Red Hat JBoss Enterprise Application Platform 8.0

JBoss EAP でのシングルサインオンの使用

JBoss EAP にデプロイされたアプリケーションにシングルサインオンを使用して認証を追加するためのガイド

Last Updated: 2024-05-15

Red Hat JBoss Enterprise Application Platform 8.0 JBoss EAP でのシングルサインオンの使用

JBoss EAP にデプロイされたアプリケーションにシングルサインオンを使用して認証を追加するためのガイド

法律上の通知

概要

JBoss EAP にデプロイされたアプリケーションにシングルサインオンを使用して認証を追加するためのガイド。

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 JBOSS EAP でのシングルサインオン

シングルサインオン (SSO) は、中央のアイデンティティープロバイダーから複数のクライアントのアイデンティティーを認証するプロセスです。たとえば、同じ SSO プロバイダーを使用するさまざまなアプリケーションにログインする場合、必要なログイン認証情報が 1 セットだけで済みます。

JBoss EAP は次の SSO プロトコルをサポートします。

OpenID Connect (OIDC)

OpenID Connect は、RFC 6749 および RFC 6750 で指定された仕様の OAuth 2.0 フレームワークに基づく認証プロトコルです。

Security Assertion Mark-up Language v2 (SAML v2)

SAML は、2 者間 (通常はアイデンティティープロバイダーとサービスプロバイダー) での認証および認可情報の交換を可能にするデータ形式およびプロトコルです。この情報は、アサーションを含む SAML トークンの形式で交換され、サービスプロバイダーによる認証のためにアイデンティティプロバイダーによってサブジェクトに発行されます。サブジェクトは、アイデンティティープロバイダーによって発行された SAML トークンを複数のサービスプロバイダーで再利用でき、SAML v2 によるブラウザーベースのシングルサインオンをサポートします。

SSO を使用すると、ベアメタル上で実行されている JBoss EAP および Red Hat OpenShift Container Platform 上で実行されている JBoss EAP にデプロイしたアプリケーションを保護できます。Red Hat OpenShift Container Platform 上で実行されている JBoss EAP にデプロイしたアプリケーションを SSO を使用して保護する方法については、OpenShift Container Platform 上の JBoss EAP の使用 を参照してください。

第2章 シングルサインオンによる JBOSS EAP にデプロイしたアプリケーションの保護

シングルサインオン (SSO) を使用してアプリケーションを保護し、Red Hat build of Keycloak などの SSO プロバイダーに認証を委譲できます。OpenID Connect (OIDC) または Security Assertion Markup Language v2 (SAML v2) のいずれかを SSO プロトコルとして使用できます。

SSO を使用してアプリケーションを保護するには、次の手順に従います。

- シングルサインオンで保護するサンプルアプリケーションの作成: この手順を使用して、SSO で保護するための単純な web-application を作成します。SSO で保護するアプリケーションがすでにある場合は、この手順をスキップしてください。
- Red Hat build of Keycloak でのレルムとユーザーの作成
- OIDC または SAML をプロトコルとして使用して、SSO でアプリケーションを保護します。
  - OIDC によるアプリケーションの保護
  - SAML によるアプリケーションの保護

2.1. シングルサインオンで保護するサンプルアプリケーションの作成

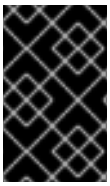
web-application を作成して JBoss EAP にデプロイし、OpenID Connect (OIDC) または Security Assertion Mark-up Language (SAML) を使用してシングルサインオン (SSO) で保護します。

注記

以下の手順は例としてのみ提供されています。保護する必要があるアプリケーションがすでにある場合は、以下の手順をスキップして、Red Hat build of Keycloak でのレルムとユーザーの作成 に直接進むことができます。

2.1.1. web-application 開発用の Maven プロジェクトの作成

web-application 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。



重要

以下の手順は一例として提示されています。実稼働環境では使用しないでください。JBoss EAP のアプリケーション作成に関する詳細は、JBoss EAP にデプロイするアプリケーションの開発のスタートガイド を参照してください。

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 プロジェクトを設定します。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。

   構文

```
$ mvn archetype:generate \
-DgroupId=${group-to-which-your-application-belongs} \
-DartifactId=${name-of-your-application} \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

例:

```
$ mvn archetype:generate \
-DgroupId=com.example.app \
-DartifactId=simple-webapp-example \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-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。

   構文

   ```
   $ cd <name-of-your-application>
   ```

   例:

   ```
   $ cd simple-webapp-example
   ```

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.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
      <version.maven.war.plugin>3.4.0</version.maven.war.plugin>
    </properties>

    <dependencies>
      <dependency>
```

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

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

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: 0.795 s
[INFO] Finished at: 2022-04-28T17:39:48+05:30
[INFO] ------------------------------------------------------------------------
```

次のステップ

- Web アプリケーションの作成

2.1.2. Web アプリケーションの作成

ログインユーザーのプリンシパルから取得したユーザー名を返すサーブレットを含む Web アプリケーションを作成します。ログインしているユーザーがないと、サーブレットは「NO AUTHENTICATED USER」のテキストを返します。

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

- Maven プロジェクトを作成している。
  詳細は、web-application 開発用の 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。

   構文

   ```
   $ mkdir -p src/main/java/<path_based_on_artifactID>
   ```

   例:

   ```
   $ mkdir -p src/main/java/com/example/app
   ```

2. 新しいディレクトリーに移動します。

   構文

   ```
   $ cd src/main/java/<path_based_on_artifactID>
   ```

   例:

   ```
   $ cd src/main/java/com/example/app
   ```

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

```
package com.example.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

/**
 * A simple secured HTTP servlet. It returns the user name of obtained
 * from the logged-in user's Principal. If there is no logged-in user,
 * it returns the text "NO AUTHENTICATED USER".
 */

@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
```

```
        try (PrintWriter writer = resp.getWriter()) {
            writer.println("<html>");
            writer.println("  <head><title>Secured Servlet</title></head>");
            writer.println("  <body>");
            writer.println("    <h1>Secured Servlet</h1>");
            writer.println("    <p>");
            writer.print(" Current Principal '");
            Principal user = req.getUserPrincipal();
            writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
            writer.print("'");
            writer.println("    </p>");
            writer.println("  </body>");
            writer.println("</html>");
        }
    }

}
```

4. アプリケーションのルートディレクトリーで、次のコマンドを使用してアプリケーションをコンパイルします。

```
$ mvn package
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.015 s
[INFO] Finished at: 2022-04-28T17:48:53+05:30
[INFO] ------------------------------------------------------------------------
```

5. アプリケーションをデプロイします。

```
$ mvn wildfly:deploy
```

検証

- ブラウザーで **http://localhost:8080/simple-webapp-example/secured** に移動します。以下のメッセージが表示されます。

```
Secured Servlet
Current Principal 'NO AUTHENTICATED USER'
```

認証メカニズムが追加されないため、アプリケーションにアクセスできます。

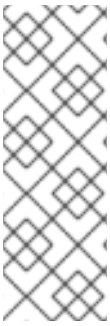
次のステップ

- Red Hat build of Keycloak でのレルムとユーザーの作成

2.2. RED HAT BUILD OF KEYCLOAK でのレルムとユーザーの作成

Red Hat build of Keycloak のレルムはテナントに相当します。各レルムにより、管理者はアプリケーションとユーザーの分離グループを作成できます。

以下の手順は、テスト目的で Red Hat build of Keycloak を使用して JBoss EAP にデプロイしたアプリケーションの保護を開始するために必要な最小限の手順を示しています。詳細な設定については、Red Hat build of Keycloak サーバー管理ガイド を参照してください。



注記

以下の手順は例としてのみ提供されています。Red Hat build of Keycloak でレルムとユーザーをすでに設定している場合は、この手順をスキップして、アプリケーションの保護に直接進むことができます。詳細は以下を参照してください。

- OIDC によるアプリケーションの保護
- SAML によるアプリケーションの保護

前提条件

- Red Hat build of Keycloak への管理者アクセス権を持っている。

手順

1. JBoss EAP のデフォルトポートは 8080 であるため、Red Hat build of Keycloak サーバーを 8080 以外のポートで起動します。

注記

start-dev コマンドは実稼働環境向けではありません。詳細は、Red Hat build of Keycloak サーバーガイドの 開発モードで Red Hat build of Keycloak を試用する を参照してください。

構文

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

例:

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

2. **http://localhost:<port>/** で管理コンソールにログインします。たとえば、**http://localhost:8180/** です。
3. レルムを作成します。
   a. **Master** の上にカーソルを合わせて、**Create Realm** をクリックします。
   b. レルムの名前を入力します。たとえば、**example_realm**。
   c. **Enabled** が **ON** に設定されていることを確認します。
   d. **Create** をクリックします。

   詳細は、Red Hat build of Keycloak サーバー管理ガイドの レルムの作成 を参照してください。
4. ユーザーを作成します。

a. **Users** をクリックし、**Add user** をクリックします。

b. ユーザー名を入力します。たとえば、**user1** です。

c. **Create** をクリックします。

詳細は、Red Hat build of Keycloak サーバー管理ガイドの ユーザーの作成 を参照してください。

5. ユーザーの認証情報を設定します。

a. **Credentials** をクリックします。

b. ユーザーのパスワードを設定します。たとえば、**passwordUser1** です。**Temporary** を **OFF** に切り替えて、**Set Password** をクリックします。確認プロンプトで、**Save** をクリックします。

詳細は、Red Hat build of Keycloak サーバー管理ガイドの ユーザー認証情報の定義 を参照してください。

6. ロールを作成します。
これは、JBoss EAP で承認のために設定するロール名です。

a. **Realm Roles** をクリックし、**Create role** をクリックします。

b. ロール名 (**Admin** など) を入力します。

c. **Save** をクリックします。

7. ユーザーにロールを割り当てます。

a. **Users** をクリックします。

b. ロールを割り当てるユーザーをクリックします。

c. **Role Mapping** をクリックします。

d. **Assign role** をクリックします。

e. 割り当てるロールを選択します。たとえば、**Admin** です。**Assign** をクリックします。

詳細は、Red Hat build of Keycloak サーバー管理ガイドの レルムロールの作成 を参照してください。

次のステップ

- このレルムを使用して、JBoss EAP にデプロイし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。
  - OIDC によるアプリケーションの保護
  - SAML によるアプリケーションの保護

関連情報

- Red Hat build of Keycloak スタートガイド

2.3. OIDC によるアプリケーションの保護

JBoss EAP のネイティブ OpenID Connect (OIDC) クライアントを使用し、外部の OpenID プロバイダーを使用してアプリケーションを保護します。OIDC は、JBoss EAP などのクライアントが OpenID プロバイダーによって実行される認証に基づいてユーザーのアイデンティティーを検証できるようにするアイデンティティーレイヤーです。たとえば、Red Hat build of Keycloak を OpenID プロバイダーとして使用して、JBoss EAP アプリケーションを保護できます。

OIDC を使用してアプリケーションを保護するには、次の手順に従います。

- JBoss EAP での OIDC クライアントの作成
- OpenID Connect を使用した Web アプリケーションの保護

2.3.1. JBoss EAP の OpenID Connect によるアプリケーションの保護

OpenID プロバイダーを使用してアプリケーションを保護する場合、セキュリティードメインリソースをローカルで設定する必要はありません。**elytron-oidc-client** サブシステムは、OpenID プロバイダー (OP) に接続するための JBoss EAP のネイティブ OpenID Connect (OIDC) クライアントを提供します。JBoss EAP は、OpenID プロバイダーの設定に基づいて、アプリケーションの仮想セキュリティードメインを自動的に作成します。**elytron-oidc-client** サブシステムは、Relying Party (RP) として機能します。

注記

JBoss EAP のネイティブ OIDC クライアントは、RP が開始するログアウトをサポートしません。

重要

OIDC クライアントは Red Hat build of Keycloak で使用することを推奨します。JSON Web トークン (JWT) であるアクセストークンを使用するように設定でき、RS256、RS384、RS512、ES256、ES384、または ES512 署名アルゴリズムを使用するように設定できる場合は、他の OpenID プロバイダーを使用できます。

OIDC の使用を有効にするには、**elytron-oidc-client** サブシステムまたはアプリケーション自体を設定できます。JBoss EAP は、次のように OIDC 認証をアクティブにします。

- アプリケーションを JBoss EAP にデプロイすると、**elytron-oidc-client** サブシステムがデプロイメントをスキャンして、OIDC 認証メカニズムが必要かどうかを検出します。
- サブシステムが **elytron-oidc-client** サブシステムまたはアプリケーションデプロイメント記述子のいずれかでデプロイメントの OIDC 設定を検出した場合、JBoss EAP はアプリケーションの OIDC 認証メカニズムを有効にします。
- サブシステムが両方の場所で OIDC 設定を検出した場合、**elytron-oidc-client** サブシステム **secure-deployment** 属性の設定が、アプリケーションデプロイメント記述子の設定よりも優先されます。

デプロイメント設定
デプロイメント記述子を使用して OIDC でアプリケーションを保護するには、アプリケーションのデプロイメント設定を次のように更新します。

- アプリケーションデプロイメント記述子 **web.xml** ファイルで **auth-method** プロパティーを **OIDC** に設定します。

デプロイメント記述子の更新例

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

- OIDC 設定情報を含む **oidc.json** というファイルを **WEB-INF** ディレクトリーに作成します。

  oidc.json コンテンツの例

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

(1) OpenID プロバイダーで OIDC クライアントを識別するための名前。

(2) OpenID プロバイダーの URL。

(3) 外部リクエストには HTTPS が必要です。

(4) OpenID プロバイダーに登録されたクライアントシークレット。

サブシステムの設定
次の方法で **elytron-oidc-client** サブシステムを設定することで、OIDC を使用してアプリケーションを保護できます。

- アプリケーションごとに同じ OpenID プロバイダーを使用する場合は、複数のデプロイメントに対して単一の設定を作成します。
- アプリケーションごとに異なる OpenID プロバイダーを使用する場合は、デプロイメントごとに異なる設定を作成します。

単一デプロイメントの XML 設定の例:

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

(1) デプロイメントランタイム名。

(2) OpenID プロバイダーで OIDC クライアントを識別するための名前。

(3) OpenID プロバイダーの URL。

4 外部リクエストには HTTPS が必要です。

5 OpenID プロバイダーに登録されたクライアントシークレット。

同じ OpenID プロバイダーを使用して複数のアプリケーションを保護するには、次の例に示すように、**provider** を個別に設定します。

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

1 デプロイメント: **customer-portal.war**

2 別のデプロイメント: **product-portal.war**

関連情報

- OpenID Connect specification
- **elytron-oidc-client** サブシステム属性
- OpenID Connect ライブラリー

2.3.2. Red Hat build of Keycloak での OIDC クライアントの作成

Red Hat build of Keycloak で OpenID Connect (OIDC) クライアントを作成し、JBoss EAP で使用してアプリケーションを保護します。

以下の手順は、テスト目的で Red Hat build of Keycloak を使用して JBoss EAP にデプロイしたアプリケーションの保護を開始するために必要な最小限の手順を示しています。詳細な設定については、Red Hat build of Keycloak サーバー管理ガイドの OpenID Connect クライアントの管理 を参照してください。

前提条件

- Red Hat build of Keycloak でレルムを作成し、ユーザーを定義している。
  詳細は、JBoss EAP でのレルムとユーザーの作成 を参照してください。

手順

1. Red Hat build of Keycloak 管理コンソールに移動します。
2. クライアントを作成します。
   a. **Clients** をクリックし、**Create client** をクリックします。
   b. **Client type** が **OpenID Connect** に設定されていることを確認します。
   c. クライアント ID を入力します。たとえば、**jbeap-oidc** です。
   d. **Next** をクリックします。
   e. **Capability Config** タブで、**Authentication Flow** が **Standard flow** および **Direct access grants** に設定されていることを確認します。
   f. **Next** をクリックします。
   g. **Login settings** タブで、**Valid redirect URIs** の値を入力します。認証成功後のページのリダイレクト先 URL を入力します (例: **http://localhost:8080/simple-webapp-example/secured/***)。
   h. **Save** をクリックします。
3. アダプターの設定を表示します。
   a. **Action** をクリックし、**Download adapter config** をクリックします。
   b. **Keycloak OIDC JSON** を **Format Option** として選択して、接続パラメーターを確認します。

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

Red Hat build of Keycloak をアイデンティティプロバイダーとして使用するように JBoss EAP アプリケーションを設定する場合は、パラメーターを次のように使用します。

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

次のステップ

- OpenID Connect を使用した Web アプリケーションの保護

関連情報

- アプリケーションおよびサービスの保護ガイド

2.3.3. OpenID Connect を使用した Web アプリケーションの保護

アプリケーションを保護するには、そのデプロイメント設定を更新するか、**elytron-oidc-client** サブシステムを設定します。

Web アプリケーションの作成 の手順で作成したアプリケーションを使用する場合、Principal の値は OpenID プロバイダーの ID トークンから取得されます。デフォルトでは、Principal はトークンからの "sub" クレームの値です。“email”、“preferred_username”、“name”、“given_name”、“family_name”、または “nickname” クレームの値をプリンシパルとして使用することもできます。次のいずれかの場所で、ID トークンのどのクレーム値をプリンシパルとして使用するかを指定します。

- **elytron-oidc-client** サブシステム属性 **principal-attribute**。
- **The oidc.json file**。

OIDC を使用するようにアプリケーションを設定するには、次の 2 つの方法があります。

- **elytron-oidc-client** サブシステムを設定する方法
  アプリケーションのデプロイメントに設定を追加しない場合は、この方法を使用します。
- デプロイメント設定を更新する方法
  サーバーに設定を追加せず、アプリケーションのデプロイメント内に設定を保持する場合は、この方法を使用します。

前提条件

- JBoss EAP にアプリケーションをデプロイしました。

手順

1. アプリケーションの **web.xml** を設定して、アプリケーションリソースを保護します。

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

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

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

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

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

1 **Admin** ロールを持つユーザーのみがアプリケーションにアクセスできるようにします。任意のロールを持つユーザーがアプリケーションにアクセスできるようにするには、**role-**

2. OpenID Connect を使用してアプリケーションを保護するには、デプロイメント設定を更新するか、**elytron-oidc-client** サブシステムを設定します。

注記

デプロイメント設定と **elytron-oidc-client** サブシステムの両方で OpenID Connect を設定する場合、**elytron-oidc-client** サブシステムの **secure-deployment** 属性の設定は、アプリケーションデプロイメント記述子の設定よりも優先されます。

- デプロイメント設定の更新:

  i. 認証方法として OIDC を指定するログイン設定を、アプリケーションの **web.xml** に追加します。

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

1 OIDC を使用してアプリケーションを保護します。

  ii. 次のように、**WEB-INF** ディレクトリーにファイル **oidc.json** を作成します。

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

- **elytron-oidc-client** サブシステムの設定:

  - アプリケーションを保護するには、次の管理 CLI コマンドを使用します。

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

3. アプリケーションのルートディレクトリーで、次のコマンドを使用してアプリケーションをコンパイルします。

```
$ mvn package
```

4. アプリケーションをデプロイします。

```
$ mvn wildfly:deploy
```

検証

1. ブラウザーで **http://localhost:8080/simple-webapp-example/secured** に移動します。Red Hat build of Keycloak のログインページにリダイレクトされます。
2. Red Hat build of Keycloak で定義したユーザーの認証情報を使用してログインできます。

これで、アプリケーションは OIDC を使用して保護されました。

関連情報

- **elytron-oidc-client** サブシステム属性

2.4. SAML によるアプリケーションの保護

Keycloak SAML アダプター機能パックによって提供される Galleon レイヤーを使用すると、Security Assertion Markup Language (SAML) で Web アプリケーションを保護できます。

Keycloak SAML アダプター機能パックについては、SAML を使用してアプリケーションを保護するための Keycloak SAML アダプター機能パック を参照してください。

SAML を使用してアプリケーションを保護するには、次の手順に従います。

- SAML を使用した Web アプリケーションの保護

2.4.1. JBoss EAP の SAML によるアプリケーションの保護

Keycloak SAML アダプター Galleon パックは、**keycloak-saml**、**keycloak-client-saml**、および **keycloak-client-saml-ejb** という 3 つのレイヤーを含む Galleon 機能パックです。この機能パックのレイヤーを使用して、Security Assertion Markup Language (SAML) を使用したシングルサインオンのアイデンティティープロバイダーとして Red Hat build of Keycloak を使用するために必要なモジュールと設定を JBoss EAP にインストールします。

次の表で各レイヤーの用途を説明します。

レイヤー	適用対象	説明
keycloak-saml	OpenShift	このレイヤーは、SAML クライアントの自動登録とともに Source to Image (s2i) に使用します。このレイヤーは **cloud-default-config** レイヤーと一緒に使用する必要があります。

レイヤー	適用対象	説明
keycloak-client-saml	ベアメタル、OpenShift	このレイヤーは、ベアメタル上の web-application に使用します。また、CLI スクリプトまたはデプロイメント設定で提供される **keycloak-saml** サブシステム設定を使用する Source to Image (s2i) に使用します。
keycloak-client-saml-ejb	ベアメタル	このレイヤーは、Jakarta Enterprise Beans にアイデンティティーを伝播するアプリケーションに使用します。

SAML の使用を有効にするには、**keycloak-saml** サブシステムまたはアプリケーション自体を設定できます。

デプロイメント設定
デプロイメント記述子を使用して SAML でアプリケーションを保護するには、アプリケーションのデプロイメント設定を次のように更新します。

- アプリケーションのデプロイメント記述子 **web.xml** ファイルで、**auth-method** プロパティーを **SAML** に設定します。

  デプロイメント記述子の更新例

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

- SAML 設定情報を含む **keycloak-saml.xml** というファイルを **WEB-INF** ディレクトリーに作成します。このファイルは SAML プロバイダーから取得できます。

  keycloak-saml.xml の例

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

```
                        validateResponseSignature="true"
                        validateAssertionSignature="false"
                        requestBinding="POST"

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

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

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

PrivateKeyPem および **CertificatePem** の値は、クライアントごとに一意です。

サブシステムの設定
keycloak-saml サブシステムを設定することで、SAML を使用してアプリケーションを保護できます。サブシステム設定コマンドを含むクライアント設定ファイルは、Red Hat build of Keycloak から取得できます。詳細は、クライアントアダプター設定の生成 を参照してください。

2.4.2. Red Hat build of Keycloak での SAML クライアントの作成

Red Hat build of Keycloak で Security Assertion Markup Language (SAML) クライアントを作成し、JBoss EAP で使用してアプリケーションを保護します。

以下の手順は、テスト目的で Red Hat build of Keycloak を使用して JBoss EAP にデプロイしたアプリケーションの保護を開始するために必要な最小限の手順を示しています。詳細な設定については、Red Hat build of Keycloak サーバー管理ガイドの SAML クライアントの作成 を参照してください。

前提条件

- Red Hat build of Keycloak でレルムを作成し、ユーザーを定義している。
  詳細は、JBoss EAP でのレルムとユーザーの作成 を参照してください。

手順

1. Red Hat build of Keycloak 管理コンソールに移動します。
2. クライアントを作成します。
   a. **Clients** をクリックし、**Create client** をクリックします。
   b. **Client type** として **SAML** を選択します。
   c. 保護するアプリケーションの URL を **Client ID** として入力します。たとえば、**http://localhost:8080/simple-webapp-example/secured/** です。

重要

クライアント ID はアプリケーションの URL と正確に一致する必要があります。クライアント ID が一致しない場合は、次のようなエラーが表示されます。

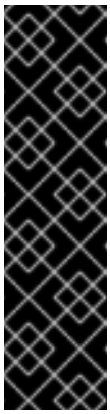
```
2023-05-17 19:54:31,586 WARN  [org.keycloak.events] (executor-thread-0) type=LOGIN_ERROR, realmId=eba0f106-389f-4216-a676-05fcd0c0c72e, clientId=null, userId=null, ipAddress=127.0.0.1, error=client_not_found, reason=Cannot_match_source_hash
```

d. クライアント名を入力します。たとえば、**jbeap-saml** です。

e. **Next** をクリックします。

f. 以下の情報を入力します。

- **Root URL**: アプリケーションの URL (例: **http://localhost:8080/simple-webapp-example/**)。
- **Home URL**: アプリケーションの URL (例: **http://localhost:8080/simple-webapp-example/**)。



重要

Home URL を設定しないと、クライアント設定の **SP entityID** が空白のままになり、エラーが発生します。

- 管理 CLI コマンドを使用すると、次のエラーが発生します。

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

それぞれの設定ファイルで **SP entityID** の値を定義することで、エラーを解決できます。

- **Valid Redirect URI**: ユーザーのログイン後に利用可能な URI (例: **http://localhost:8080/simple-webapp-example/secured/***)。
- **Master SAML Processing URL**: アプリケーションの URL の後に **saml** を付けます。たとえば、**http://localhost:8080/simple-webapp-example/saml** です。

重要

URL に **saml** を追加しないと、リダイレクトエラーが発生します。

詳細は、SAML クライアントの作成 を参照してください。

これで、設定したクライアントを使用して、JBoss EAP にデプロイした Web アプリケーションを保護できるようになりました。詳細は、SAML を使用した Web アプリケーションの保護 を参照してください。

次のステップ

- SAML を使用した Web アプリケーションの保護

関連情報

- Red Hat build of Keycloak サーバー管理ガイド

2.4.3. SAML を使用した Web アプリケーションの保護

Keycloak SAML アダプター機能パックは、OpenShift 以外のデプロイメント用の **keycloak-client-saml** および **keycloak-client-saml-ejb** という 2 つのレイヤーを提供します。**keycloak-client-saml** レイヤーはサーブレットベースの Web アプリケーションの保護に、**keycloak-client-saml-ejb** は Jakarta Enterprise Beans アプリケーションの保護に使用します。

SAML を使用するようにアプリケーションを設定するには、次の 2 つの方法があります。

- **keycloak-saml** サブシステムを設定する方法
  アプリケーションのデプロイメントに設定を追加しない場合は、この方法を使用します。

- デプロイメント設定を更新する方法
  サーバーに設定を追加せず、アプリケーションのデプロイメント内に設定を保持する場合は、この方法を使用します。

前提条件

- Red Hat build of Keycloak で SAML クライアントを作成している。
  詳細は、Red Hat build of Keycloak での SAML クライアントの作成 を参照してください。

- **jboss-eap-installation-manager** を使用して JBoss EAP をインストールしている。
  詳細は、**Red Hat JBoss Enterprise Application Platform のインストール方法**ガイドの **jboss-eap-installation-manager** を使用した JBoss EAP 8.0 のインストール を参照してください。

手順

1. **jboss-eap-installation-manager** を使用して、必要な Keycloak SAML アダプターレイヤーをサーバーに追加します。利用可能なレイヤーの詳細は次のとおりです。

   - 機能パック: **org.keycloak:keycloak-saml-adapter-galleon-pack**

   - レイヤー:

     - **keycloak-client-saml**: このレイヤーを使用してサーブレットを保護します。

     - **keycloak-client-saml-ejb**: このレイヤーを使用して、サーブレットから Jakarta Enterprise Beans にアイデンティティーを伝播します。
       JBoss EAP での機能パックとレイヤーの追加については、**Red Hat JBoss Enterprise Application Platform のインストール方法** ガイドの jboss-eap-installation-manager を使用した既存の JBoss EAP サーバーへの機能パックの追加 を参照してください。

2. アプリケーションの **web.xml** を設定して、アプリケーションリソースを保護します。

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

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

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

      <auth-constraint>
        <role-name>Admin</role-name>
      </auth-constraint>
    </security-constraint>

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

1 **Admin** ロールを持つユーザーのみがアプリケーションにアクセスできるようにします。任意のロールを持つユーザーがアプリケーションにアクセスできるようにするには、**role-name** の値としてワイルドカード ** を使用します。

3. 管理 CLI を使用するか、アプリケーションのデプロイメントを更新することにより、SAML でアプリケーションを保護します。

- アプリケーションのデプロイメントを更新する場合

a. 認証方法として SAML を指定するログイン設定を、アプリケーションの **web.xml** に追加します。

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

1 SAML を使用してアプリケーションを保護します。

b. Red Hat build of Keycloak から **keycloak-saml.xml** 設定ファイルをダウンロードし、アプリケーションの **WEB-INF/** ディレクトリーに保存します。
詳細は、クライアントアダプター設定の生成 を参照してください。

keycloak-saml.xml の例

```
<keycloak-saml-adapter>
    <SP entityID=""
        sslPolicy="EXTERNAL"
        logoutPage="SPECIFY YOUR LOGOUT PAGE!">
        <Keys>
            <Key signing="true">
                <PrivateKeyPem>PRIVATE KEY NOT SET UP OR
KNOWN</PrivateKeyPem>
                <CertificatePem>...</CertificatePem>
            </Key>
```

```
        </Keys>
        <IDP entityID="idp"
            signatureAlgorithm="RSA_SHA256"
            signatureCanonicalizationMethod="http://www.w3.org/2001/10/xml-exc-
c14n#">
            <SingleSignOnService signRequest="true"
                                 validateResponseSignature="true"
                                 validateAssertionSignature="false"
                                 requestBinding="POST"

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

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

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

PrivateKeyPem および **CertificatePem** の値は、クライアントごとに一意です。

- 管理 CLI を使用する場合

  a. Red Hat build of Keycloak からクライアント設定ファイル **keycloak-saml-subsystem.cli** をダウンロードします。
  詳細は、クライアントアダプター設定の生成 を参照してください。

  keycloak-saml-subsystem.cli の例

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

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

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

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

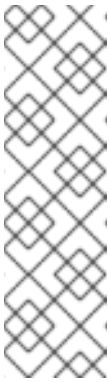
```
    SingleLogoutService={ \
        signRequest=true, \
        signResponse=true, \
        validateRequestSignature=true, \
        validateResponseSignature=true, \
        requestBinding=POST, \
        responseBinding=POST, \
        postBindingUrl=http://localhost:8180/realms/example-saml-realm/protocol/saml,
\
        redirectBindingUrl=http://localhost:8180/realms/example-saml-
realm/protocol/saml} \
)

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

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

PrivateKeyPem および **CertificatePem** の値は、クライアントごとに一意です。

b. クライアント設定ファイル内のすべての **YOUR-WAR.war** を、実際のアプリケーション WAR の名前 (例: **simple-webapp-example.war**) で更新します。



注記

生成される CLI スクリプトの 2 番目のステートメントの末尾には、**)** がありません。

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

不足している **)** を追加する必要があります。

c. 管理 CLI を使用して **keycloak-saml-subsystem.cli** スクリプトを実行して、JBoss EAP を設定します。

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

4. アプリケーションをデプロイします。

```
$ mvn wildfly:deploy
```

検証

1. ブラウザーで、アプリケーションの URL に移動します。たとえば、**http://localhost:8080/simple-webapp-example/secured** です。
Red Hat build of Keycloak のログインページにリダイレクトされます。

2. Red Hat build of Keycloak で定義したユーザーの認証情報を使用してログインできます。

これで、アプリケーションが SAML を使用して保護されました。

関連情報

- SAML を使用してアプリケーションを保護するための Keycloak SAML アダプター機能パック

第3章 OIDC 使用時のサーブレットから JAKARTA ENTERPRISE BEAN へのアイデンティティーの伝播

OpenID Connect (OIDC) プロバイダーから取得したセキュリティーアイデンティティーは、次の 2 つの方法でサーブレットから Jakarta Enterprise Beans に伝播できます。

- 同じ仮想セキュリティードメインを使用して、サーブレットと Jakarta Enterprise Beans の両方を保護する。
- サーブレットに関連付けられた仮想セキュリティードメインから、Jakarta Enterprise Beans を保護するセキュリティードメインにアイデンティティーを伝播する。

3.1. OIDC 使用時の JAKARTA ENTERPRISE BEANS へのアイデンティティーの伝播

OpenID Connect (OIDC) を使用してアプリケーションを保護すると、**elytron-oidc-client** サブシステムが仮想セキュリティードメインを自動的に作成します。OIDC プロバイダーから取得した仮想セキュリティードメイン内のセキュリティーアイデンティティーを、アプリケーションが呼び出す Jakarta Enterprise Beans に伝播できます。

次の表は、使用するセキュリティードメインとアプリケーションのデプロイ方法に応じた必要な設定を示しています。

Jakarta Enterprise Beans を保護するために使用するセキュリティードメイン	サーブレットと Jakarta Enterprise Beans が同じ WAR または EAR 内にある	サーブレットと Jakarta Enterprise Beans が異なる WAR または EAR 内にある
仮想セキュリティードメイン	必要な設定はありません。 仮想セキュリティードメインはセキュリティーアイデンティティーを Jakarta Enterprise Beans に自動的にアウトフローします。ただし、Jakarta Enterprise Beans にセキュリティードメイン設定が明示的に指定されている場合を除きます。	次のように設定します。 ● **virtual-security-domain** リソースを作成します。 ● **virtual-security-domain** リソースの名前を参照する **@SecurityDomain** アノテーションを Jakarta Enterprise Beans に追加します。 詳細は、仮想セキュリティードメインを使用した Jakarta Enterprise Beans アプリケーションの保護 を参照してください。

Jakarta Enterprise Beans を保護するために使用するセキュリティードメイン	サーブレットと Jakarta Enterprise Beans が同じ WAR または EAR 内にある	サーブレットと Jakarta Enterprise Beans が異なる WAR または EAR 内にある
別のセキュリティードメイン	セキュリティーアイデンティティーを仮想セキュリティードメインから別のセキュリティードメインにアウトフローするには、次のリソースを設定する必要があります。 • **virtual-security-domain**: 仮想セキュリティードメインによって確立されたセキュリティーアイデンティティーが、他のセキュリティードメインに自動的にアウトフローするように指定します。 • **security-domain**: 設定した仮想セキュリティードメインによって確立されたセキュリティーアイデンティティーを信頼するように指定します。 詳細は、仮想セキュリティードメインからセキュリティードメインへのアイデンティティーの伝播 を参照してください。	

3.2. 仮想セキュリティードメインを使用した JAKARTA ENTERPRISE BEANS アプリケーションの保護

elytron-oidc-client サブシステムによって作成された仮想セキュリティードメインを使用すると、Jakarta Enterprise Beans がその呼び出し元のサーブレットと同じデプロイメントに配置されている場合でも、異なるデプロイメントに配置されている場合でも、Jakarta Enterprise Beans を保護できます。

Jakarta Enterprise Beans がその呼び出し元のサーブレットと同じデプロイメントに配置されている場合、サーブレットから Jakarta Enterprise Beans にセキュリティーアイデンティティーをアウトフローするための設定は必要ありません。

サーブレットから別のデプロイメントにある Jakarta Enterprise Beans にセキュリティーアイデンティティーをアウトフローするには、この手順のステップを実行します。

前提条件

- OpenID Connect (OIDC) プロバイダーを使用して、Jakarta Enterprise Beans の呼び出し元のアプリケーションを保護している。
- 保護対象の Jakarta Enterprise Beans を作成している。

手順

1. OIDC で保護されたサーブレットを含む WAR、または OIDC で保護されたサブデプロイメントを含む EAR を参照する **virtual-security-domain** リソースを作成します。

   構文

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

   例:

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

2. アプリケーションの保護に使用する仮想セキュリティードメインリソースを参照する **org.jboss.ejb3.annotation.SecurityDomain** アノテーションを Jakarta Enterprise Beans アプリケーションに追加します。

   構文

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

   例:

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

     @Resource
     private SessionContext context;

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

   }
   ```

   OIDC で保護されたサーブレットからこの Jakarta Enterprise Beans を呼び出す場合、**whoAmI()** によって返されるプリンシパルは、サーブレットが OIDC プロバイダーから取得したプリンシパルと一致します。

3. Jakarta Enterprise Beans をデプロイします。

   例:

   ```
   $ mvn wildfly:deploy
   ```

関連情報

- **virtual-security-domain** 属性

3.3. 仮想セキュリティードメインからセキュリティードメインへのアイデンティティーの伝播

OpenID Connect (OIDC) プロバイダーから取得したセキュリティーアイデンティティーを、仮想セキュリティードメインから別のセキュリティードメインに伝播できます。仮想セキュリティードメインによってではなく、アイデンティティーの伝播先のセキュリティードメインによってセキュリティーアイデンティティーのロールを決定する場合は、この手順を行うことをお勧めします。

次の手順のステップは、Jakarta Enterprise Beans を呼び出すサーブレットと Jakarta Enterprise Beans が同じデプロイメントにある場合と、それらが別のデプロイメントにある場合の両方に適用されます。

前提条件

- OIDC プロバイダーを使用して、Jakarta Enterprise Beans の呼び出し元のアプリケーションを保護している。
- 保護対象の Jakarta Enterprise Beans を作成している。
- Jakarta Enterprise Beans をセキュリティードメインで保護している。

手順

1. OIDC で保護されたサーブレットを含む WAR、または OIDC で保護されたサブデプロイメントを含む EAR を参照する **virtual-security-domain** リソースを作成します。

   構文

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

   例:

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

2. **virtual-security-domain** を信頼するように Jakarta Enterprise Beans のセキュリティードメイン設定を更新します。

   構文

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

   例:

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

3. サーバーをリロードします。

   ```
   reload
   ```

4. Jakarta Enterprise Beans をデプロイします。

   例:

   ```
   $ mvn wildfly:deploy
   ```

関連情報

- **security-domain** 属性
- **virtual-security-domain** 属性

第4章 OPENID プロバイダーによる JBOSS EAP 管理コンソールの保護

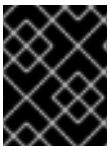
OIDC を使用して、Red Hat build of Keycloak などの外部アイデンティティプロバイダーで JBoss EAP 管理コンソールを保護できます。外部アイデンティティープロバイダーを使用すると、認証をそのアイデンティティープロバイダーに委譲できます。

OIDC を使用して JBoss EAP 管理コンソールを保護するには、以下の手順に従います。

- JBoss EAP 管理コンソールを保護するための Red Hat build of Keycloak の設定
- OpenID Connect を使用した JBoss EAP 管理コンソールの保護

4.1. OIDC を使用した JBOSS EAP 管理コンソールの保護

Red Hat build of Keycloak などの OIDC プロバイダーと **elytron-oidc-client** サブシステムを設定することにより、OpenID Connect (OIDC) を使用して JBoss EAP 管理コンソールを保護できます。

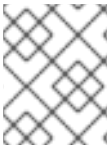


重要

OIDC を使用した、マネージドドメインとして実行されている JBoss EAP の管理コンソールの保護はサポートされていません。

OIDC を使用した JBoss EAP 管理コンソールのセキュリティーの仕組みは次のとおりです。

- **elytron-oidc-client** サブシステムで **secure-server** リソースを設定すると、JBoss EAP 管理コンソールがログインのために OIDC プロバイダーのログインページにリダイレクトするようになります。
- さらに、JBoss EAP は **secure-deployment** リソース設定を使用して、ベアラートークン認証で管理インターフェイスを保護します。



注記

OIDC は、ブラウザーからの Web アプリケーションへのアクセスに依存します。したがって、JBoss EAP 管理 CLI は OIDC では保護できません。

RBAC のサポート

OIDC プロバイダーでロールを設定および割り当てて、JBoss EAP 管理コンソールにロールベースのアクセス制御 (RBAC) を実装できます。JBoss EAP は、JBoss EAP の RBAC 設定で定義されている RBAC のユーザーロールを追加または除外します。RBAC の詳細は、JBoss EAP 7.4 **セキュリティーアーキテクチャー** ガイドの ロールベースのアクセス制御 を参照してください。

関連情報

- JBoss EAP 管理コンソールを保護するための Red Hat build of Keycloak の設定
- OpenID Connect を使用した JBoss EAP 管理コンソールの保護

4.2. JBOSS EAP 管理コンソールを保護するための RED HAT BUILD OF KEYCLOAK の設定

OpenID Connect (OIDC) プロバイダーで必要なユーザー、ロール、クライアントを設定して、JBoss EAP 管理コンソールを保護します。

OIDC を使用して管理コンソールを保護するには、2 つのクライアントが必要です。設定する必要があるクライアントは次のとおりです。

- 標準フロー用に設定したクライアント。
- ベアラー専用クライアントとして設定したクライアント。

次の手順は、テスト目的で OIDC を使用して JBoss EAP 管理コンソールの保護を開始するために必要な最小限の手順を示しています。詳細な設定については、Red Hat build of Keycloak のドキュメント を参照してください。

前提条件

- Red Hat build of Keycloak への管理者アクセス権を持っている。
- Red Hat build of Keycloak が実行中である。

手順

1. Red Hat build of Keycloak 管理コンソールを使用して Red Hat build of Keycloak にレルムを作成します (**example_jboss_infra** など)。このレルムを使用して、必要なユーザー、ロール、およびクライアントを作成します。
   詳細は、レルムの作成 を参照してください。
2. ユーザーを作成します。たとえば、**user1** です。
   詳細は、ユーザーの作成 を参照してください。
3. ユーザーのパスワードを作成します。たとえば、**passwordUser1** です。
   詳細は、ユーザーのパスワードの設定 を参照してください。
4. ロールを作成します。たとえば、**Administrator** です。
   JBoss EAP でロールベースのアクセス制御 (RBAC) を有効にするには、この名前が標準の RBAC ロールの 1 つ (**Administrator** など) である必要があります。JBoss EAP の RBAC の詳細は、JBoss EAP 7.4 **セキュリティーアーキテクチャー** ガイドの ロールベースのアクセス制御 を参照してください。

   Red Hat build of Keycloak でのロールの作成の詳細は、レルムロールの作成 を参照してください。
5. ユーザーにロールを割り当てます。
   詳細は、ロールマッピングの割り当て を参照してください。
6. OpenID Connect クライアント (例: **jboss-console)** を作成します。
   - 以下に示す機能の設定値のチェックボックスがオンになっていることを確認します。
     - Standard flow
     - Direct access grants
   - **Login settings** ページで少なくとも次の属性を設定します。
     - **Valid Redirect URIs** を管理コンソール URI に設定します。たとえば、http://localhost:9990 です。

- **Web Origins** を管理コンソール URI に設定します。たとえば、 http://localhost:9990 です。

7. 別の OpenID Connect クライアント (例: **jboss-management)** をベアラー専用クライアントとして作成します。
   - 機能の設定で、次のオプションのチェックボックスをオフにします。
     - Standard flow
     - Direct access grants
   - **Login settings** ページのフィールドを指定する必要はありません。

これで、定義したクライアントを使用して JBoss EAP 管理コンソールを保護できるようになりました。詳細は、OpenID Connect を使用した JBoss EAP 管理コンソールの保護 参照してください。

関連情報

- OIDC を使用した JBoss EAP 管理コンソールの保護

4.3. OPENID CONNECT を使用した JBOSS EAP 管理コンソールの保護

OpenID Connect (OIDC) を使用して JBoss EAP 管理コンソールを保護すると、JBoss EAP は、ユーザーが管理コンソールにログインできるように、OIDC プロバイダーにリダイレクトします。

前提条件

- OIDC プロバイダーで必要なクライアントを設定している。
  詳細は、JBoss EAP 管理コンソールを保護するための Red Hat build of Keycloak の設定 を参照してください。

手順

1. **elytron-oidc-client** サブシステムで OIDC プロバイダーを設定します。

   構文

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

   例:

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

2. 管理インターフェイスを保護するために、**wildfly-management** という名前の **secure-deployment** リソースを作成します。

   構文

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

例:

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

3. オプション: 次のコマンドを使用して、ロールベースのアクセス制御 (RBAC) を有効にできます。

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

4. **jboss-console** クライアントを参照する **wildfly-console** という名前の **secure-server** リソースを作成します。

構文

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

例:

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

重要

JBoss EAP 管理コンソールには、**wildfly-console** という特定の名前の **secure-server** リソースが必要です。

検証

1. 管理コンソールにアクセスします。デフォルトでは、管理コンソールには http://localhost:9990 からアクセスできます。
OIDC プロバイダーにリダイレクトされます。

2. OIDC プロバイダーで作成したユーザーの認証情報を使用してログインします。

これで、JBoss EAP 管理コンソールが OIDC で保護されました。

関連情報

- OIDC を使用した JBoss EAP 管理コンソールの保護
- **elytron-oidc-client** サブシステム属性

第5章 参照

5.1. ELYTRON-OIDC-CLIENT サブシステム属性

elytron-oidc-client サブシステムは、その動作を設定するための属性を提供します。

表5.1 elytron-oidc-client サブシステム属性

属性	説明
provider	OpenID Connect プロバイダーの設定。
secure-deployment	OpenID Connect プロバイダーによって保護されたデプロイメント。
realm	Red Hat build of Keycloak レルムの設定。これは便宜上提供されています。keycloak クライアントアダプターで設定をコピーして、ここで使用できます。代わりに **provider** を使用することを推奨します。

次の目的で、3 つの **elytron-oidc-client** 属性を使用します。

- **provider**: OpenID Connect プロバイダーを設定します。詳細は、**provider 属性** を参照してください。
- **secure-deployment**: OpenID Connect によって保護されたデプロイメントを設定します。詳細は、**secure-deployment 属性** を参照してください。
- **realm**: Red Hat build of Keycloak を設定するための属性です。詳細は、 **realm 属性** を参照してください。**realm** の使用は推奨しません。便宜上提供されています。keycloak クライアントアダプターで設定をコピーして、ここで使用できます。代わりに、**provider** 属性を使用することを推奨します。

表5.2 provider 属性

属性	デフォルト値	説明
allow-any-hostname	**false**	値を **true** に設定すると、OpenID プロバイダーと通信するときにホスト名の検証がスキップされます。これは、テスト時に役立ちます。実稼働環境ではこれを **true** に設定しないでください。
always-refresh-token		**true** に設定すると、アプリケーションが Web リクエストを受信するたびに、サブシステムがトークンを更新し、新しいリクエストを OpenID プロバイダーに送信して新しいアクセストークンを取得します。

属性	デフォルト値	説明
auth-server-url		Red Hat build of Keycloak レルム認証サーバーのベース URL。この属性を使用する場合は、**realm** 属性も定義する必要があります。 または、**provider-url** 属性を使用して、ベース URL とレルムの両方を 1 つの属性で提供することもできます。
autodetect-bearer-only	**false**	bearer-only リクエストを自動検出するかどうかを設定します。 bearer-only リクエストを受信し、**autodetect-bearer-only** が **true** に設定されている場合、アプリケーションはブラウザーログインに組み込まれません。 この属性を使用すると、**X-Requested-With**、**SOAPAction**、**Accept** などのヘッダーに基づいて Simple Object Access Protocol (SOAP) または REST クライアントを自動的に検出できます。
client-id		OpenID プロバイダーに登録された JBoss EAP の client-id。
client-key-password		**client-keystore** を指定する場合は、この属性にそのパスワードを指定します。
client-keystore		アプリケーションが HTTPS を介して OpenID プロバイダーと通信する場合は、この属性でクライアントキーストアへのパスを設定します。
client-keystore-password		**client keystore** を指定する場合は、この属性でそれにアクセスするためのパスワードを指定します。
confidential-port	**8443**	OpenID プロバイダーが使用する機密ポート (SSL/TLS) を指定します。
connection-pool-size		OpenID プロバイダーと通信するときに使用する接続プールのサイズを指定します。
connection-timeout-millis	**-1L**	リモートホストとの接続を確立するためのタイムアウトをミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。**-1L** は、値が未定義であることを示します。これがデフォルトです。
connection-ttl-millis	**-1L**	接続を維持する時間をミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。**-1L** は、値が未定義であることを示します。これがデフォルトです。

属性	デフォルト値	説明
cors-allowed-headers		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Allow-Headers** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-allowed-methods		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより Access-Control-Allow-Methods ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-exposed-headers		CORS が有効な場合は、Access-Control-Expose-Headers ヘッダーの値を設定します。これはコンマ区切りの文字列である必要があります。これは最適です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-max-age		Cross-Origin Resource Sharing (CORS) Max-Age ヘッダーの値を設定します。値は **-1L** から **2147483647L** の間です。この属性は、**enable-cors** が **true** に設定されている場合にのみ有効になります。
disable-trust-manager		HTTPS 経由で OpenID プロバイダーと通信するときにトラストマネージャーを使用するかどうかを指定します。
enable-cors	**false**	Red Hat build of Keycloak の Cross-Origin Resource Sharing (CORS) のサポートを有効にします。
expose-token	**false**	**true** に設定すると、認証されたブラウザークライアントは、Javascript HTTP 呼び出し、URL **root/k_query_bearer_token** を介して署名付きアクセストークンを取得できます。これは任意です。この属性は Red Hat build of Keycloak 固有のものです。
ignore-oauth-query-parameter	**false**	**access_token** のクエリーパラメーター解析を無効にします。
principal-attribute		アイデンティティーのプリンシパルとして使用する ID トークンのクレーム値を指定します。
provider-url		OpenID プロバイダーの URL を指定します。
proxy-url		HTTP プロキシーを使用する場合は、その URL を指定します。
realm-public-key		レルムの公開鍵を指定します。

属性	デフォルト値	説明
register-node-at-startup	**false**	**true** に設定すると、登録リクエストが Red Hat build of Keycloak に送信されます。この属性は、アプリケーションがクラスター化されている場合にのみ役立ちます。
register-node-period		ノードを再登録する頻度を指定します。
socket-timeout-millis		データを待機するソケットのタイムアウトをミリ秒単位で指定します。
ssl-required	**external**	OpenID プロバイダーとの通信を HTTPS 経由で行うかどうかを指定します。値は次のいずれかになります。 ● **all** - すべての通信は HTTPS を介して行われます。 ● **external** - 外部クライアントとの通信のみが HTTP を介して行われます。 ● **none** - HTTP は使用されません。
token-signature-algorithm	**RS256**	OpenID プロバイダーが使用するトークン署名アルゴリズムを指定します。サポートされているアルゴリズムは次のとおりです。 ● **RS256** ● **RS384** ● **RS512** ● **ES256** ● **ES384** ● **ES512**
token-store		auth-session データの Cookie またはセッションストレージを指定します。
truststore		クライアント HTTPS 要求に使用されるトラストストアを指定します。
truststore-password		トラストストアのパスワードを指定します。
verify-token-audience	**false**	**true** に設定されている場合、bearer-only の認証中に、トークンにこのクライアント名 (**resource**) がオーディエンスとして含まれているかどうかを確認します。

表5.3 secure-deployment 属性

属性	デフォルト値	説明
allow-any-hostname	**false**	値を **true** に設定すると、OpenID プロバイダーと通信するときにホスト名の検証がスキップされます。これは、テスト時に役立ちます。これを実稼働環境で **ture** に設定しないでください。
always-refresh-token		**true** に設定すると、JBoss EAP はすべての Web 要求でトークンを更新します。
auth-server-url		Red Hat build of Keycloak レルム認可サーバーのベース URL。代わりに **provider-url** 属性を使用することもできます。
autodetect-bearer-only	**false**	bearer-only リクエストを自動検出するかどうかを設定します。bearer-only リクエストを受信し、**autodetect-bearer-only** が **true** に設定されている場合、アプリケーションはブラウザーログインに組み込まれません。
bearer-only	**false**	Bearer トークン認証でアプリケーションを保護するには、これを **true** に設定します。Bearer トークン認証が有効になっている場合、ユーザーはログインするために OpenID プロバイダーにリダイレクトされません。代わりに、**elytron-oidc-client** サブシステムがユーザーの bearer トークンを検証しようとします。
client-id		OpenID プロバイダーに登録されているクライアントの一意の識別子。
client-key-password		**client-keystore** を指定する場合は、この属性にそのパスワードを指定します。
client-keystore		アプリケーションが HTTPS を介して OpenID プロバイダーと通信する場合は、この属性でクライアントキーストアへのパスを設定します。

属性	デフォルト値	説明
client-keystore-password		**client keystore** を指定する場合は、この属性でそれにアクセスするためのパスワードを指定します。
confidential-port	**8443**	OpenID プロバイダーが使用する機密ポート (SSL/TLS) を指定します。
connection-pool-size		OpenID プロバイダーと通信するときに使用する接続プールのサイズを指定します。
connection-timeout-millis	**-1L**	リモートホストとの接続を確立するためのタイムアウトをミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。**-1L** は、値が未定義であることを示します。これがデフォルトです。
connection-ttl-millis	**-1L**	接続を維持する時間をミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。-1L は、値が未定義であることを示します。これがデフォルトです。
cors-allowed-headers		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Allow-Headers** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-allowed-methods		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Allow-Methods** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。

属性	デフォルト値	説明
cors-exposed-headers		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Expose-Headers** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-max-age		Cross-Origin Resource Sharing (CORS) Max-Age ヘッダーの値を設定します。値は **-1L** から **2147483647L** の間です。この属性は、**enable-cors** が **true** に設定されている場合にのみ有効になります。
credential		OpenID プロバイダーとの通信に使用する認証情報を指定します。
disable-trust-manager		HTTPS 経由で OpenID プロバイダーと通信するときにトラストマネージャーを使用するかどうかを指定します。
enable-cors	**false**	Red Hat build of Keycloak の Cross-Origin Resource Sharing (CORS) のサポートを有効にします。
enable-basic-auth	**false**	Basic 認証を有効にして、bearer トークンの取得に使用する認証情報を指定します。
expose-token	**false**	**true** に設定すると、認証されたブラウザークライアントは、Javascript HTTP 呼び出し、URL **root/k_query_bearer_token** を介して署名付きアクセストークンを取得できます。これは任意です。この属性は Red Hat build of Keycloak 固有のものです。
ignore-oauth-query-parameter	**false**	**access_token** のクエリーパラメーター解析を無効にします。

属性	デフォルト値	説明
min-time-between-jwks-requests	サブシステムが不明な公開鍵によって署名されたトークンを検出した場合、JBoss EAP は **elytron-oidc-client** サーバーから新しい公開鍵をダウンロードしようとします。この属性は、JBoss EAP が次のダウンロードを試行するまで待機する間隔を秒単位で指定します。値は **-1L** から **2147483647L** までの範囲にすることができます。	principal-attribute
	アイデンティティーのプリンシパルとして使用する ID トークンのクレーム値を指定します。	provider
	OpenID プロバイダーを指定します。	provider-url
	OpenID プロバイダーの URL を指定します。	proxy-url
	HTTP プロキシーを使用する場合は、その URL を指定します。	public-client
false	**true** に設定すると、OpenID プロバイダーとの通信時にクライアント認証情報は送信されません。これは任意です。	realm
	Red Hat build of Keycloak での接続に使用するレルム。	realm-public-key
	OpenID プロバイダーの公開鍵を PEM 形式で指定します。	redirect-rewrite-rule
	リダイレクト URI に適用する書き換えルールを指定します。	register-node-at-startup
false	**true** に設定すると、登録リクエストが Red Hat build of Keycloak に送信されます。この属性は、アプリケーションがクラスター化されている場合にのみ役立ちます。	register-node-period
	ノードを再登録する頻度を秒単位で指定します。	resource

属性	デフォルト値	説明
	OIDC で保護するアプリケーションの名前を指定します。または、**client-id** を指定することもできます。	socket-timeout-millis
	データを待機するソケットのタイムアウトをミリ秒単位で指定します。	ssl-required
external	OpenID プロバイダーとの通信を HTTPS 経由で行うかどうかを指定します。値は次のいずれかになります。 ● **all** - すべての通信は HTTPS を介して行われます。 ● **external** - 外部クライアントとの通信のみが HTTP を介して行われます。 ● **none** - HTTP は使用されません。	token-minimum-time-to-live
	現在のトークンが期限切れになるか、秒単位で設定した時間内に期限切れになる場合、アダプターはトークンを更新します。	token-signature-algorithm
RS256	OpenID プロバイダーが使用するトークン署名アルゴリズムを指定します。サポートされているアルゴリズムは次のとおりです。 ● **RS256** ● **RS384** ● **RS512** ● **ES256** ● **ES384** ● **ES512**	token-store
	auth-session データの Cookie またはセッションストレージを指定します。	truststore

属性	デフォルト値	説明
	アダプタークライアントの HTTPS 要求に使用されるトラストストアを指定します。	truststore-password
	トラストストアのパスワードを指定します。	turn-off-change-session-id-on-login
false	ログインに成功すると、デフォルトでセッション ID が変更されます。これをオフにするには、値を **true** に設定します。	use-resource-role-mappings
false	トークンから取得したリソースレベルの権限を使用します。	verify-token-audience

表5.4 secure-server 属性

属性	デフォルト値	説明
adapter-state-cookie-path		この属性は、設定されている場合、サブシステムによって設定される Cookie で使用されるパスを定義します。設定されていない場合、**""** がパスとして使用されます。
allow-any-hostname	**false**	値を **true** に設定すると、OpenID プロバイダーと通信するときにホスト名の検証がスキップされます。これは、テスト時に役立ちます。実稼働環境ではこれを **true** に設定しないでください。
always-refresh-token		**true** に設定すると、アプリケーションが Web リクエストを受信するたびに、サブシステムがトークンを更新し、新しいリクエストを OpenID プロバイダーに送信して新しいアクセストークンを取得します。
auth-server-url-for-backend-requests		ロードバランサーやリバースプロキシーを経由せずに OpenID プロバイダーを直接呼び出すバックエンドリクエストにのみ使用する URL を指定します。

属性	デフォルト値	説明
auth-server-url		Red Hat build of Keycloak レルム認可サーバーのベース URL。代わりに **provider-url** 属性を使用することもできます。
autodetect-bearer-only	**false**	bearer-only リクエストを自動検出するかどうかを設定します。 bearer-only リクエストを受信し、**autodetect-bearer-only** が **true** に設定されている場合、アプリケーションはブラウザーログインに組み込まれません。 この属性を使用すると、**X-Requested-With**、**SOAPAction**、**Accept** などのヘッダーに基づいて Simple Object Access Protocol (SOAP) または REST クライアントを自動的に検出できます。
bearer-only	**false**	Bearer トークン認証でアプリケーションを保護するには、これを **true** に設定します。 Bearer トークン認証が有効になっている場合、ユーザーはログインするために OpenID プロバイダーにリダイレクトされません。代わりに、**elytron-oidc-client** サブシステムがユーザーの bearer トークンを検証しようとします。
client-id		OpenID プロバイダーに登録されているクライアントの一意の識別子。
client-key-password		**client-keystore** を指定する場合は、この属性にそのパスワードを指定します。
client-keystore-password		**client keystore** を指定する場合は、この属性でそれにアクセスするためのパスワードを指定します。

属性	デフォルト値	説明
client-keystore		HTTPS で OpenID プロバイダーと通信する場合は、この属性にクライアントキーストアへのパスを設定します。
confidential-port	**8443**	OpenID プロバイダーが使用する機密ポート (SSL/TLS) を指定します。
connection-pool-size		OpenID プロバイダーと通信するときに使用する接続プールのサイズを指定します。
connection-timeout-millis	**-1L**	リモートホストとの接続を確立するためのタイムアウトをミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。**-1L** は、値が未定義であることを示します。これがデフォルトです。
connection-ttl-millis	**-1L**	接続を維持する時間をミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。-1L は、値が未定義であることを示します。これがデフォルトです。
cors-allowed-headers		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Allow-Headers** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-allowed-methods		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Allow-Methods** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。

属性	デフォルト値	説明
cors-exposed-headers		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Expose-Headers** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-max-age		Cross-Origin Resource Sharing (CORS) Max-Age ヘッダーの値を設定します。値は **-1L** から **2147483647L** の間です。この属性は、**enable-cors** が **true** に設定されている場合にのみ有効になります。
credential		OpenID プロバイダーとの通信に使用する認証情報を指定します。
disable-trust-manager		HTTPS 経由で OpenID プロバイダーと通信するときにトラストマネージャーを使用するかどうかを指定します。
enable-basic-auth	**false**	Basic 認証を有効にして、bearer トークンの取得に使用する認証情報を指定します。
enable-cors	**false**	Red Hat build of Keycloak の Cross-Origin Resource Sharing (CORS) のサポートを有効にします。
expose-token	**false**	**true** に設定すると、認証されたブラウザークライアントは、Javascript HTTP 呼び出し、URL **root/k_query_bearer_token** を介して署名付きアクセストークンを取得できます。これは任意です。この属性は Red Hat build of Keycloak 固有のものです。
ignore-oauth-query-parameter	**false**	**access_token** のクエリーパラメーター解析を無効にします。

属性	デフォルト値	説明
min-time-between-jwks-requests		サブシステムが不明な公開鍵によって署名されたトークンを検出した場合、JBoss EAP は **elytron-oidc-client** サーバーから新しい公開鍵をダウンロードしようとします。ただし、JBoss EAP は、この属性に設定した値 (秒単位) 未満ですでに新しい公開鍵を試行している場合、新しい公開鍵をダウンロードしようとはしません。値は **-1L** から **2147483647L** の間です。
principal-attribute		アイデンティティーのプリンシパルとして使用する ID トークンのクレーム値を指定します。
principal-attribute		アイデンティティーのプリンシパルとして使用する ID トークンのクレーム値を指定します。
provider		OpenID プロバイダーを指定します。
provider-url		OpenID プロバイダーの URL を指定します。
proxy-url		HTTP プロキシーを使用する場合は、その URL を指定します。
public-client	**false**	**true** に設定すると、OpenID プロバイダーとの通信時にクライアント認証情報は送信されません。これは任意です。
public-key-cache-ttl		新しい公開鍵を取得する 2 つのリクエスト間の最大間隔 (秒単位)。
realm-public-key		OpenID プロバイダーの公開鍵を PEM 形式で指定します。
realm		Red Hat build of Keycloak での接続に使用するレルム。
redirect-rewrite-rule		リダイレクト URI に適用する書き換えルールを指定します。

属性	デフォルト値	説明
register-node-at-startup	**false**	**true** に設定すると、登録リクエストが Red Hat build of Keycloak に送信されます。この属性は、アプリケーションがクラスター化されている場合にのみ役立ちます。
register-node-period		ノードを再登録する頻度を秒単位で指定します。
resource		OIDC で保護するアプリケーションの名前を指定します。または、**client-id** を指定することもできます。
socket-timeout-millis		データを待機するソケットのタイムアウトをミリ秒単位で指定します。
ssl-required	**external**	OpenID プロバイダーとの通信を HTTPS 経由で行うかどうかを指定します。値は次のいずれかになります。 ● **all** - すべての通信は HTTPS を介して行われます。 ● **external** - 外部クライアントとの通信のみが HTTP を介して行われます。 ● **none** - HTTP は使用されません。
token-minimum-time-to-live		現在のトークンが期限切れになるか、秒単位で設定した時間内に期限切れになる場合、アダプターはトークンを更新します。

属性	デフォルト値	説明
token-signature-algorithm	**RS256**	OpenID プロバイダーが使用するトークン署名アルゴリズムを指定します。サポートされているアルゴリズムは次のとおりです。 • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**
token-store		auth-session データの Cookie またはセッションストレージを指定します。
truststore-password		トラストストアのパスワードを指定します。
truststore		アダプタークライアントの HTTPS 要求に使用されるトラストストアを指定します。
turn-off-change-session-id-on-login	**false**	ログインに成功すると、デフォルトでセッション ID が変更されます。これをオフにするには、値を **true** に設定します。
use-resource-role-mappings	**false**	トークンから取得したリソースレベルの権限を使用します。
verify-token-audience	**false**	**true** に設定すると、bearer-only の認証中に、アダプターはトークンにこのクライアント名 (**resource**) がオーディエンスとして含まれているかどうかを確認します。

表5.5 realm 属性

属性	デフォルト値	説明
allow-any-hostname	**false**	値を **true** に設定すると、OpenID プロバイダーと通信するときにホスト名の検証がスキップされます。これは、テスト時に役立ちます。これを実稼働環境で **ture** に設定しないでください。
always-refresh-token		**true** に設定すると、アプリケーションが Web リクエストを受信するたびに、サブシステムがトークンを更新し、新しいリクエストを OpenID プロバイダーに送信して新しいアクセストークンを取得します。
auth-server-url		Red Hat build of Keycloak レルム認可サーバーのベース URL。代わりに **provider-url** 属性を使用することもできます。
autodetect-bearer-only	**false**	bearer-only リクエストを自動検出するかどうかを設定します。bearer-only リクエストを受信し、**autodetect-bearer-only** が **true** に設定されている場合、アプリケーションはブラウザーログインに組み込まれません。
client-key-password		**client-keystore** を指定する場合は、この属性にそのパスワードを指定します。
client-keystore		アプリケーションが HTTPS を介して OpenID プロバイダーと通信する場合は、この属性でクライアントキーストアへのパスを設定します。
client-keystore-password		**client keystore** を指定する場合は、この属性でそれにアクセスするためのパスワードを指定します。
confidential-port	**8443**	Red Hat build of Keycloak が使用する機密ポート (SSL/TLS) を指定します。

属性	デフォルト値	説明
connection-pool-size		Red Hat build of Keycloak と通信するときに使用する接続プールのサイズを指定します。
connection-timeout-millis	**-1L**	リモートホストとの接続を確立するためのタイムアウトをミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。**-1L** は、値が未定義であることを示します。これがデフォルトです。
connection-ttl-millis	**-1L**	接続を維持する時間をミリ秒単位で指定します。最小値は **-1L**、最大値は **2147483647L** です。**-1L** は、値が未定義であることを示します。これがデフォルトです。
cors-allowed-headers		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより Access-Control-Allow-Headers ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-allowed-methods		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Allow-Methods header** の値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。
cors-exposed-headers		Cross-Origin Resource Sharing (CORS) が有効になっている場合、これにより **Access-Control-Expose-Headers** ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。これは任意です。設定されていない場合、このヘッダーは CORS 応答で返されません。

属性	デフォルト値	説明
cors-max-age		Cross-Origin Resource Sharing (CORS) Max-Age ヘッダーの値を設定します。値は **-1L** から **2147483647L** の間です。この属性は、**enable-cors** が **true** に設定されている場合にのみ有効になります。
disable-trust-manager		HTTPS 経由で OpenID プロバイダーと通信するときにトラストマネージャーを使用するかどうかを指定します。
enable-cors	**false**	Red Hat build of Keycloak の Cross-Origin Resource Sharing (CORS) のサポートを有効にします。
expose-token	**false**	**true** に設定すると、認証されたブラウザークライアントは、Javascript HTTP 呼び出し、URL **root/k_query_bearer_token** を介して署名付きアクセストークンを取得できます。これは任意です。
ignore-oauth-query-parameter	**false**	**access_token** のクエリーパラメーター解析を無効にします。
principal-attribute		アイデンティティーのプリンシパルとして使用する ID トークンのクレーム値を指定します。
provider-url		OpenID プロバイダーの URL を指定します。
proxy-url		HTTP プロキシーを使用する場合は、その URL を指定します。
realm-public-key		レルムの公開鍵を指定します。
register-node-at-startup	**false**	**true** に設定すると、登録リクエストが Red Hat build of Keycloak に送信されます。この属性は、アプリケーションがクラスター化されている場合にのみ役立ちます。
register-node-period		ノードを再登録する頻度を指定します。

属性	デフォルト値	説明
socket-timeout-millis		データを待機するソケットのタイムアウトをミリ秒単位で指定します。
ssl-required	**external**	OpenID プロバイダーとの通信を HTTPS 経由で行うかどうかを指定します。値は次のいずれかになります。 ● **all** - すべての通信は HTTPS を介して行われます。 ● **external** - 外部クライアントとの通信のみが HTTP を介して行われます。 ● **none** - HTTP は使用されません。
token-signature-algorithm	**RS256**	OpenID プロバイダーが使用するトークン署名アルゴリズムを指定します。サポートされているアルゴリズムは次のとおりです。 ● **RS256** ● **RS384** ● **RS512** ● **ES256** ● **ES384** ● **ES512**
token-store		auth-session データの Cookie またはセッションストレージを指定します。
truststore		クライアント HTTPS 要求に使用されるトラストストアを指定します。
truststore-password		トラストストアのパスワードを指定します。

属性	デフォルト値	説明
verify-token-audience	**false**	**true** に設定すると、bearer-only の認証中に、アダプターはトークンにこのクライアント名 (リソース) がオーディエンスとして含まれているかどうかを確認します。

関連情報

- JBoss EAP の OpenID Connect によるアプリケーションの保護
- OpenID Connect を使用した Web アプリケーションの保護

5.2. SECURITY-DOMAIN 属性

属性を設定することにより、**security-domain** を設定できます。

属性	説明
default-realm	このセキュリティードメインに含まれるデフォルトのレルム。
evidence-decoder	このドメインで使用される EvidenceDecoder への参照。
outflow-anonymous	この属性は、セキュリティードメインへのアウトフローが不可能な場合に、匿名アイデンティティーを使用するかどうかを指定します。アウトフローが不可能な状況は、次の場合に発生します。 ● アウトフロー先のドメインがこのドメインを信頼していない。 ● ドメインにアウトフローするアイデンティティーが当該ドメインに存在しない。 匿名アイデンティティーをアウトフローすると、そのドメインに対して以前に確立されたアイデンティティーがすべて消去されます。
outflow-security-domains	このドメインのセキュリティーアイデンティティーの自動アウトフロー先となるセキュリティードメインのリスト。
permission-mapper	このドメインで使用される PermissionMapper への参照。
post-realm-principal-transformer	指定のアイデンティティー名でレルムが動作した後に適用するプリンシパルトランスフォーマーへの参照。
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への参照。

属性	説明
principal-decoder	このドメインで使用される PrincipalDecoder への参照。
realm-mapper	このドメインで使用される RealmMapper への参照。
realms	このセキュリティードメインに含まれるレルムのリスト。
role-decoder	このドメインで使用される RoleDecoder への参照。
role-mapper	このドメインで使用される RoleMapper への参照。
security-event-listener	セキュリティーイベントのリスナーへの参照。
trusted-security-domains	このセキュリティードメインによって信頼されているセキュリティードメインのリスト。
trusted-virtual-security-domains	このセキュリティードメインによって信頼されている仮想セキュリティードメインのリスト。

5.3. VIRTUAL-SECURITY-DOMAIN 属性

属性を設定することにより、**virtual-security-domain** を設定できます。

表5.6 virtual-security-domain 属性

属性	説明
outflow-anonymous	セキュリティードメインへのアウトフローが不可能な場合に、匿名アイデンティティーをアウトフローするには、この属性を **true** に設定します。アウトフローが不可能な状況は、次の場合に発生します。 ● アウトフロー先のドメインがこの仮想ドメインを信頼していない。 ● ドメインにアウトフローするアイデンティティーが当該ドメインに存在しない。 匿名アイデンティティーをアウトフローすると、そのドメインに対してすでに確立されているアイデンティティーがすべて消去されます。 デフォルト値は **false** です。
outflow-security-domains	この仮想ドメインのセキュリティーアイデンティティーの自動アウトフロー先となるセキュリティードメインのリスト。