/TLS の有効化: この手順は、開発環境またはテスト環境でのみ使用してください。この手順は、設定を行わずにアプリケーションの SSL/TLS を迅速に有効化する上で役立ちます。
- ウィザードを使用して JBoss EAP にデプロイされたアプリケーションの一方向 SSL/TLS の有効化: この手順を使用し、CLI ベースのウィザードを使用して SSL/TLS を迅速にセットアップします。Elytron はウィザードへの入力内容に基づいて、必要なリソースを作成します。
- サブシステムコマンドを使用したアプリケーションの一方向 SSL/TLS の有効化: この方法を使用して、SSL/TLS を手動で有効化するために必要なリソースを設定します。リソースを手動で設定すると、サーバー設定をより詳細に制御できます。

関連情報

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

第2章 管理インターフェイスとアプリケーションの双方向 SSL/TLS の有効化

SSL/TLS、または Transport Layer Security (TLS) は、ネットワークを介して通信する 2 つのエンティティー間のデータ転送を保護するために使用される証明書ベースのセキュリティープロトコルです。サーバーが信頼できるクライアントとのみ接続するようにする場合は、双方向 SSL/TLS を使用します。

双方向 SSL/TLS は、次のセキュリティー機能を提供します。

認証

一方向 SSL/TLS の場合、サーバーは証明書をクライアントに提示してサーバー自体を認証します。双方向 SSL/TLS の場合、クライアントもサーバーに証明書を提示し、サーバーがクライアントを認証します。したがって、双方向 SSL/TLS は相互認証とも呼ばれます。

機密性

クライアントとサーバー間で転送されるデータは暗号化されます。

データの整合性

TLS プロトコルは、メッセージ認証コード (MAC) の計算に使用されるセキュアなハッシュ関数でデータインテグリティーを提供します。SSL コンテキストリソースの **cipher-suite-filter** 属性と **cipher-suite-names** 属性を使用して、接続に特定のアルゴリズムとハッシュ関数を適用できます。

詳細については、**server-ssl-context** 属性 を参照してください。

双方向 SSL/TLS を使用して、JBoss EAP 管理インターフェイスとデプロイされたアプリケーションの両方を保護できます。

双方向 SSL/TLS で管理インターフェイスを保護するには、次の手順を使用します。

- クライアントの認証局 (CA) から証明書を取得します。または、非実稼働環境では、クライアント証明書の生成 の手順に従って、自己署名証明書を生成できます。
- クライアント証明書のトラストストアとトラストマネージャーを設定します。
- 双方向 SSL/TLS のサーバー証明書を設定します。
- SSL/TLS で JBoss EAP 管理インターフェイスをセキュアにするための SSL コンテキストを設定します。

双方向 SSL/TLS を使用して JBoss EAP にデプロイされたアプリケーションを保護するには、次の手順を使用します。

- クライアントの認証局 (CA) から証明書を取得します。または、非実稼働環境では、クライアント証明書の生成 の手順に従って、自己署名証明書を生成できます。
- クライアント証明書のトラストストアとトラストマネージャーを設定します。
- 双方向 SSL/TLS 用サーバー証明書の設定
- SSL/TLS を使用して JBoss EAP にデプロイされたアプリケーションを保護するための SSL コンテキストを設定します。

Elytron における証明書失効チェックの設定 の手順に従って、証明書失効チェックを設定できます。

2.1. クライアント証明書の生成

双方向 SSL/TLS 設定のテストと開発を目的として、CLI で **keytool** コマンドを使用し、自己署名クライアント証明書を生成します。

重要

実稼働環境では自己署名証明書を使用しないでください。認証局 (CA) によって署名された証明書のみ使用してください。

手順

1. クライアント証明書を生成します。

   構文

   ```
   $ 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>
   ```

   例

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

2. クライアント証明書をファイルにエクスポートします。

   構文

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

   例

   ```
   $ 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>
   ```

生成されたクライアント証明書を使用して、サーバーのトラストストアとトラストマネージャーをサーバーに設定できるようになりました。詳細については、クライアント証明書用のトラストストアとトラストマネージャーの設定 を参照してください。

2.2. クライアント証明書のトラストストアおよびトラストマネージャーの設定

クライアント証明書を使用してトラストストアを設定し、そのトラストストアへの参照を使用してトラストマネージャーを設定して、TLS ハンドシェイク中にクライアント証明書を検証します。

前提条件

- クライアント証明書を取得または生成している。
  詳細は、クライアント証明書の生成 を参照してください。

- JBoss EAP が実行されている。

手順

1. 管理 CLI を使用して、クライアント証明書でトラストストアを設定します。
   a. 信頼するクライアント証明書を格納するためのサーバートラストストアを作成します。

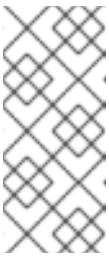
      構文

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

      例

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

   b. クライアント証明書のエイリアスを指定して、クライアント証明書をサーバートラストストアにインポートします。サーバーのトラストストアが信頼する証明書を提示するクライアントのみがサーバーに接続できます。

      注記

      自己署名証明書を使用して双方向 SSL/TLS を設定している場合、証明書のトラストチェーンが存在しないため、**validate** を **false** に設定します。

      CA によって署名された証明書を使用して実稼働環境で双方向 SSL/TLS を設定する場合は、**validate** を **true** に設定します。

      構文

      ```
      /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>)
      ```

      例

      ```
      /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. クライアント証明書をトラストストアファイルにエクスポートします。

      構文

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

例

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

2. TLS ハンドシェーク中にクライアント証明書を検証するようにトラストマネージャーを設定します。

構文

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

例

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

設定されたトラストストア内のクライアント証明書は、サーバーとの TLS ハンドシェイク中にクライアントが提示する証明書を検証するために使用されます。

関連情報

- 証明書失効リストを使用した証明書失効チェックの設定
- Elytron における OCSP を使用した証明書失効チェックの設定
- **key-store** 属性
- **trust-manager** 属性

2.3. 双方向 SSL/TLS 用サーバー証明書の設定

TLS ハンドシェイク中にクライアントに提示されるサーバー証明書を設定します。

前提条件

- JBoss EAP が実行されている。

手順

1. テストおよび開発目的で使用する自己署名サーバー証明書を生成します。認証局 (CA) から証明書を取得している場合は、この手順を省略してください。

重要

実稼働環境では自己署名証明書を使用しないでください。認証局 (CA) によって署名された証明書のみ使用してください。

a. サーバー証明書を格納するキーストアを作成します。

構文

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

例

```
/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. キーストアでサーバー証明書を生成します。

構文

```
/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>")
```

例

```
/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. キーストアをファイルに格納します。

構文

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

例

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

d. サーバー証明書をエクスポートします。

構文

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

例

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

2. サーバーキーストアを参照するキーマネージャーを作成します。

構文

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

例

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

SSL/TLS が有効な場合、サーバーはこの証明書をクライアントに提示します。

3. サーバー証明書をクライアントのトラストストアにインポートして、クライアントが SSL ハンドシェーク中にサーバー証明書を検証できるようにします。

構文

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

例

```
$ 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]:
```

yes を入力します。次の出力が得られます。

```
Certificate was added to keystore
```

次のステップ

- 双方向 SSL/TLS で管理インターフェイスを保護にするには、こののの手順に従います。
- SSL/TLS で JBoss EAP インターフェイスを保護するための SSL コンテキストの設定
- SSL/TLS を使用して JBoss EAP にデプロイされたアプリケーションを保護するには、以下の手順に従います。
  SL/TLS を使用して JBoss EAP にデプロイされたアプリケーションを保護するための SSL コンテキストを設定します。

関連情報

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

2.4. SSL/TLS を使用して JBOSS EAP 管理インターフェイスをセキュリティー保護するための SSL コンテキスト設定

サーバーが信頼する証明書を提示するクライアントのみがサーバーの管理インターフェイスに接続できるように、JBoss EAP 管理インターフェイスを双方向 SSL/TLS でセキュリティー保護します。

前提条件

- JBoss EAP が実行されている。
- クライアント証明書用のサーバートラストストアとトラストマネージャーを設定している。
  詳細については、クライアント証明書用のトラストストアとトラストマネージャーの設定 を参照してください。
- サーバー証明書を設定している。
  詳細は、SSL/TLS 用サーバー証明書の設定 を参照してください。

手順

1. サーバー SSL コンテキストを設定して、双方向 SSL を有効にします。

   構文

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

例

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

デフォルトでは、SSL コンテキストは TLSv1.2 を使用します。次に示すとおり、TLSv1.3 を使用するように **protocol** 属性を設定できます。

構文

```
/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. http 管理インターフェイスに使用する SSLContext への参照を追加します。

構文

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

例

```
/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. HTTPS 管理インターフェイスのソケットに使用するソケットバインディング設定を定義します。

構文

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

例

```
/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. サーバーをリロードします。

```
reload


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

T または **P** を入力して、サーバーから提供された証明書を一時的または永続的に許可します。

クライアント証明書が提示されることを想定しているため、管理 CLI は切断します。

検証

- 管理コンソールが保護されていることを確認します。
  a. CLI を使用して確認します。

  構文

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

  例

  ```
  $ 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. ブラウザーを使用して確認します。

  i. クライアント証明書をブラウザーにインポートします。クライアント証明書の生成 手順で作成されたサンプル証明書は **exampleclient.keystore.pkcs12** と呼ばれ、それをインポートするためのサンプルパスワードは **secret** です。
  ブラウザーへの証明書のインポートについては、ブラウザーのドキュメントを参照してください。

  ii. ブラウザーで **https://localhost:9993** にアクセスします。
  ブラウザーは、サーバーで識別するための証明書を提示するように求めます。

  iii. ブラウザーにインポートした証明書を選択します。たとえば、**exampleclient.keystore.pkcs12** です。
  自己署名証明書を使用する場合、サーバーによって提示された証明書が不明であるという警告がブラウザーに表示されます。

iv. 証明書を調べて、ブラウザーに表示されるフィンガープリントが、キーストアの証明書のフィンガープリントと一致することを確認します。次のコマンドを使用して、キーストアの証明書を表示できます。

構文

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

例

```
/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",
```

サーバー証明書を許可すると、ログイン認証情報の入力を求められます。既存 JBoss EAP ユーザーのユーザー認証情報を使用してログインできます。

- 管理 CLI が保護されていることを確認します。
  - 次の内容でファイル **wildfly-config.xml** を作成します。

例

```
<?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>
```

注記

key-store-clear-password 要素で、クリアテキストの代わりにマスクされたパスワードを使用して難読化できます。

- クライアント証明書を提示して、管理 CLI にアクセスします。

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

両方のクライアント (クライアントの Web ブラウザーと管理 CLI) はサーバーの証明書を信頼し、サーバーは両方のクライアントを信頼します。クライアントとサーバー間の通信は SSL/TLS を介して行われます。

関連情報

- **server-ssl-context** 属性

2.5. JBOSS EAP にデプロイされたアプリケーションを SSL/TLS で保護するための SERVER-SSL-CONTEXT 設定

Elytron は、SSL/TLS の設定に使用できる **applicationSSC** と呼ばれるデフォルトの **server-ssl-context** を提供します。または、Elytron で独自の SSL コンテキストを作成できます。次の手順では、デフォルトの SSL コンテキストである **applicationSSC** を 使用して、アプリケーションの SSL/TLS を設定する方法を説明します。

前提条件

- JBoss EAP が実行されている。
- クライアント証明書用のサーバートラストストアとトラストマネージャーを設定している。
  詳細については、クライアント証明書用のトラストストアとトラストマネージャーの設定 を参照してください。
- サーバー証明書を設定している。
  詳細は、SSL/TLS 用サーバー証明書の設定 を参照してください。

手順

1. デフォルトのサーバー SSL コンテキストを設定して、双方向 SSL を有効にします。

```
/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"
    }
}
```

デフォルトでは、SSL コンテキストは TLSv1.2 を使用します。次に示すとおり、TLSv1.3 を使用するように **protocol** 属性を設定できます。

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

2. サーバー SSL コンテキストのトラストマネージャーを設定します。

構文

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

例

```
/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. サーバー SSL コンテキストの鍵マネージャーを設定します。

構文

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

例

```
/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. サーバーをリロードします。

```
reload
```

検証

検証

- JBoss EAP のウェルカムページにアクセスできることを確認します。
  a. CLI を使用して確認します。

     構文

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

     例

     ```
     $ 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. ブラウザーを使用して確認します。

     i. クライアント証明書をブラウザーにインポートします。クライアント証明書の生成 手順で作成されたサンプル証明書は **exampleclient.keystore.pkcs12** と呼ばれ、それをインポートするためのサンプルパスワードは **secret** です。
     ブラウザーへの証明書のインポートについては、ブラウザーのドキュメントを参照してください。

     ii. ブラウザーで **https://localhost:8443** に移動します。
     ブラウザーは、サーバーで識別するための証明書を提示するように求めます。

     iii. ブラウザーにインポートした証明書を選択します。たとえば、**exampleclient.keystore.pkcs12** です。
     自己署名証明書を使用する場合、サーバーによって提示された証明書が不明であるという警告がブラウザーに表示されます。

     iv. 証明書を調べて、ブラウザーに表示されるフィンガープリントが、キーストアの証明書のフィンガープリントと一致することを確認します。次のコマンドを使用して、キーストアの証明書を表示できます。

     構文

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

     例

     ```
     /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",
     ```

  サーバー証明書を許可すると、JBoss EAP のウェルカムページにアクセスできます。

これでアプリケーション用の双方向 SSL/TLS が設定されました。

関連情報

- **server-ssl-context** 属性

第3章 ELYTRON における証明書失効チェックの設定

有効期限が切れる前に発行元の認証局 (CA) によって失効された証明書を Elytron または Elytron クライアントが信頼しないようにするには、証明書失効チェックを設定します。証明書失効チェックには、Certificate Revocation Lists (CRL) または Online Certificate Status Protocol (OCSP) レスポンダーのいずれかを使用できます。CRL 全体をダウンロードしたくない場合は、OCSP を使用します。

3.1. 証明書失効リストを使用した証明書失効チェックの設定

双方向 SSL/TLS を有効にするために使用される Elytron トラストマネージャーで証明書失効リスト (CRL) を使用して証明書失効チェックを設定し、有効期限前に発行元の認証局 (CA) によって失効された証明書を Elytron が信頼しないようにします。

前提条件

- JBoss EAP が実行されている。
- トラストマネージャーが設定されている。
  詳細については、クライアント証明書用のトラストストアとトラストマネージャーの設定 を参照してください。

手順

1. 次のいずれかの手順を使用して、CRL を使用するようにトラストマネージャーを設定します。
   - トラストマネージャーが、証明書で参照されるディストリビューションポイントから取得した CRL を使用するように設定します。

     構文

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

     例

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

   - 証明書で参照されているディストリビューションポイントから取得した CRL を上書きします。

     構文

     ```
     /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>"}])
     ```

     例

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

2. 証明書失効チェックに CRL を使用するようにトラストマネージャーを設定します。

- OCSP レスポンダーも証明書失効チェック用に設定されている場合は、トラストマネージャーで **ocsp.prefer-crls** 属性の値 **true** を追加し、証明書失効チェックに CRL を使用します。

  構文

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

  例

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

- OCSP レスポンダーが証明書失効チェック用に設定されていない場合、設定は完了です。

関連情報

- **trust-manager** 属性

3.2. ELYTRON における OCSP を使用した証明書失効チェックの設定

双方向 SSL/TLS が証明書失効リストに Online Certificate Status Protocol (OCSP) レスポンダーを使用するために使用されるトラストマネージャーを設定します。OCSP は RFC6960 で定義されてい ます。

OCSP レスポンダーと CRL の両方が証明書失効チェック用に設定されている場合、デフォルトで OCSP レスポンダーが呼び出されます。

前提条件

- JBoss EAP が実行されている。
- トラストマネージャーが設定されている。
  詳細については、クライアント証明書用のトラストストアとトラストマネージャーの設定 を参照してください。

手順

- 次のいずれかの手順で、OCSP による証明書失効に使用するトラストマネージャーを設定します。
  - 証明書失効チェックに、証明書で定義された OCSP レスポンダーを使用するようにトラストマネージャーを設定します。

    構文

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

    例

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

- 証明書で定義された OCSP レスポンダーを上書きします。

  構文

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

  例

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

関連情報

- **trust-manager** 属性

3.3. ELYTRON クライアントにおける CRL を使用した証明書失効チェックの設定

Elytron クライアントで証明書失効リスト (CRL) を使用して証明書失効チェックを設定し、有効期限前に発行元の認証局 (CA) によって失効された証明書をクライアントが信頼しないようにします。

前提条件

- Elytron クライアント用の **wildfly-config.xml** ファイルを作成しました。

手順

- **wildfly-config.xml** ファイルの **<ssl-context>** 要素に次のコンテンツを追加します。

  構文

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

  例

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

関連情報

- **trust-manager** 属性

3.4. ELYTRON クライアントにおける OCSP を使用した証明書失効チェックの設定

Elytron クライアントで Online Certificate Status Protocol (OCSP) を使用して証明書失効チェックを設定し、有効期限前に発行元の認証局 (CA) によって失効された証明書をクライアントが信頼しないようにします。OCSP レスポンダーを使用する場合、CRL 全体をダウンロードする必要はありません。

前提条件

- Elytron クライアント用の **wildfly-config.xml** ファイルを作成しました。

手順

- wildfly-config.xml の **<ssl-context>** 要素に次のコンテンツを追加します。

  構文

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

  例

  ```
  <ocsp />
  ```

関連情報

- **trust-manager** 属性

第4章 ELYTRON クライアントのデフォルト SSLCONTEXT セキュリティープロバイダーを JBOSS EAP クライアントで使用する

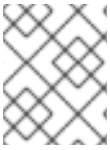
Java 仮想マシン (JVM) が、Elytron クライアント設定を使用してデフォルトの **SSLcontext** を提供するようにするには、**WildFlyElytronClientDefaultSSLContextProvider** を使用できます。このプロバイダーを使用して、デフォルトの **SSLContext** を要求する際に、クライアントライブラリーが Elytron クライアント設定を自動的に使用するようにします。

4.1. ELYTRON クライアントのデフォルト SSL コンテキストセキュリティープロバイダー

Elytron クライアントは、Java 仮想マシン (JVM) 全体のデフォルト SSL コンテキストの登録に使用できる Java セキュリティープロバイダー **org.wildfly.security.auth.client.WildFlyElytronClientDefaultSSLContextProvider** を提供します。

WildFlyElytronClientDefaultSSLContextProvider プロバイダーは以下のように動作します。

- プロバイダーは、**SSLContext.getDefault()** メソッドが呼び出されると、**SSLContext** をインスタンス化します。**SSLContext** は、以下のいずれかの場所から取得された認証コンテキストから開始されます。
  - プロバイダーへの引数として渡された Elytron クライアント設定ファイル。
  - ファイルシステム上で自動的に検出された **wildfly-config.xml** ファイル。詳細は、デフォルト設定アプローチ を参照してください。
    プロバイダーに引数として渡されるクライアント設定ファイルが優先されます。
- **SSLContext.getDefault()** メソッドが呼び出されると、JVM はプロバイダーによってインスタンス化された **SSLContext** を返します。
  Elytron クライアントには複数の SSL コンテキストを設定できるため、ルールを使用し、接続用に単一の SSL コンテキストを選択します。デフォルトの SSL コンテキストは、すべてのルールに一致するものです。このプロバイダーは、このデフォルトの SSL コンテキストを返します。



注記

デフォルトの **SSLContext** が設定されていないか、設定が存在しない場合、プロバイダーは無視されます。

WildFlyElytronClientDefaultSSLContextProvider プロバイダーを登録すると、**SSLContext.getDefault()** メソッドを使用するすべてのクライアントライブラリーは、コードで Elytron クライアント API を使用せずに Elytron クライアント設定を使用します。

プロバイダーを登録するには、以下のアーティファクトにランタイム依存関係を追加する必要があります。

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

プロバイダーは、プログラムを用いて、クライアントコードに、または **java.security** ファイルに静的に登録できます。登録して使用するプロバイダーを動的に決定する場合は、プログラムによる登録を使用します。

プロバイダーのプログラムでの登録

以下に示すように、プロバイダーをクライアントコードにプログラムで登録できます。

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

プロバイダーの静的登録

以下に示すように、プロバイダーを **java.security** ファイルに登録できます。

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

関連情報

- デフォルトの SSLContext をロードするクライアントの作成例

4.2. デフォルトの SSL コンテキストをロードするクライアントの作成例

以下の例は、**WildFlyElytronClientDefaultSSLContextProvider** プロバイダーをプログラムで登録し、**SSLContext.getDefault()** メソッドを使用して Elytron クライアントによって初期化された SSLContext を取得する方法を示しています。この例では、プロバイダーへの引数として提供された静的クライアント設定を使用します。

4.2.1. JBoss EAP クライアントの Maven プロジェクトの作成

JBoss EAP にデプロイされたアプリケーションのクライアントを作成するには、必要な依存関係とディレクトリー構造で Maven プロジェクトを作成します。

前提条件

- Maven がインストールされている。詳細は、Downloading Apache Maven を参照してください。

手順

1. **mvn** コマンドを使用して Maven プロジェクトをセットアップします。このコマンドは、プロジェクトのディレクトリー構造と **pom.xml** 設定ファイルを作成します。

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

2. アプリケーションのルートディレクトリーに移動します。

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

3. 生成された **pom.xml** ファイルの内容を、以下のテキストに置き換えます。

```
<?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** の依存関係。

(2) **wildfly-client-config** の依存関係。

4. **src/test** ディレクトリーを削除します。

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

検証

- アプリケーションのルートディレクトリーで、次のコマンドを入力します。

```
$ mvn install
```

次のような出力が得られます。

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

次のステップ

- デフォルトの SSLContext をロードするクライアントの作成

4.2.2. デフォルトの SSLContextをロードするクライアントの作成

SSLContext.getDefault() メソッドを使用して **SSLContext** をロードする JBoss EAP にデプロイされたアプリケーションのクライアントを作成します。

この手順では、**<application_home>** は、アプリケーションの **pom.xml** 設定ファイルが含まれるディレクトリーを参照します。

前提条件

- 双方向 TLS で JBoss EAP にデプロイされたアプリケーションを保護しました。
  これを実行するには、以下の手順を実行します。
  - クライアント証明書の生成
  - クライアント証明書のトラストストアとトラストマネージャーを設定します。
  - 双方向 SSL/TLS 用サーバー証明書の設定
  - SL/TLS を使用して JBoss EAP にデプロイされたアプリケーションを保護するための SSL コンテキストを設定します。
- Maven プロジェクトを作成している。
  詳細は、JBoss EAP クライアントの Maven プロジェクトの作成 を参照してください。
- JBoss EAP が実行されている。

手順

1. Java ファイルを保存するディレクトリーを作成します。

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

2. 新しいディレクトリーに移動します。

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

3. 以下の内容で Java ファイル **App.java** を作成します。

```
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 JBoss EAP ホームページの URL を定義します。

2 セキュリティープロバイダーを登録します。**1** は、このプロバイダーの優先度を定義します。プロバイダーを静的に登録するには、代わりに **java.security** ファイルに **security.provider.1=org.wildfly.security.auth.client.WildFlyElytronClientDefaultSSLContextProvider <PATH>/<TO>/wildfly-config-two-way-tls.xml** のようにプロバイダーを追加できます。

3 デフォルトの SSL コンテキストを取得します。

4. **<application_home>/src** ディレクトリーに、"wildfly-config-two-way-tls.xml" というクライアント設定ファイルを作成します。

```
<?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>
```

以下のプレースホルダーの値を、実際のパスに置き換えます。

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

検証

1. **<application_home>** ディレクトリーに移動します。
2. アプリケーションを実行します。

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

出力例

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

SSL Default SSLContext is: WildFlyElytronClientDefaultSSLContextProvider
```

第5章 参照

5.1. KEY-MANAGER 属性

属性を設定することで、**key-manager** を設定できます。

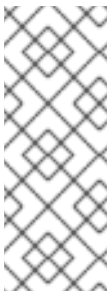
表5.1 key-manager 属性

属性	説明
algorithm	基礎となる **KeyManagerFactory** を作成するために使用するアルゴリズムの名前。これは JDK によって提供されます。たとえば、SunJSSE を使用する JDK は、**PKIX** および **SunX509** アルゴリズムを提供します。詳細については、Oracle Web サイトの Support Classes and Interfaces を参照してください。
alias-filter	キーストアから返されるエイリアスに適用するフィルター。これは、返すエイリアスのコンマ区切りのリストまたは次のいずれかの形式で指定できます。 • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**
credential-reference	キーストア項目を復号化するための認証情報リファレンス。これはクリアテキストで、または **credential-store** に保存されている認証情報への参照として指定できます。これはキーストアのパスワードではありません。
generate-self-signed-certificate-host	キーストアをサポートするファイルが存在せず、この属性が設定されている場合、指定されたホスト名に対して自己署名証明書が生成されます。実稼働環境ではこの属性を設定しないでください。
key-store	基礎となる **KeyManagerFactory** の初期化に使用する **key-store** への参照。
provider-name	基礎となる **KeyManagerFactory** を作成するために使用するプロバイダーの名前。
providers	基礎となる **KeyManagerFactory** の作成時に使用する **Provider[]** を取得するための参照。

5.2. KEY-STORE 属性

属性を設定することにより、**key-store** を設定できます。

表5.2 key-store 属性

属性	説明
alias-filter	キーストアから返されるエイリアスに適用するフィルターは、返すエイリアスのコンマ区切りリストまたは以下の形式のいずれかになります。 • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**  **注記** **alias-filter** 属性は、大文字と小文字を区別します。**elytronAppServer** などの大文字・小文字を合わせたエイリアスまたは大文字のエイリアスの使用は一部のキーストアプロバイダーで認識されない可能性があるため、**elytronappserver** などの小文字のエイリアスを使用することが推奨されます。
credential-reference	キーストアへのアクセスに使用するパスワード。これはクリアテキストで、または **credential-store** に保存されている認証情報への参照として指定できます。
path	キーストアファイルへのパス。
provider-name	キーストアのロードに使用するプロバイダーの名前。この属性を設定すると、指定されたタイプのキーストアを作成できる最初のプロバイダーの検索が無効になります。
providers	検索するプロバイダーインスタンスのリストを取得するために使用されるプロバイダーへの参照。指定しない場合は、代わりにプロバイダーのグローバルリストが使用されます。
relative-to	このストアが相対するベースパス。完全パスまたは **jboss.server.config.dir** などの事前定義パスを指定できます。
required	**true** に設定した場合、参照されるキーストアファイルは、キーストアサービスの開始時に存在している必要があります。デフォルト値は **false** です。

属性	説明
type	**JKS** などのキーストアタイプ。 **注記** 次のキーストアタイプが自動的に検出されます。 - **JKS** - **JCEKS** - **PKCS12** - **BKS** - **BCFKS** - **UBER** その他のキーストアタイプは手動で指定する必要があります。 キーストアタイプの完全なリストは、Oracle JDK ドキュメントの Java Cryptography Architecture Standard Algorithm Name Documentation for JDK 11 にあります。

5.3. SERVER-SSL-CONTEXT 属性

属性を設定することで、サーバー SSL コンテキスト (**server-ssl-context**) を設定できます。

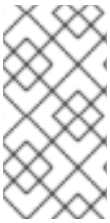
表5.3 server-ssl-context 属性

属性	説明
authentication-optional	**true** の場合、セキュリティードメインがクライアント証明書を拒否しても接続は妨害されません。このため、フォールスルーにより、クライアント証明書がセキュリティードメインによって拒否される場合に、別の認証メカニズム (フォームログインなど) を使用できます。これはセキュリティードメインが設定されている場合に限り有効になります。デフォルトは **false** です。

属性	説明
cipher-suite-filter	有効な暗号スイートを指定するために適用するフィルター。このフィルターは、コロン、コンマ、またはスペースで区切られた項目のリストを取得します。各項目は、OpenSSL 形式の暗号スイート名、標準の SSL/TLS 暗号スイート名、または **TLSv1.2** や **DES** などのキーワードになります。キーワードの詳細な一覧およびフィルター作成の詳細は、**CipherSuiteSelector** クラスの Javadoc で掲載されています。デフォルト値は **DEFAULT** で、**NULL** 暗号化のない既知の暗号スイートすべてに対応し、認証のない暗号スイートを除外します。
cipher-suite-names	TLSv1.3 の有効な暗号スイートを指定するために適用するフィルター。
final-principal-transformer	このメカニズムレルムに適用する最終のプリンシパルトランスフォーマー。
key-manager	**SSLContext** 内で使用するキーマネージャーへの参照。
maximum-session-cache-size	キャッシュされる SSL/TLS セッションの最大数。
need-client-auth	**true** の場合は、SSL ハンドシェイクでクライアント証明書が必要になります。信頼されたクライアント証明書がない接続は拒否されます。デフォルトは **false** です。
post-realm-principal-transformer	レルムの選択後に適用するプリンシパルトランスフォーマー。
pre-realm-principal-transformer	レルムが選択される前に適用するプリンシパルトランスフォーマー。

属性	説明
プロトコル	有効なプロトコル。許可されているオプション: • **SSLv2** • **SSLv3** • **TLSv1** • **TLSv1.1** • **TLSv1.2** • **TLSv1.3** これは、デフォルトで **TLSv1**、**TLSv1.1**、**TLSv1.2**、および **TLSv1.3** を有効にするように設定されています。 **警告** SSLv2、SSLv3、TLSv1.0 の代わりに、TLSv1.2 または TLSv1.3 を使用します。SSLv2、SSLv3、または TLSv1.0 を使用するとセキュリティー上のリスクが生じるため、明示的に無効にする必要があります。 プロトコルを指定しない場合は、**cipher-suite-names** を設定すると、**protocols** の値が **TLSv1.3** に設定されます。
provider-name	使用するプロバイダーの名前。指定されていない場合は、プロバイダーからの **providers** はすべて **SSLContext** に渡されます。
providers	**SSLContext** の読み込みに使用する **Provider[]** を取得するプロバイダーの名前。
realm-mapper	SSL/TLS 認証に使用されるレルムマッパー。
security-domain	SSL/TLS セッション確立中の認証に使用するセキュリティードメイン。

属性	説明
session-timeout	SSL セッションのタイムアウト (秒単位)。 値 **-1** は、Java 仮想マシン (JVM) のデフォルト値を使用するよう Elytron に指示します。 値 **0** は、タイムアウトがあることを示します。 デフォルト値は **-1** です。
trust-manager	SSLContext 内で使用する **trust-manager** への参照。
use-cipher-suites-order	**true** に設定すると、サーバーで定義された暗号スイートの順序が使用されます。**false** に設定すると、クライアントによって提示された暗号スイートの順序が使用されます。デフォルトは **true** です。
want-client-auth	**true** に設定すると、SSL ハンドシェイクでクライアント証明書が要求されますが、必須ではありません。セキュリティードメインが参照され、X509 エビデンスをサポートしている場合、**want-client-auth** は自動的に **true** に設定されます。これは、**need-client-auth** が設定されると無視されます。デフォルトは **false** です。
wrap	**true** の場合は、返された **SSLEngine**、**SSLSocket**、および **SSLServerSocket** インスタンスがラップされ、さらなる変更に対して保護されます。デフォルトは **false** です。



注記

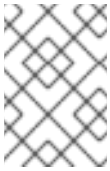
server-ssl-context の **realm-mapper** 属性および **principal-transformer** 属性は、証明書がトラストマネージャーによって検証される SASL EXTERNAL メカニズムにのみ適用されます。HTTP CLIENT-CERT 認証設定は、**http-authentication-factory** で設定されます。

5.4. TRUST-MANAGER 属性

属性を設定することで、トラストマネージャー (**trust-manager**) を設定できます。

表5.4 trust-manager 属性

属性	説明
algorithm	基礎となる **TrustManagerFactory** を作成するために使用するアルゴリズムの名前。これは JDK によって提供されます。たとえば、SunJSSE を使用する JDK は、**PKIX** および **SunX509** アルゴリズムを提供します。SunJSSE の詳細は、Oracle ドキュメントの Java Secure Socket Extension (JSSE) Reference Guide で Support Classes and Interfaces を参照してください。

属性	説明
alias-filter	key-store から返されたエイリアスに適用するフィルター。これは、返すエイリアスのコンマ区切りのリストまたは次のいずれかの形式で指定できます。 • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**
certificate-revocation-list	トラストマネージャーで証明書失効リストのチェックを有効にします。この属性を使用して定義できる CRL パスは 1 つだけです。複数の CRL パスを定義するには、**certificate-revocation-lists** を使用します。**certificate-revocation-list** の属性は以下の通りです。 • **maximum-cert-path**: 証明書パスに存在可能な自己発行でない中間証明書の最大数。デフォルト値は **5** です。この属性は非推奨になりました。代わりに、**trust-manager** では **maximum-cert-path** を使用してください。 • **path** - 証明書失効リストへのパス。 • **relative-to**: 証明書失効リストファイルのベースパス。
certificate-revocation-lists	複数の証明書失効リストを使用して、トラストマネージャーで証明書失効リストチェックを有効にします。**certificate-revocation-list** の属性は以下の通りです。 • **path** - 証明書失効リストへのパス。 • **relative-to**: 証明書失効リストファイルのベースパス。
key-store	基礎となる **TrustManagerFactory** の初期化に使用する **key-store** への参照。
maximum-cert-path	証明書パスに存在可能な自己発行でない中間証明書の最大数。デフォルト値は **5** です。 この属性は、JBoss EAP 7.3 では、 **trust-manager** 内で **certificate-revocation-list** から **trust-manager** に移動されています。後方互換性を確立するためにも、属性は **certificate-revocation-list** にも存在します。次に進む前に、**trust-manager** で **maximum-cert-path** を使用します。  **注記** **trust-manager** または **certificate-revocation-list** のいずれかで **maximum-cert-path** を定義します。

属性	説明
ocsp	トラストマネージャーでのオンライン証明書ステータスプロトコル (OCSP) チェックを有効にします。**ocsp** の属性は次のとおりです。 ● **responder** - 証明書から解決された OCSP レスポンダー URI をオーバーライドします。 ● **responder-certificate** - **responder-keystore** が定義されていない場合は、**responder-keystore** または **trust-manager** キーストアにあるレスポンダー証明書のエイリアス。 ● **Responder-keystore** - レスポンダー証明書の代替キーストア。**Responder-certificate** を定義する必要があります。 ● **prefer-crls** - OCSP メカニズムと CRL メカニズムの両方が設定されている場合、OCSP メカニズムが最初に呼び出されます。**Prefer-crls** を **true** に設定すると、CRL メカニズムが最初に呼び出されます。
only-leaf-cert	リーフ証明書のみの失効ステータスを確認します。これは任意の属性です。デフォルト値は **false** です。
provider-name	基礎となる **TrustManagerFactory** を作成するために使用するプロバイダーの名前。
providers	基礎となる **TrustManagerFactory** の作成時に使用する providers を取得するための参照。