Red Hat JBoss Enterprise Application Platform 8.0

アイデンティティーストアを使用したアプリケーションと管理インターフェイスの保護

ファイルシステム、データベース、LDAP (Lightweight Directory Access Protocol) などのアイデンティティーストア、またはカスタムアイデンティティーストアを使用して JBoss EAP 管理インターフェイスおよびデプロイされたアプリケーションをセキュアにするためのガイド

Last Updated: 2024-02-10

Red Hat JBoss Enterprise Application Platform 8.0 アイデンティティーストアを使用したアプリケーションと管理インターフェイスの保護

ファイルシステム、データベース、LDAP (Lightweight Directory Access Protocol) などのアイデンティティーストア、またはカスタムアイデンティティーストアを使用して JBoss EAP 管理インターフェイスおよびデプロイされたアプリケーションをセキュアにするためのガイド

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

ファイルシステム、データベース、LDAP (Lightweight Directory Access Protocol) などのアイデンティティーストア、またはカスタムアイデンティティーストアを使用して JBoss EAP 管理インターフェイスおよびデプロイされたアプリケーションをセキュアにするためのガイド。

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 ID ストアの設定

1.1. FILESYSTEM-REALM の作成

1.1.1. Elytron のファイルシステムレルム

ファイルシステムセキュリティーレルム **filesystem-realm** を使用すると、ファイルシステムベースのアイデンティティーストアを Elytron で使用して、ユーザーの認証情報と属性を保存できます。Elytron は、関連する認証情報と属性をファイルシステムの XML ファイルに保存します。XML ファイルの名前はアイデンティティーの名前です。複数の認証情報および属性を各アイデンティティーに関連付けることができます。

デフォルトでは、アイデンティティーは以下のようにファイルシステムに保存されます。

- Elytron は、ID が保存されるディレクトリー構造に 2 つのレベルのディレクトリーハッシュを適用します。たとえば、"user1" という名前のアイデンティティーは、**u/s/user1.xml** の場所に保存されます。
  これは、単一のディレクトリーに保存できるファイルの数とパフォーマンス上の理由で、ファイルシステムにより設定された制限を解決するために行われます。

  levels 属性を使用して、適用するディレクトリーハッシュのレベル数を設定します。

- ID 名は、ファイル名として使用される前に Base32 でエンコードされます。これは、一部のファイルシステムが大文字と小文字を区別しない場合や、ファイル名で許可される文字のセットを制限する可能性があるためです。
  encoded 属性を **false** に設定することで、エンコーディングをオフにできます。

その他の属性とそのデフォルト値の詳細は、**filesystem-realm 属性** を参照してください。

暗号化
filesystem-realm は、アイデンティティーを ID ファイルに保存する際に、クリアパスワード、ハッシュ化されたパスワード、および属性に Base64 エンコーディングを使用します。セキュリティーを強化するために、認証情報ストアに保存されている秘密鍵を使用して、クリアテキストのパスワード、ハッシュ化されたパスワード、および属性を暗号化できます。秘密鍵は、パスワードと属性を暗号化および復号するために使用されます。

整合性チェック
filesystem-realm で作成されたアイデンティティーが改ざんされないようにするために、作成時に **filesystem-realm** のキーペアを参照して、**filesystem-realm** で整合性チェックを有効にできます。

整合性チェックは、以下のように **filesystem-realm** で機能します。

- 整合性チェックを有効にして **filesystem-realm** でアイデンティティーを作成すると、Elytron はアイデンティティーファイルを作成し、その署名を生成します。

- アイデンティティーファイルが読み取られるときに (アイデンティティーの更新時や認証のためにアイデンティティーをロードするときなど)、Elytron はアイデンティティーファイルの内容を署名と照合して検証し、最後に認可された書き込み以降にファイルが改ざんされていないことを確認します。

- 関連する署名を持つ既存のアイデンティティーを更新すると、Elytron はコンテンツを更新し、元のコンテンツが検証に合格した後に新しい署名を生成します。
  検証に失敗した場合は、以下の失敗メッセージが表示されます。

  ```
  {
  ```

```
    "outcome" => "failed",
    "failure-description" => "WFLYCTL0158: Operation handler
failed:java.lang.RuntimeException: WFLYELY01008: Failed to obtain the authorization
identity.",
    "rolled-back" => true
}
```

関連情報

- **filesystem-realm** 属性
- Elytron での **filesystem-realm** の作成
- Elytron での暗号化された **filesystem-realm** の作成
- Elytron で整合性のある **filesystem-realm** の作成

1.1.2. Elytron での filesystem-realm の作成

JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護するためにレルムを参照する **filesystem-realm** およびセキュリティードメインを作成します。

前提条件

- JBoss EAP が実行されている。

手順

1. Elytron で **filesystem-realm** を作成します。

   構文

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>)
   ```

   例

   ```
   /subsystem=elytron/filesystem-realm=exampleSecurityRealm:add(path=fs-realm-
   users,relative-to=jboss.server.config.dir)
   {"outcome" => "success"}
   ```

2. レルムにユーザーを追加し、ユーザーのロールを設定します。

   a. ユーザーを追加します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-
      identity(identity=<user_name>)
      ```

      例

      ```
      /subsystem=elytron/filesystem-realm=exampleSecurityRealm:add-identity(identity=user1)
      {"outcome" => "success"}
      ```

b. ユーザーのパスワードを設定します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:set-password(identity=<user_name>, clear={password=<password>})
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:set-password(identity=user1, clear={password="passwordUser1"})
{"outcome" => "success"}
```

c. ユーザーのロールを設定します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-attribute(identity=<user_name>, name=<roles_attribute_name>, value=[<role_1>,<role_N>])
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:add-identity-attribute(identity=user1, name=Roles, value=["Admin","Guest"])
{"outcome" => "success"}
```

3. **filesystem-realm** を参照するセキュリティードメインを作成します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:add(default-realm=<filesystem_realm_name>,permission-mapper=default-permission-mapper,realms=[{realm=<filesystem_realm_name>,role-decoder="<role_decoder_name>"}])
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=[{realm=exampleSecurityRealm}])
{"outcome" => "success"}
```

検証

- Elytron が **filesystem-realm** からアイデンティティーをロードできることを確認するには、以下のコマンドを使用します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:read-identity(name=<username>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:read-identity(name=user1)
{
    "outcome" => "success",
    "result" => {
        "name" => "user1",
        "attributes" => {"Roles" => [
            "Admin",
            "Guest"
        ]},
        "roles" => [
            "Guest",
            "Admin"
        ]
    }
}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **filesystem-realm** 属性
- **security-domain** 属性
- **simple-role-decoder** 属性

1.1.3. Elytron で暗号化された filesystem-realm の作成

暗号化された **filesystem-realm** を作成して JBoss EAP アプリケーションまたはサーバーインターフェイスをセキュアにし、ユーザーの認証情報が暗号化され、セキュアであることを確認します。

1.1.3.1. スタンドアロンサーバー用の secret-key-credential-store の作成

管理 CLI を使用して **secret-key-credential-store** を作成します。**secret-key-credential-store** を作成すると、JBoss EAP はデフォルトでシークレットキーを生成します。生成された鍵の名前は **key** で、そのサイズは 256 ビットです。

前提条件

- JBoss EAP が実行されている。
- 少なくとも、JBoss EAP を実行しているユーザーアカウントの **secret-key-credential-store** を格納したディレクトリーへの読み取り/書き込みアクセスが割り当てられている。

手順

- 以下のコマンドで、管理 CLI を使用して **secret-key-credential-store** を作成します。

  構文

```
/subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:add(path="<path_to_the_credential_store>", relative-to=<path_to_store_file>)
```

例

```
/subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:add(path=examplePropertiesCredentialStore.cs, relative-to=jboss.server.config.dir)
{"outcome" => "success"}
```

1.1.3.2. 暗号化された filesystem-realm の作成

レルムを参照する暗号化された **filesystem-realm** と、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護するセキュリティードメインを作成します。

前提条件

- JBoss EAP が実行されている。
- **secret-key-credential-store** を作成している。
  詳細は、スタンドアロンサーバー用 **secret-key-credential-store** の作成 を参照してください。

手順

1. Elytron で暗号化された **filesystem-realm** を作成します。

   構文

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>,credential-store=<name_of_credential_store>,secret-key=<key>)
   ```

   例

   ```
   /subsystem=elytron/filesystem-realm=exampleSecurityRealm:add(path=fs-realm-users,relative-to=jboss.server.config.dir, credential-store=examplePropertiesCredentialStore, secret-key=key)
   {"outcome" => "success"}
   ```

2. レルムにユーザーを追加し、ユーザーのロールを設定します。

   a. ユーザーを追加します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity(identity=<user_name>)
      ```

      例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:add-identity(identity=user1)
{"outcome" => "success"}
```

b. ユーザーのパスワードを設定します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:set-
password(identity=<user_name>, clear={password=<password>})
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:set-
password(identity=user1, clear={password="passwordUser1"})
{"outcome" => "success"}
```

c. ユーザーのロールを設定します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-
attribute(identity=<user_name>, name=<roles_attribute_name>, value=
[<role_1>,<role_N>])
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:add-identity-
attribute(identity=user1, name=Roles, value=["Admin","Guest"])
{"outcome" => "success"}
```

3. **filesystem-realm** を参照するセキュリティードメインを作成します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:add(default-
realm=<filesystem_realm_name>,permission-mapper=default-permission-mapper,realms=
[{realm=<filesystem_realm_name>,role-decoder="<role_decoder_name>"}])
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm}])
{"outcome" => "success"}
```

検証

- Elytron が暗号化された **filesystem-realm** からアイデンティティーをロードできることを確認するには、以下のコマンドを使用します。

構文

```
```

```
/subsystem=elytron/security-domain=<security_domain_name>:read-identity(name=<username>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:read-identity(name=user1)
{
    "outcome" => "success",
    "result" => {
        "name" => "user1",
        "attributes" => {"Roles" => [
            "Admin",
            "Guest"
        ]},
        "roles" => [
            "Guest",
            "Admin"
        ]
    }
}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになります。

関連情報

- **filesystem-realm** 属性
- **security-domain** 属性
- **simple-role-decoder** 属性

1.1.4. Elytron で整合性サポートのある filesystem-realm の作成

JBoss EAP アプリケーションまたはサーバーインターフェイスをセキュアにするための整合性をサポートする **filesystem-realm** を作成し、ユーザーの認証情報が改ざんされないようにします。

1.1.4.1. 管理 CLI を使用したキーペアの作成

キーペアを持つキーストアを Elytron に作成します。

前提条件

- JBoss EAP が実行されている。

手順

1. キーストアを作成します。

   構文

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

-

例

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

2. キーストアにキーペアを作成します。

構文

```
/subsystem=elytron/key-store=<key_store_name>:generate-key-
pair(alias=<alias>,algorithm=<key_algorithm>,key-
size=<size_of_key>,validity=<validity_in_days>,distinguished-
name="<distinguished_name>")
```

例

```
/subsystem=elytron/key-store=exampleKeystore:generate-key-
pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,distinguished-
name="CN=localhost")
{"outcome" => "success"}
```

3. キーペアをキーストアファイルに永続化します。

構文

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

例

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

関連情報

- **key-store** 属性

1.1.4.2. 整合性サポートのある filesystem-realm の作成

整合性サポートのある **filesystem-realm** と、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護するためにレルムを参照するセキュリティードメインを作成します。

前提条件

- JBoss EAP が実行されている。
- **secret-key-credential-store** を作成している。

詳細は、管理 CLI を使用したキーペアの作成 を参照してください。

手順

1. Elytron で **filesystem-realm** を作成します。

   構文

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>,key-store=<key_store_name>,key-store-alias=<key_store_alias>)
   ```

   例

   ```
   /subsystem=elytron/filesystem-realm=exampleSecurityRealm:add(path=fs-realm-users,relative-to=jboss.server.config.dir, key-store=exampleKeystore, key-store-alias=localhost)
   {"outcome" => "success"}
   ```

2. レルムにユーザーを追加し、ユーザーのロールを設定します。

   a. ユーザーを追加します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity(identity=<user_name>)
      ```

      例

      ```
      /subsystem=elytron/filesystem-realm=exampleSecurityRealm:add-identity(identity=user1)
      {"outcome" => "success"}
      ```

   b. ユーザーのパスワードを設定します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:set-password(identity=<user_name>, clear={password=<password>})
      ```

      例

      ```
      /subsystem=elytron/filesystem-realm=exampleSecurityRealm:set-password(identity=user1, clear={password="passwordUser1"})
      {"outcome" => "success"}
      ```

   c. ユーザーのロールを設定します。

      構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-
attribute(identity=<user_name>, name=<roles_attribute_name>, value=
[<role_1>,<role_N>])
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:add-identity-
attribute(identity=user1, name=Roles, value=["Admin","Guest"])
{"outcome" => "success"}
```

3. **filesystem-realm** を参照するセキュリティードメインを作成します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:add(default-
realm=<filesystem_realm_name>,permission-mapper=default-permission-mapper,realms=
[{realm=<filesystem_realm_name>,role-decoder="<role_decoder_name>"}])
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm}])
{"outcome" => "success"}
```

検証

- Elytron が **filesystem-realm** からアイデンティティーをロードできることを確認するには、以下のコマンドを使用します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:read-
identity(name=<username>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:read-identity(name=user1)
{
    "outcome" => "success",
    "result" => {
        "name" => "user1",
        "attributes" => {"Roles" => [
            "Admin",
            "Guest"
        ]},
        "roles" => [
            "Guest",
            "Admin"
        ]
    }
}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **filesystem-realm** 属性
- **security-domain** 属性
- **simple-role-decoder** 属性

1.1.4.3. 整合性サポートを有効にして既存の filesystem-realm のキーペアの更新

既存の鍵が侵害された場合に整合性サポートを有効にして、**filesystem-realm** で参照されるキーペアを更新できます。また、キーをローテーションすることを推奨します。

前提条件

- キーペアを生成している。
- 整合性チェックを有効にして **filesystem-realm** を作成している。
  詳細は、整合性サポートのある **filesystem-realm** の作成 を参照してください。

手順

1. 既存のキーストアにキーペアを作成します。

   構文

   ```
   /subsystem=elytron/key-store=<key_store_name>:generate-key-
   pair(alias=<alias>,algorithm=<key_algorithm>,key-
   size=<size_of_key>,validity=<validity_in_days>,distinguished-
   name="<distinguished_name>")
   ```

   例

   ```
   /subsystem=elytron/key-store=exampleKeystore:generate-key-
   pair(alias=localhost2,algorithm=RSA,key-size=1024,validity=365,distinguished-
   name="CN=localhost")
   {"outcome" => "success"}
   ```

2. キーペアをキーストアファイルに永続化します。

   構文

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

   例

   ```
   /subsystem=elytron/key-store=exampleKeystore:store()
   {
       "outcome" => "success",
   ```

```
    "result" => undefined
}
```

3. 新しいキーペアを参照するようにキーストアエイリアスを更新します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:write-attribute(name=key-store-alias, value=<key_store_alias>)
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:write-attribute(name=key-store-alias, value=localhost2)
{
    "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
```

5. 新しいキーペアを使用して、**filesystem-realm** のファイルを新規署名で更新します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:update-key-pair()
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:update-key-pair()
{"outcome" => "success"}
```

検証

- **filesystem-realm** で参照されるキーペアが以下の管理 CLI コマンドを使用して更新されていることを確認します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:read-resource()
```

例

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:read-resource()
{
    "outcome" => "success",
    "result" => {
```

```
        "credential-store" => undefined,
        "encoded" => true,
        "hash-charset" => "UTF-8",
        "hash-encoding" => "base64",
        "key-store" => "exampleKeystoreFSRealm",
        "key-store-alias" => "localhost2",
        "levels" => 2,
        "secret-key" => undefined,
        "path" => "fs-realm-users",
        "relative-to" => "jboss.server.config.dir"
    }
}
```

filesystem-realm で参照されるキーペアが更新されました。

関連情報

- **filesystem-realm** 属性

1.1.5. 暗号化されていない **filesystem-realm** の暗号化

Elytron に **filesystem-realm** が設定されている場合は、Wild Elytron Tool を使用して暗号化を追加できます。

1.1.5.1. スタンドアロンサーバー用の secret-key-credential-store の作成

管理 CLI を使用して **secret-key-credential-store** を作成します。**secret-key-credential-store** を作成すると、JBoss EAP はデフォルトでシークレットキーを生成します。生成された鍵の名前は **key** で、そのサイズは 256 ビットです。

前提条件

- JBoss EAP が実行されている。
- 少なくとも、JBoss EAP を実行しているユーザーアカウントの **secret-key-credential-store** を格納したディレクトリーへの読み取り/書き込みアクセスが割り当てられている。

手順

- 以下のコマンドで、管理 CLI を使用して **secret-key-credential-store** を作成します。

  構文

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:add(path="<path_to_the_credential_store>", relative-to=<path_to_store_file>)
  ```

  例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:add(path=examplePropertiesCredentialStore.cs, relative-to=jboss.server.config.dir)
  {"outcome" => "success"}
  ```

1.1.5.2. 暗号化されていない filesystem-realm を暗号化された filesystem-realm に変換する

WildFly Elytron ツールの **filesystem-realm-encrypt** コマンドを使用して、暗号化されていない **filesystem-realm** を暗号化されたものに変換できます。

前提条件

- 既存の **filesystem-realm** がある。
  詳細は、Elytron での **filesystem-realm** の作成 を参照してください。
- **secret-key-credential-store** を作成している。
  詳細は、スタンドアロンサーバー用 **secret-key-credential-store** 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. 暗号化されていない **filesystem-realm** を暗号化されたものに変換します。

   構文

   ```
   $ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-encrypt --input-location
   <existing_filesystem_realm_name> --output-location
   JBOSS_HOME/standalone/configuration/<target_filesystem_realm_name> --credential-
   store <path_to_credential_store>/<credential_store>
   ```

   例

   ```
   $ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-encrypt --input-location
   JBOSS_HOME/standalone/configuration/fs-realm-users --output-location
   JBOSS_HOME/standalone/configuration/fs-realm-users-enc --credential-store
   JBOSS_HOME/standalone/configuration/examplePropertiesCredentialStore.cs

   Creating encrypted realm for: JBOSS_HOME/standalone/configuration/fs-realm-users
   Found credential store and alias, using pre-existing key
   ```

   WildFly Elytron コマンド **filesystem-realm-encrypt** によって、**--output-location** 引数で指定された **filesystem-realm** が作成されます。また、**filesystem-realm** のルートに CLI スクリプトが作成されます。このスクリプトを使用すると、**elytron** サブシステムに **filesystem-realm** リソースを追加できます。

   ヒント

   --summary オプションを使用して、コマンド実行の概要を表示します。

2. 生成された CLI スクリプトを使用して、**elytron** サブシステムに **filesystem-realm** リソースを追加します。

   構文

   ```
   $ JBOSS_HOME/bin/jboss-cli.sh --connect --
   file=<target_filesystem_realm_directory>/<target_filesystem_realm_name>.cli
   ```

例

```
$ JBOSS_HOME/bin/jboss-cli.sh --connect --
file=JBOSS_HOME/standalone/configuration/fs-realm-users-enc/encrypted-filesystem-
realm.cli
```

暗号化された **filesystem-realm** を使用して、レルムを参照して JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護するセキュリティードメインを作成できます。

関連情報

- **filesystem-realm** 属性
- **secret-key-credential-store** 属性
- Elytron での暗号化された **filesystem-realm** の作成
- WildFly Elytron ツール **filesystem-realm-encrypt** コマンドの詳細は、**filesystem-realm-encrypt --help** コマンドを実行します。

```
$ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-encrypt --help
```

1.1.6. 既存の filesystem-realm への整合性サポートの追加

Elytron で **filesystem-realm** が設定されている場合は、WildFly Elytron ツールを使用してキーペアでそのファイルシステムレルムに署名し、整合性チェックを有効にすることができます。

1.1.6.1. 管理 CLI を使用したキーペアの作成

キーペアを持つキーストアを Elytron に作成します。

前提条件

- JBoss EAP が実行されている。

手順

1. キーストアを作成します。

   構文

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

   例

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

2. キーストアにキーペアを作成します。

構文

```
/subsystem=elytron/key-store=<key_store_name>:generate-key-
pair(alias=<alias>,algorithm=<key_algorithm>,key-
size=<size_of_key>,validity=<validity_in_days>,distinguished-
name="<distinguished_name>")
```

例

```
/subsystem=elytron/key-store=exampleKeystore:generate-key-
pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,distinguished-
name="CN=localhost")
{"outcome" => "success"}
```

3. キーペアをキーストアファイルに永続化します。

構文

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

例

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

関連情報

- **key-store** 属性

1.1.6.2. filesystem-realm の整合性チェックの有効化

WildFly Elytron ツールの **filesystem-realm-integrity** コマンドを使用して、既存の空ではない **filesystem-realm** から、整合性チェックを備えた **filesystem-realm** を作成できます。

filesystem-realm-integrity コマンドは、次の場合に使用できます。

- 既存の **filesystem-realm** から、整合性チェックを備えた新しい **filesystem-realm** を作成する。
- 既存の **filesystem-realm** に整合性チェックを追加する。

前提条件

- 既存の **filesystem-realm** がある。
  詳細は、Elytron での **filesystem-realm** の作成 を参照してください。
- キーペアを生成している。
  詳細は、管理 CLI を使用したキーペアの作成 を参照してください。
- JBoss EAP が実行されている。

手順

1. 既存の **filesystem-realm** を使用し、キーペアで署名することにより、整合性をサポートする **filesystem-realm** を作成します。
既存の **filesystem-realm** に整合性サポートを追加する場合は、次のコマンドの **--output-location** および **--realm-name** オプションを省略してください。**--output-location** および **--realm-name** オプションを指定すると、既存の filesystem-realm が更新されず、整合性チェックを備えた新しい **filesystem-realm** が作成されます。

   構文

   ```
   $ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-integrity  --input-location <path_to_existing_filesystem_realm> --keystore <path_to_key_store_file> --password <keystore_password> --key-pair <key_pair_alias> --output-location <path_for_new_filesystem_realm> --realm-name <name_of_new_filesystem_realm>
   ```

   例

   ```
   $ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-integrity --input-location JBOSS_HOME/standalone/configuration/fs-realm-users/ --keystore JBOSS_HOME/standalone/configuration/keystore --password secret --key-pair localhost --output-location JBOSS_HOME/standalone/configuration/fs-realm-users --realm-name exampleRealmWithIntegrity
   ```

   出力例

   ```
   Creating filesystem realm with integrity verification for:
   JBOSS_HOME/standalone/configuration/fs-realm-users
   ```

   WildFly Elytron コマンド **filesystem-realm-integrity** によって、**--output-location** 引数で指定された **filesystem-realm** が作成されます。また、**filesystem-realm** のルートに CLI スクリプトが作成されます。このスクリプトを使用すると、**elytron** サブシステムに **filesystem-realm** リソースを追加できます。

   ヒント

   --summary オプションを使用して、コマンド実行の概要を表示します。

2. 生成された CLI スクリプトを使用して、**elytron** サブシステムに **filesystem-realm** リソースを追加します。

   構文

   ```
   $ JBOSS_HOME/bin/jboss-cli.sh --connect --file=<target_filesystem_realm_directory>/<target_filesystem_realm_name>.cli
   ```

   例

   ```
   $ JBOSS_HOME/bin/jboss-cli.sh --connect --file=JBOSS_HOME/standalone/configuration/fs-realm-users/exampleRealmWithIntegrity.cli
   ```

filesystem-realm を使用して、レルムを参照するセキュリティードメインを作成し、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護できます。

関連情報

- **filesystem-realm** 属性
- Elytron で整合性サポートのある filesystem-realm の作成
- WildFly Elytron ツールの **filesystem-realm-integrity** コマンドの詳細を確認するには、**filesystem-realm-integrity --help** コマンドを実行してください。

```
$ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-integrity --help
```

1.2. JDBC レルムの作成

1.2.1. Elytron での jdbc-realm の作成

JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護するために、レルムを参照する **jdbc-realm** およびセキュリティードメインを作成します。

この手順の例では、次のように設定された PostgreSQL データベースを使用します。

- データベース名: postgresdb
- データベースのログイン認証情報:
  - ユーザー名: postgres
  - パスワード: ポストグル
- テーブル名: example_jboss_eap_users
- example_jboss_eap_users コンテンツ:

username	password	roles
user1	passwordUser1	Admin
user2	passwordUser2	Guest

前提条件

- ユーザーを含むデータベースを設定している。
- JBoss EAP が実行されている。
- 適切な JDBC ドライバーをダウンロードしている。

手順

1. 管理 CLI を使用して、データベースのデータベースドライバーをデプロイします。

   構文

```
deploy <path_to_jdbc_driver>/<jdbc-driver>
```

例

```
deploy PATH_TO_JDBC_DRIVER/postgresql-42.2.9.jar
```

2. データベースをデータソースとして設定します。

構文

```
data-source add --name=<data_source_name> --jndi-name=<jndi_name> --driver-name=<jdbc-driver>  --connection-url=<database_URL> --user-name=<database_username> --password=<database_username>
```

例

```
data-source add --name=examplePostgresDS --jndi-name=java:jboss/examplePostgresDS --driver-name=postgresql-42.2.9.jar  --connection-url=jdbc:postgresql://localhost:5432/postgresdb --user-name=postgres --password=postgres
```

3. Elytron で **jdbc-realm** を作成します。

構文

```
/subsystem=elytron/jdbc-realm=<jdbc_realm_name>:add(principal-query=[<sql_query_to_load_users>])
```

例

```
/subsystem=elytron/jdbc-realm=exampleSecurityRealm:add(principal-query=[{sql="SELECT password,roles FROM example_jboss_eap_users WHERE username=?",data-source=examplePostgresDS,clear-password-mapper={password-index=1},attribute-mapping=[{index=2,to=Roles}]}])
{"outcome" => "success"}
```

注記

この例では、1 つの **principal-query** からパスワードおよびロールを取得する方法を示しています。ロールや追加の認証または認可情報を取得するために複数のクエリーが必要な場合は、**attribute-mapping** 属性を使用して追加の **プリンシパルクエリー** を作成することもできます。

サポート対象のパスワードマッパーの一覧は、パスワードマッパー を参照してください。

4. **jdbc-realm** を参照するセキュリティードメインを作成します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:add(default-
realm=<jdbc_realm_name>,permission-mapper=default-permission-mapper,realms=
[{realm=< jdbc_realm_name>,role-decoder="<role_decoder_name>"}])
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm}])
{"outcome" => "success"}
```

検証

- Elytron がデータベースからデータをロードできることを確認するには、次のコマンドを使用します。

  構文

```
/subsystem=elytron/security-domain=<security_domain_name>:read-
identity(name=<username>)
```

  例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:read-identity(name=user1)
{
    "outcome" => "success",
    "result" => {
        "name" => "user1",
        "attributes" => {"Roles" => ["Admin"]},
        "roles" => ["Admin"]
    }
}
```

  この出力は、Elytron がデータベースからデータをロードできることを示しています。

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **jdbc-realm** 属性
- パスワードマッパー
- **security-domain** 属性

1.3. LDAP レルムの作成

1.3.1. Elytron の LDAP レルム

Elytron の Lightweight Directory Access Protocol (LDAP) レルム (**ldap-realm**) は、LDAP ID ストアから ID をロードするために使用できるセキュリティーレルムです。

以下の例は、LDAP の ID と JBoss EAP の Elytron ID のマッピング方法を示しています。

LDAP データ交換フォーマット (LDIF) ファイルの例

```
dn: ou=Users,dc=wildfly,dc=org
objectClass: organizationalUnit
objectClass: top
ou: Users

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: userPassword1

dn: ou=Roles,dc=wildfly,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=Admin,ou=Roles,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfNames
cn: Admin
member: uid=user1,ou=Users,dc=wildfly,dc=org
```

LDAP レルムを作成するためのコマンドの例

```
/subsystem=elytron/dir-
context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",credentia
-reference={clear-text="secret"})

/subsystem=elytron/ldap-realm=exampleSecurityRealm:add(dir-context=exampleDirContext,identity-
mapping={search-base-dn="ou=Users,dc=wildfly,dc=org",rdn-identifier="uid",user-password-mapper=
{from="userPassword"},attribute-mapping=[{filter-base-dn="ou=Roles,dc=wildfly,dc=org",filter="(&
(objectClass=groupOfNames)(member={1}))",from="cn",to="Roles"}]})
```

コマンドにより、設定は以下のようになります。

```
<ldap-realm name="exampleLDAPRealm" dir-context="exampleDirContext"> 1
    <identity-mapping rdn-identifier="uid" search-base-dn="ou=Users,dc=wildfly,dc=org"> 2
        <attribute-mapping> 3
            <attribute from="cn" to="Roles" filter="(&amp;(objectClass=groupOfNames)(member={1}))"
filter-base-dn="ou=Roles,dc=wildfly,dc=org"/> 4
        </attribute-mapping>
        <user-password-mapper from="userPassword"/> 5
    </identity-mapping>
</ldap-realm>
```

1. 以下はレルムの定義になります。
   - **name** は **ldap-realm** レルム名です。
   - **dir-context** は、LDAP サーバーに接続するための設定です。
2. ID のマッピング方法を定義します。
   - **rdn-identifier** は、LDAP エントリーからプリンシパルの名前を取得する際に使用するプリンシパルの識別名 (DN) の相対識別名 (RDN) です。LDIF の例では、**uid** はベース **DN=ou=Users,dc=wildfly,dc=org** からのプリンシパルの名前を表すように設定されています。
     search-base-dn は、ID を検索するためのベース DN です。LDIF の例では、 **dn: ou=Users,dc=wildfly,dc=org** と定義されています。
3. LDAP 属性を ID の属性マッピングに定義します。
4. 特定の LDAP 属性を Elytron ID 属性としてマップする方法を設定します。
   - **from** は、マップする LDAP 属性です。定義されていない場合は、エントリーの DN が使用される。
   - **to** は、LDAP 属性からマップされた ID の属性の名前です。指定されない場合、属性の名前は **from** で定義されたものと同じになります。**from** も定義されていない場合は、エントリーの DN が使用されます。
   - **filter** は、特定の属性の値を取得するために使用するフィルターです。文字列 '{0}' はユーザー名に置き換えられ、'{1}' はユーザー ID の DN に置き換えられます。
     - **objectClass** は、使用する LDAP オブジェクトクラスです。LDIF の例では、使用するオブジェクトクラスは **groupOfNames** と定義されています。
     - **member** はマップするメンバーです。**{0}** はユーザー名に置き換えられ、**{1}** はユーザー ID の DN に置き換えられます。この例では、**{1}** を使用して、**member** を **user1** にマップしています。
   - **filter-base-dn** は、フィルターを適用する必要があるコンテキストの名前です。
     サンプルフィルターの結果は、ユーザー **user1** が **Admin** ロールにマップされていることです。
5. **user-password-mapper** は、ID のパスワードの取得先である LDAP 属性を定義します。この例では、**userPassword** として設定されており、LDIF では **userPassword1** として定義されています。

関連情報

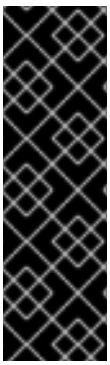
- Elytron での **ldap-realm** の作成
- **ldap-realm** 属性

1.3.2. Elytron での ldap-realm の作成

Lightweight Directory Access Protocol (LDAP) ID ストアにサポートされた Elytron セキュリティーレルムを作成します。セキュリティーレルムを使用してセキュリティードメインを作成し、管理インターフェイスまたはサーバーにデプロイされたアプリケーションに認証と認可を追加します。

注記

キャッシングレルムとして設定された **ldap-realm** は、Active Directory をサポートしていません。詳細は、Changing LDAP/AD User Password via JBossEAP CLI for Elytron を参照してください。



重要

elytron サブシステムが LDAP サーバーを使用して認証を実行する場合、JBoss EAP は **500** エラーコードまたは内部サーバーエラー (LDAP サーバーにアクセスできない場合) を返します。

LDAP サーバーが使用できなくなった場合でも、LDAP レルムを使用して保護されている管理インターフェイスやアプリケーションにアクセスできるようにするには、フェイルオーバーレルムを使用します。詳細は、フェイルオーバーレルムの作成 を参照してください。

この手順の例では、以下の LDAP Data Interchange Format (LDIF) が使用されます。

```
dn: ou=Users,dc=wildfly,dc=org
objectClass: organizationalUnit
objectClass: top
ou: Users

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: userPassword1

dn: ou=Roles,dc=wildfly,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=Admin,ou=Roles,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfNames
cn: Admin
member: uid=user1,ou=Users,dc=wildfly,dc=org
```

この例で使用されている LDAP 接続パラメーターは以下のとおりです。

- LDAP URL: **ldap://10.88.0.2**
- LDAP 管理者パスワード: **secret**
  これは、Elytron が LDAP サーバーに接続するために必要です。
- LDAP 管理者識別名 (DN): **(cn=admin,dc=wildfly,dc=org)**
- LDAP 組織: **wildfly**
  組織名が指定されていない場合、デフォルトで **Example Inc** になります。

- LDAP ドメイン: **wildfly.org**
  これは、プラットフォームが LDAP 検索リファレンスを受信したときに照合される名前です。

前提条件

- LDAP ID ストアを設定している。
- JBoss EAP が実行されている。

手順

1. LDAP サーバーへの接続に使用される URL とプリンシパルを提供するディレクトリーコンテキストを設定します。

   構文

   ```
   /subsystem=elytron/dir-context=<dir_context_name>:add(url="<LDAP_URL>",principal="<principal_distinguished_name>",credential-reference=<credential_reference>)
   ```

   例

   ```
   /subsystem=elytron/dir-context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",credential-reference={clear-text="secret"})
   ```

2. ディレクトリーコンテキストを参照する LDAP レルムを作成します。検索ベース DN とユーザーのマッピング方法を指定します。

   構文

   ```
   /subsystem=elytron/ldap-realm=<ldap_realm_name>add:(dir-context=<dir_context_name>,identity-mapping=search-base-dn="ou=<organization_unit>,dc=<domain_component>",rdn-identifier="<relative_distinguished_name_identifier>",user-password-mapper={from=<password_attribute_name>},attribute-mapping=[{filter-base-dn="ou=<organization_unit>,dc=<domain_component>",filter="<ldap_filter>",from="<ldap_attribute_name>",to="<identity_attribute_name>"}]})
   ```

   例

   ```
   /subsystem=elytron/ldap-realm=exampleSecurityRealm:add(dir-context=exampleDirContext,identity-mapping={search-base-dn="ou=Users,dc=wildfly,dc=org",rdn-identifier="uid",user-password-mapper={from="userPassword"},attribute-mapping=[{filter-base-dn="ou=Roles,dc=wildfly,dc=org",filter="(&(objectClass=groupOfNames)(member={1}))",from="cn",to="Roles"}]})
   ```

   LDIF ファイルにハッシュ化されたパスワードを保存する場合、以下の属性を指定することができます。

- **hash-encoding**: この属性は、パスワードがプレーンテキストで保存されない場合の文字列形式を指定します。デフォルトでは **base64** エンコーディングに設定されていますが、**hex** もサポートされています。
- **hash-charset**: この属性は、パスワード文字列をバイト配列に変換する際に使用する文字セットを指定します。デフォルトでは **UTF-8** に設定されています。

警告

参照される LDAP サーバーの参照元にループが含まれる場合は、JBoss EAP で **java.lang.OutOfMemoryError** エラーが発生することがあります。

3. 属性をロールにマップするロールデコーダーを作成します。

構文

```
/subsystem=elytron/simple-role-decoder=<role_decoder_name>:add(attribute=<attribute>)
```

例

```
/subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
```

4. LDAP レルムとロールデコーダーを参照するセキュリティードメインを作成します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:add(realms=
[{realm=<ldap_realm_name>,role-decoder=<role_decoder_name>}],default-
realm=<ldap_realm_name>,permission-mapper=<permission_mapper>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(realms=
[{realm=exampleSecurityRealm,role-decoder=from-roles-attribute}],default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper)
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **ldap-realm** 属性
- **security-domain** 属性

1.4. プロパティーレルムの作成

1.4.1. Elytron で properties-realm を参照するセキュリティードメインを作成する方法

JBoss EAP 管理インターフェイスまたはサーバーにデプロイしたアプリケーションを保護するために、**properties-realm** とそのレルムを参照するセキュリティードメインを作成します。

前提条件

- JBoss EAP が実行されている。
- 許可されたユーザーと既存のレガシープロパティーファイルがあり、**users.properties** ファイルのコメントアウト行に正しいレルムが書き込まれている。

  例: $EAP_HOME/standalone/configuration/my-example-users.properties

  ```
  #$REALM_NAME=exampleSecurityRealm$
  user1=078ed9776d4b8e63b6e51135ec45cc75
  ```

  - **user1** のパスワードは **userPassword1** です。パスワードは **HEX(MD5(user1:exampleSecurityRealm:userPassword1))** としてファイルにハッシュされます。
- **users.properties** ファイルにリストされている許可されたユーザーには、**groups.properties** ファイルにロールが指定されています。

  例: $EAP_HOME/standalone/configuration/my-example-groups.properties

  ```
  user1=Admin
  ```

手順

1. Elytron で **properties-realm** を作成します。

   構文

   ```
   /subsystem=elytron/properties-realm=<properties_realm_name>:add(users-properties={path=<file_path>},groups-properties={path=<file_path>})
   ```

   例

   ```
   /subsystem=elytron/properties-realm=exampleSecurityRealm:add(users-properties={path=my-example-users.properties,relative-to=jboss.server.config.dir,plain-text=true},groups-properties={path=my-example-groups.properties,relative-to=jboss.server.config.dir})
   ```

2. **properties-realm** を参照するセキュリティードメインを作成します。

   構文

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:add(default-realm=<properties_realm_name>,permission-mapper=default-permission-mapper,realms=[{realm=<properties_realm_name>,role-decoder="<role_decoder_name>"}])
   ```

   例

   -

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm,role-decoder=groups-to-roles}])
```

検証

- Elytron がプロパティーファイルからデータをロードできることを確認するには、次のコマンドを使用します。

  構文

  ```
  /subsystem=elytron/security-domain=<security_domain_name>:read-
  identity(name=<username>)
  ```

  例

  ```
  /subsystem=elytron/security-domain=exampleSecurityDomain:read-identity(name=user1)
  {
      "outcome" => "success",
      "result" => {
          "name" => "user1",
          "attributes" => {"Roles" => ["Admin"]},
          "roles" => ["Admin"]
      }
  }
  ```

  この出力から、Elytron がプロパティーファイルからデータをロードできることを確認できます。

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **properties-realm** 属性
- **security-domain** 属性
- **simple-role-decoder** 属性

1.5. カスタムレルムの作成

1.5.1. Elytron で custom-realm セキュリティーレルムを追加する方法

custom-realm を使用して、ユースケースに合わせた Elytron セキュリティーレルムを作成できます。既存の Elytron セキュリティーレルムがユースケースに合わない場合、**custom-realm** を追加できます。

前提条件

- JBoss EAP がインストールされ、実行されている。

- Maven がインストールされている。
- 実装されたカスタムレルム Java クラスがある。

手順

1. カスタムレルム Java クラスを実装し、**JAR** ファイルとしてパッケージ化します。

```
$ mvn package
```

2. カスタムレルムの実装を含むモジュールを追加します。

構文

```
module add --name=<name_of_your_wildfly_module>
--resources=<path_to_custom_realm_jar> --dependencies=org.wildfly.security.elytron
```

例

```
module add --name=com.example.customrealm --resources=EAP_HOME/custom-realm.jar --dependencies=org.wildfly.security.elytron
```

3. **custom-realm** を作成します。

構文

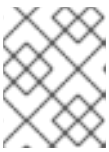
```
/subsystem=elytron/custom-realm=<name_of_your_custom_realm>:add(module=<name_of_your_wildfly_module>,class-name=<class_name_of_custom_realm_>,configuration={<configuration_option_1>=<configuration_value_1>,<configuration_option_2>=<configuration_value_2>})
```

例

```
/subsystem=elytron/custom-realm=example-realm:add(module=com.example.customrealm,class-name=com.example.customrealm.ExampleRealm,configuration={exampleConfigOption1=exampleConfigValue1,exampleConfigOption2=exampleConfigValue2})
```

注記

この例では、実装されたカスタムレルムのクラス名を **com.example.customrealm.ExampleRealm** と想定しています。



注記

configuration 属性を使用して、**key**/**value** 設定を **custom-realm** に渡すことができます。**configuration** 属性はオプションです。

4. 作成したレルムに基づいてセキュリティードメインを定義します。

構文

```
/subsystem=elytron/security-domain=<your_security_domain_name>:add(realms=
[{realm=<your_realm_name>}],default-realm=<your_realm_name>,permission-
mapper=<your_permission_mapper_name>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(realms=[{realm=example-
realm}],default-realm=example-realm,permission-mapper=default-permission-mapper)
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **custom-realm** 属性
- **security-domain** 属性
- **module add** コマンドの詳細は、JBoss EAP 管理 CLI で **--help** コマンドを実行して確認できます。

```
module add --help
```

第2章 管理インターフェイスとアプリケーションのセキュリティー保護

2.1. 管理インターフェイスへの認証および認可の追加

管理インターフェイスに認証と認可を追加し、セキュリティードメインを使用して管理インターフェイスを保護できます。認証と認可を追加した後に管理インターフェイスにアクセスするには、ユーザーはログイン認証情報を入力する必要があります。

以下のように JBoss EAP 管理インターフェイスを保護できます。

- 管理 CLI
  sasl-authentication-factory の設定による保護。
- 管理コンソール
  http-authentication-factory の設定による保護。

前提条件

- セキュリティーレルムを参照するセキュリティードメインを作成しました。
- JBoss EAP が実行されている。

手順

1. **http-authentication-factory** または **sasl-authentication-factory** を作成します。
   - **http-authentication-factory** を作成します。

     構文

     ```
     /subsystem=elytron/http-authentication-factory=<authentication_factory_name>:add(http-server-mechanism-factory=global, security-domain=<security_domain_name>, mechanism-configurations=[{mechanism-name=<mechanism-name>, mechanism-realm-configurations=[{realm-name=<realm_name>}]}])
     ```

     例

     ```
     /subsystem=elytron/http-authentication-factory=exampleAuthenticationFactory:add(http-server-mechanism-factory=global, security-domain=exampleSecurityDomain, mechanism-configurations=[{mechanism-name=BASIC, mechanism-realm-configurations=[{realm-name=exampleSecurityRealm}]}])
     {"outcome" => "success"}
     ```

   - **sasl-authentication-factory** を作成します。

     構文

     ```
     /subsystem=elytron/sasl-authentication-factory=<sasl_authentication_factory_name>:add(security-domain=<security_domain>,sasl-server-factory=configured,mechanism-configurations=
     ```

```
[{mechanism-name=<mechanism-name>,mechanism-realm-configurations=[{realm-name=<realm_name>}]}])
```

例

```
/subsystem=elytron/sasl-authentication-factory=exampleSaslAuthenticationFactory:add(security-domain=exampleSecurityDomain,sasl-server-factory=configured,mechanism-configurations=[{mechanism-name=PLAIN,mechanism-realm-configurations=[{realm-name=exampleSecurityRealm}]}])
{"outcome" => "success"}
```

2. 管理インターフェイスを更新します。

- **http-authentication-factory** を使用して管理コンソールを保護します。

  構文

  ```
  /core-service=management/management-interface=http-interface:write-attribute(name=http-authentication-factory, value=<authentication_factory_name>)
  ```

  例

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

- **sasl-authentication-factory** を使用して管理 CLI を保護します。

  構文

  ```
  /core-service=management/management-interface=http-interface:write-attribute(name=http-upgrade,value={enabled=true,sasl-authentication-factory=<sasl_authentication_factory>})
  ```

  例

  ```
  /core-service=management/management-interface=http-interface:write-attribute(name=http-upgrade,value={enabled=true,sasl-authentication-factory=exampleSaslAuthenticationFactory})
  {
      "outcome" => "success",
      "response-headers" => {
          "operation-requires-reload" => true,
          "process-state" => "reload-required"
      }
  }
  ```

3. サーバーをリロードします。

```
reload
```

検証

- 管理コンソールに認証と認可が必要であることを確認するには、**http://127.0.0.1:9990/console/index.html** から管理コンソールに移動します。ユーザー名とパスワードの入力を求められます。
- 管理 CLI に認証と認可が必要であることを確認するには、次のコマンドを使用して管理 CLI を起動します。

```
$ bin/jboss-cli.sh --connect
```

ユーザー名とパスワードの入力を求められます。

関連情報

- **http-authentication-factory** 属性
- **sasl-authentication-factory** 属性

2.2. セキュリティードメインを使用したアプリケーションユーザーの認証と認可

セキュリティーレルムを参照するセキュリティードメインを使用して、アプリケーションユーザーを認証および認可します。アプリケーションを開発するための手順は、例としてのみ提供されています。

2.2.1. 簡単な Web アプリケーションの開発

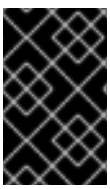
簡単な Web アプリケーションを作成して、セキュリティーレルムの設定例に従うことができます。

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、アプリケーションへの認証と認可の追加 に直接進むことができます。

2.2.1.1. web アプリケーション開発用の 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] ------------------------------------------------------------------------
```

web-application を作成できるようになりました。

2.2.1.2. 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 アプリケーション開発用の 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。

これで、セキュリティードメインを使用してこのアプリケーションを保護し、認証されたユーザーのみがアクセスできるようになります。

2.2.2. アプリケーションへの認証と認可の追加

Web アプリケーションに認証と認可を追加して、セキュリティードメインを使用してアプリケーションを保護できます。認証と認可を追加した後に Web アプリケーションにアクセスするには、ユーザーはログイン認証情報を入力する必要があります。

前提条件

- セキュリティーレルムを参照するセキュリティードメインを作成しました。
- JBoss EAP にアプリケーションをデプロイしました。
- JBoss EAP が実行されている。

手順

1. **undertow subsystem** で **application-security-domain** を設定します。

   構文

   ```
   /subsystem=undertow/application-security-domain=<application_security_domain_name>:add(security-domain=<security_domain_name>)
   ```

   例

   ```
   /subsystem=undertow/application-security-domain=exampleApplicationSecurityDomain:add(security-domain=exampleSecurityDomain)
   {"outcome" => "success"}
   ```

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。

   構文

   ```
   <!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd" >

   <web-app>

    <!-- Define the security constraints for the application resources.
        Specify the URL pattern for which a challenge is -->

    <security-constraint>
        <web-resource-collection>
            <web-resource-name><!-- Name of the resources to protect --></web-resource-name>
            <url-pattern> <!-- The URL to protect  --></url-pattern>
        </web-resource-collection>

        <!-- Define the role that can access the protected resource -->
        <auth-constraint>
   ```

```
            <role-name> <!-- Role name as defined in the security domain --></role-name>
            <!-- To disable authentication you can use the wildcard *
              To authenticate but allow any role, use the wildcard **. -->
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>
         <!-- The authentication method to use. Can be:
          BASIC
          CLIENT-CERT
          DIGEST
          FORM
          SPNEGO
          -->
        </auth-method>

        <realm-name><!-- The name of realm to send in the challenge  --></realm-name>
    </login-config>
</web-app>
```

例

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>

 <!-- Define the security constraints for the application resources.
     Specify the URL pattern for which a challenge is -->

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

        <!-- Define the role that can access the protected resource -->
        <auth-constraint>
            <role-name>Admin</role-name>
            <!-- To disable authentication you can use the wildcard *
              To authenticate but allow any role, use the wildcard **. -->
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>

        <realm-name>exampleSecurityRealm</realm-name>
    </login-config>
</web-app>
```

注記

別の **auth-method** を使用できます。

3. アプリケーションで **jboss-web.xml** ファイルを作成するか、**undertow** サブシステムでデフォルトのセキュリティードメインを設定することにより、セキュリティードメインを使用するようにアプリケーションを設定します。

   - **application-security-domain** を参照するアプリケーションの **WEB-INF** ディレクトリーに **jboss-web.xml** ファイルを作成します。

     構文

     ```
     <jboss-web>
       <security-domain> <!-- The security domain to associate with the application --
     ></security-domain>
     </jboss-web>
     ```

     例

     ```
     <jboss-web>
       <security-domain>exampleApplicationSecurityDomain</security-domain>
     </jboss-web>
     ```

   - アプリケーションの **undertow** サブシステムでデフォルトのセキュリティードメインを設定します。

     構文

     ```
     /subsystem=undertow:write-attribute(name=default-security-
     domain,value=<application_security_domain_to_use>)
     ```

     例

     ```
     /subsystem=undertow:write-attribute(name=default-security-
     domain,value=exampleApplicationSecurityDomain)
     {
         "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

1. 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] ------------------------------------------------------------------------
```

2. アプリケーションをデプロイします。

```
$ mvn wildfly:deploy
```

3. ブラウザーで **http://localhost:8080/simple-webapp-example/secured** に移動します。アプリケーションにアクセスするには認証が必要であることを確認するログインプロンプトが表示されます。

これで、アプリケーションはセキュリティードメインで保護され、ユーザーは認証後にのみログインできます。さらに、指定された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ます。

第3章 ローカルユーザーの認証および認可を容易にする ID レルムを使用した ELYTRON 設定

Elytron が提供する **identity-realm** を使用して、ローカルユーザーが JBoss EAP 管理インターフェイスに接続できるようにします。

JBoss EAP 管理 CLI は、**local** という名前の **identity-realm** を使用するように事前設定されています。そのため、ローカルユーザーは認証情報を提供しなくても接続できます。ID レルムは、**JBOSS-LOCAL-USER** メカニズムでのみ使用できます。

3.1. ID レルムによる管理インターフェイスのセキュリティー保護

JBOSS-LOCAL-USER メカニズムで **identity-realm** セキュリティーレルムを使用することにより、管理インターフェイスのセキュリティーを保護できます。

前提条件

- JBoss EAP が実行されている。

手順

1. ローカル **identity-realm** を作成します。

   構文

   ```
   /subsystem=elytron/identity-realm=
   <local_identity_realm_name>:add(identity="$local",attribute-
   name=<attribute_name>,attribute-values=<attribute_value>)
   ```

   例

   ```
   /subsystem=elytron/identity-
   realm=exampleLocalIdentityRealm:add(identity="$local",attribute-
   name=AttributeName,attribute-values=Value)
   ```

   a. **オプション**: ローカル **identity-realm** の名前を **$local** 以外にする場合は、**configurable-sasl-server-factory= <sasl_server_factory>** 属性の **wildfly.sasl.local-user.default-user** プロパティー値を変更します。

      構文

      ```
      /subsystem=elytron/configurable-sasl-server-factory=<sasl_server_factory>:write-
      attribute(name=properties,value={"wildfly.sasl.local-user.default-user" =>
      "<new_local_username>", "wildfly.sasl.local-user.challenge-path" => expression
      "${jboss.server.temp.dir}/auth"})
      ```

      例

      ```
      /subsystem=elytron/configurable-sasl-server-factory=configured:write-
      attribute(name=properties,value={"wildfly.sasl.local-user.default-user" => "john",
      "wildfly.sasl.local-user.challenge-path" => expression "${jboss.server.temp.dir}/auth"})
      ```

2. 作成した **identity-realm** を参照するセキュリティードメインを作成します。

   構文

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:add(default-realm=
   <local_identity_realm_name>,permission-mapper=<permission_mapper_name>,realms=
   [{realm=<Local_identity_realm_name>}])
   ```

   例

   ```
   /subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
   realm=exampleLocalIdentityRealm,permission-mapper=default-permission-mapper,realms=
   [{realm=exampleLocalIdentityRealm}])
   ```

3. SASL 認証ファクトリーを追加します。

   構文

   ```
   /subsystem=elytron/sasl-authentication-factory=<sasl_auth_factory_name>:add(security-
   domain=<security_domain_name>,sasl-server-factory=configured,mechanism-
   configurations=[{mechanism-name=JBOSS-LOCAL-USER}])
   ```

   例

   ```
   /subsystem=elytron/sasl-authentication-
   factory=exampleSaslAuthenticationFactory:add(security-
   domain=exampleSecurityDomain,sasl-server-factory=configured,mechanism-configurations=
   [{mechanism-name=JBOSS-LOCAL-USER}])
   ```

4. 管理インターフェイスの SASL 認証ファクトリーを有効にします。

   構文

   ```
   /core-service=management/management-interface=http-interface:write-attribute(name=http-
   upgrade,value={enabled=true,sasl-authentication-factory=<sasl_auth_factory_name>})
   ```

   例

   ```
   /core-service=management/management-interface=http-interface:write-attribute(name=http-
   upgrade,value={enabled=true,sasl-authentication-
   factory=exampleSaslAuthenticationFactory})
   ```

5. 管理インターフェイスをリロードします。

   ```
   $ reload
   ```

関連情報

- **identity-realm** 属性
- **security-domain** 属性

- **sasl-authentication-factory** 属性

第4章 ELYTRON の監査ロギングの設定

Elytron を使用し、トリガーとなるイベントのセキュリティー監査を完了することができます。セキュリティー監査とは、認可または認証の試行に応答して、ログへの書き込みなどのイベントをトリガーすることを指します。

イベントに対して行われるセキュリティー監査の種類は、セキュリティーレルムの設定によって異なります。

4.1. ELYTRON 監査ロギング

elytron サブシステムで監査ロギングを有効にすると、アプリケーションサーバー内の Elytron の認証および認可イベントをログに記録できるようになります。Elytron は、監査ログエントリーを **JSON** または **SIMPLE** 形式で保存します。人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。

Elytron の監査ロギングは、JBoss EAP 管理インターフェイスの監査ロギングなど、他のタイプの監査ロギングとは異なります。

Elytron の監査ロギングはデフォルトでは無効ですが、次のログハンドラーのいずれかを設定することで監査ロギングを有効にできます。ログハンドラーをセキュリティードメインに追加することもできます。

- ファイル監査ロギング
  詳細は、Elytron のファイル監査ロギングの有効化 を参照してください。
- 定期的なローテーションファイル監査ロギング
  詳細は、Elytron の定期ローテーションファイル監査ロギングの有効化 を参照してください。
- サイズローテーションファイル監査ロギング
  詳細は、Elytron のサイズローテーションファイル監査ロギングの有効化 を参照してください。
- **syslog** 監査ロギング
  詳細は、Elytron の syslog 監査ロギングの有効化 を参照してください。
- カスタム監査ロギング
  詳細は、Elytron でのカスタムセキュリティーイベントリスナーの使用 を参照してください。

aggregate-security-event-listener リソース を使用すると、ロガーなどのより多くの宛先にセキュリティーイベントを送信することができます。**aggregate-security-event-listener リソース** は、全イベントをアグリゲートリスナー定義で指定されたリスナーすべてに配信します。

4.2. ELYTRON のファイル監査ロギングの有効化

ファイル監査ロギングは、監査ログメッセージをファイルシステムにあるファイル 1 つに保存します。

デフォルトでは、Elytron はファイル監査ロガーとして **local-audit** を指定します。

local-audit を有効にして、スタンドアロンサーバーでは **EAP_HOME/standalone/log/audit.log** に、マネージドドメインでは **EAP_HOME/domain/log/audit.log** に、Elytron 監査ログを書き込めるようにする必要があります。

前提条件

- アプリケーションをセキュリティー保護している。
  詳細は、Elytron での **filesystem-realm** の作成 を参照してください。

手順

1. ファイルの監査ログを作成します。

   構文

   ```
   /subsystem=elytron/file-audit-log=<audit_log_name>:add(path="<path_to_log_file>",format=<format_type>,synchronized=<whether_to_log_immediately>)
   ```

   例

   ```
   /subsystem=elytron/file-audit-log=exampleFileAuditLog:add(path="file-audit.log",relative-to=jboss.server.log.dir,format=SIMPLE,synchronized=true)
   ```

2. ファイル監査ログをセキュリティードメインに追加します。

   構文

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=security-event-listener,value=<audit_log_name>)
   ```

   例

   ```
   /subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-event-listener,value=exampleFileAuditLog)
   ```

検証

1. ブラウザーで、セキュリティー保護されたアプリケーションにログインします。
   たとえば、セキュリティードメインを使用したアプリケーションユーザーの認証と認可 で作成したアプリケーションにログインするには、http://localhost:8080/simple-webapp-example/secured に移動してログインします。

2. 監査ログを保存するように設定されたディレクトリーに移動します。この手順のコマンド例を使用する場合、ディレクトリーは **EAP_HOME**/standalone/log です。
   file-audit.log というファイルが作成されることに注意してください。これには、アプリケーションへのログインによってトリガーされたイベントのログが含まれています。

   file-audit.log ファイルの例

   ```
   2023-10-24 23:31:04,WARNING,{event=SecurityPermissionCheckSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true,permission=[type=org.wildfly.security.auth.permission.LoginPermission,actions=,name=]}
   2023-10-24 23:31:04,WARNING,{event=SecurityAuthenticationSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true}
   ```

関連情報

- **file-audit-log** 属性

4.3. ELYTRON の定期ローテーションファイル監査ロギングの有効化

elytron サブシステムを使用して、スタンドアロンサーバーまたはマネージドドメインとして実行されているサーバーの定期ローテーションファイル監査ロギングを有効にできます。

定期ローテーションファイル監査ロギングでは、設定されたスケジュールに基づいて監査ログファイルを自動的にローテーションします。定期ローテーションファイル監査ロギングは、デフォルトのファイル監査ロギングに似ていますが、定期ローテーションファイル監査ロギングには、**suffix** という追加の属性が含まれています。

suffix 属性の値は、**java.time.format.Date Time Formatter** 形式で指定された日付 **(.yyyy-MM-dd** など) です。Elytron は、その接尾辞で指定された値から自動的にローテーションの周期を計算します。**elytron** サブシステムは、ログファイル名の最後に接尾辞を付加します。

前提条件

- アプリケーションをセキュリティー保護している。
  詳細は、Elytron での **filesystem-realm** の作成 を参照してください。

手順

1. 定期的なローテーションファイル監査ログを作成します。

   構文

   ```
   /subsystem=elytron/periodic-rotating-file-audit-log=<periodic_audit_log_name>:add(path="<periodic_audit_log_filename>",format=<record_format>,synchronized=<whether_to_log_immediately>,suffix="<suffix_in_DateTimeFormatter_format>")
   ```

   例

   ```
   /subsystem=elytron/periodic-rotating-file-audit-log=examplePreiodicFileAuditLog:add(path="periodic-file-audit.log",relative-to=jboss.server.log.dir,format=SIMPLE,synchronized=true,suffix="yyyy-MM-dd")
   ```

2. 定期的なローテーションファイル監査ロガーをセキュリティードメインに追加します。

   構文

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=security-event-listener,value=<periodic_audit_log_name>)
   ```

   例

   ```
   /subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-event-listener,value=examplePreiodicFileAuditLog)
   ```

検証

1. ブラウザーで、セキュリティー保護されたアプリケーションにログインします。
たとえば、セキュリティードメインを使用したアプリケーションユーザーの認証と認可 で作成したアプリケーションにログインするには、http://localhost:8080/simple-webapp-example/secured に移動してログインします。

2. 監査ログを保存するように設定されたディレクトリーに移動します。この手順のコマンド例を使用する場合、ディレクトリーは **EAP_HOME**/standalone/log です。
periodic-file-audit.log というファイルが作成されることに注意してください。これには、アプリケーションへのログインによってトリガーされたイベントのログが含まれています。

   periodic-file-audit.log ファイルの例

   ```
   2023-10-24 23:31:04,WARNING,{event=SecurityPermissionCheckSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true,permission=
   [type=org.wildfly.security.auth.permission.LoginPermission,actions=,name=]}
   2023-10-24 23:31:04,WARNING,{event=SecurityAuthenticationSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true}
   ```

関連情報

- **periodic-rotating-file-audit-log** 属性

4.4. ELYTRON のサイズローテーションファイル監査ロギングの有効化

elytron サブシステムを使用して、スタンドアロンサーバーまたはマネージドドメインとして実行されているサーバーのサイズローテーションファイル監査ロギングを有効にできます。

サイズローテーションファイル監査ロギングでは、ログファイルが設定されたファイルサイズに達すると、監査ログファイルが自動的にローテーションされます。サイズローテーションファイル監査ロギングは、デフォルトのファイル監査ロギングに似ていますが、サイズローテーションファイル監査ロギングには、追加の属性が含まれています。

ログファイルのサイズが **rotate-size** 属性で定義された制限を超えると、Elytron は現在のファイルの末尾に接尾辞 **.1** を追加し、新しいログファイルを作成します。Elytron は既存のログファイルの接尾辞を 1 ずつ増やします。たとえば、Elytron は **audit_log.1** の名前を **audit_log.2** に変更します。Elytron は、ログファイルの数が **max-backup-index** で定義されたログファイルの最大数に達するまで数を増やし続けます。ログファイルが **max-backup-index** 値を超えると、Elytron はファイルを削除します。たとえば、**max-backup-index の値として "98" が定義されている場合、audit_log.99 ファイルはこの制限を超えることになります。**

前提条件

- **アプリケーションをセキュリティー保護している。**

  詳細は、Elytron での filesystem-realm の作成 を参照してください。

手順

1. **サイズローテーションファイルの監査ログを作成します。**

構文

```
/subsystem=elytron/size-rotating-file-audit-log=<audit_log_name>:add(path="
<path_to_log_file>",format=<record_format>,synchronized=<whether_to_log_immediately
>,rotate-size="<max_file_size_before_rotation>",max-backup-
index=<max_number_of_backup_files>)
```

例

```
/subsystem=elytron/size-rotating-file-audit-log=exampleSizeFileAuditLog:add(path="size-file-
audit.log",relative-to=jboss.server.log.dir,format=SIMPLE,synchronized=true,rotate-
size="10m",max-backup-index=10)
```

2. **サイズローテーション監査ロガーをセキュリティードメインに追加します。**

構文

```
/subsystem=elytron/security-domain=<domain_size_logger>:write-attribute(name=security-
event-listener,value=<audit_log_name>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-
event-listener,value=exampleSizeFileAuditLog)
```

検証

1. ブラウザーで、セキュリティー保護されたアプリケーションにログインします。

   たとえば、セキュリティードメインを使用したアプリケーションユーザーの認証と認可 で作成したアプリケーションにログインするには、http://localhost:8080/simple-webapp-example/secured に移動してログインします。

2. 監査ログを保存するように設定されたディレクトリーに移動します。この手順のコマンド例を使用する場合、ディレクトリーは EAP_HOME/standalone/log です。

   size-file-audit.log というファイルが作成されることに注意してください。これには、アプリケーションへのログインによってトリガーされたイベントのログが含まれています。

   size-file-audit.log ファイルの例

```
2023-10-24 23:31:04,WARNING,{event=SecurityPermissionCheckSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true,permission=[type=org.wildfly.security.auth.permission.LoginPermission,actions=,name=]}
2023-10-24 23:31:04,WARNING,{event=SecurityAuthenticationSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true}
```

関連情報

- size-rotating-file-audit-log 属性

4.5. ELYTRON の SYSLOG 監査ロギングの有効化

elytron サブシステムを使用して、スタンドアロンサーバーまたはマネージドドメインとして実行されているサーバーの syslog 監査ロギングを有効にできます。syslog 監査ロギングを使用すると、ロギング結果を syslog サーバーに送信するため、ローカルファイルへのロギングよりもセキュリティーオプションが多くなります。

syslog ハンドラーは、syslog サーバーのホスト名や syslog サーバーがリッスンするポートなど、syslog サーバーへの接続に使用するパラメーターを指定します。複数の syslog ハンドラーを定義し、それらを同時にアクティブにすることができます。

対応するログ形式は、RFC5424 と RFC3164 です。伝送プロトコルは、UDP、TCP、TCP with SSL に対応しています。

最初のインスタンスの syslog を定義すると、次の例に示すメッセージを含む INFORMATIONAL 優先度イベントが、ロガーによって syslog サーバーに送信されます。

```
"Elytron audit logging enabled with RFC format: <format>"
```

<format> は監査ロギングハンドラー用に設定された Request for Comments (RFC) 形式を示します。デフォルトは RFC5424 です。

前提条件

- アプリケーションをセキュリティー保護している。

  詳細は、Elytron での filesystem-realm の作成 を参照してください。

手順

1. syslog ハンドラーを追加します。

   構文

   ```
   /subsystem=elytron/syslog-audit-log=<syslog_audit_log_name>:add(host-
   name=<record_host_name>,port=<syslog_server_port_number>,server-
   address=<syslog_server_address>,format=<record_format>,
   transport=<transport_layer_protocol>)
   ```

   また、TLS 経由で syslog サーバーにログを送信することもできます。

TLS 経由でログを送信するための syslog 設定の構文

```
/subsystem=elytron/syslog-audit-log=<syslog_audit_log_name>:add(transport=SSL_TCP,server-address=<syslog_server_address>,port=<syslog_server_port_number>,host-name=<record_host_name>,ssl-context=<client_ssl_context>)
```

2. **セキュリティードメインに syslog 監査ロガーを追加します。**

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=security-event-listener,value=<syslog_audit_log_name>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-event-listener,value=exampleSyslog)
```

関連情報

- **syslog-audit-log 属性**
- **client-ssl-context の使用**
- **The Syslog Protocol**

- The BSD syslog Protocol

4.6. ELYTRON でのカスタムセキュリティーイベントリスナーの使用

Elytron を使用して、カスタムイベントリスナーを定義できます。カスタムイベントリスナーは、受信したセキュリティーイベントを処理します。イベントリスナーは、カスタム監査ロギングに使用することも、内部 ID ストレージに対してユーザーを認証するために使用することもできます。

重要

module 管理 CLI コマンドを使用してモジュールを追加および削除する機能は、テクノロジープレビュー機能としてのみ提供されています。module コマンドは、マネージドドメインで使用する場合や、リモート管理 CLI を使用して接続する場合には適していません。実稼働環境ではモジュールを手動で追加または削除する必要があります。

テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲は、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

前提条件

- アプリケーションをセキュリティー保護している。

詳細は、Elytron での filesystem-realm の作成 を参照してください。

手順

1. java.util.function.Consumer<org.wildfly.security.auth.server.event.SecurityEvent> インターフェイスを実装するクラスを作成します。

指定のインターフェイスを使用する Java クラスの作成例

```
public class MySecurityEventListener implements Consumer<SecurityEvent> {
```

```
    public void accept(SecurityEvent securityEvent) {
        if (securityEvent instanceof SecurityAuthenticationSuccessfulEvent) {
            System.err.printf("Authenticated user \"%s\"\n",
securityEvent.getSecurityIdentity().getPrincipal());
        } else if (securityEvent instanceof SecurityAuthenticationFailedEvent) {
            System.err.printf("Failed authentication as user \"%s\"\n",
((SecurityAuthenticationFailedEvent)securityEvent).getPrincipal());
        }
    }
}
```

この例の Java クラスは、ユーザー認証に成功または失敗するたびに、メッセージを表示します。

2. **カスタムイベントリスナーをモジュールとして提供する JAR ファイルを、JBoss EAP に追加します。**

以下に、カスタムイベントリスナーをモジュールとして Elytron に追加する管理 CLI コマンドの例を示します。

module コマンドを使用して、カスタムイベントリスナーを Elytron にモジュールとして追加した例

```
/subsystem=elytron/custom-security-event-
listener=<listener_name>:add(module=<module_name>, class-name=<class_name>)
```

3. **セキュリティードメインのカスタムイベントリスナーを参照します。**

Application Domain でカスタムイベントリスナーを参照する例

```
/subsystem=elytron/security-domain=<domain_name>:write-attribute(name=security-event-
listener, value=<listener_name>)
```

4. **サービスを再起動します。**

```
$ reload
```

イベントリスナーは、指定されたセキュリティードメインからセキュリティーイベントを受信します。

関連情報

- **カスタムモジュールの手動作成**
- **手作業によるカスタムモジュールの削除**
- **カスタムコンポーネントの Elytron への追加**

第5章 参照

5.1. CUSTOM-REALM 属性

属性を設定することで custom-realm を設定できます。

表5.1 custom-realm 属性

属性	説明
class-name	カスタムレルムの実装の完全修飾クラス名。
configuration	カスタムレルムのオプションの **key**/**value** 設定。
module	カスタムレルムのロードに使用するモジュールの名前。

5.2. FILESYSTEM-REALM 属性

属性を設定することで、filesystem-realm を設定できます。

表5.2 filesystem-realm 属性

属性	説明
credential-store	レルムのクリアテキストパスワード、ハッシュ化されたパスワード、および属性を暗号化および復号するための秘密鍵が含まれる認証情報ストアへの参照。この属性を使用する場合は、**secret-key** 属性で定義して使用する秘密鍵も指定する必要があります。
encoded	ID 名をエンコード (Base32) してファイル名に格納する必要があるかどうかを示す属性。デフォルト値は **true** です。
hash-charset	パスワード文字列をバイト配列に変換する際に使用する文字セット。デフォルトは UTF-8 です。
hash-encoding	パスワードがプレーンテキストで保存されていない場合のパスワードの文字列形式。これは、以下のいずれかになります。 ● base64 ● hex デフォルトは base64 です。

属性	説明
key-store	整合性の検証に使用するキーペアが含まれるキーストアへの参照。この属性を定義する場合は、**key-store-alias** 属性にキーストアエイリアスも指定する必要があります。
key-store-alias	整合性の検証に使用するキーストア内の秘密鍵エントリーを識別するエイリアス。**key-store** 属性を定義してキーストアへの参照を追加している場合は、この属性を使用します。
levels	適用するディレクトリーハッシュのレベルの数。デフォルト値は **2** です。
path	レルムを含むディレクトリーへのパス。
relative-to	**path** で使用する事前に定義された相対パス。例: **jboss.server.config.dir**
secret-key	レルム内のクリアテキストパスワード、ハッシュ化されたパスワード、および属性を暗号化および復号する秘密鍵のエイリアス。**credential-store** 属性を定義してクレデンシャルストアへの参照を追加している場合は、この属性を使用します。

5.3. FILE-AUDIT-LOG 属性

表5.3 file-audit-log 属性

属性	説明
autoflush	監査イベントが発生するたびに、出力ストリームのフラッシュが必要かどうかを指定します。属性を定義しない場合、**synchronized** 属性値がデフォルトになります。
encoding	監査ファイルのエンコーディングを指定します。デフォルトは **UTF-8** です。指定可能な値は次のとおりです。 • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**

属性	説明
format	デフォルト値は **SIMPLE** です。人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。
path	ログファイルの保存場所を定義します。
relative-to	任意の属性です。ログファイルの保存場所を定義します。
synchronized	デフォルト値は **true** です。監査イベントが発生するたびにファイル記述子が同期されることを指定します。

5.4. HTTP-AUTHENTICATION-FACTORY 属性

http-authentication-factory を設定するには、属性を設定します。

表5.4 http-authentication-factory 属性

属性	説明
http-server-mechanism-factory	このリソースに関連付ける **HttpServerAuthenticationMechanismFactory**。
mechanism-configurations	メカニズム固有の設定のリスト。
security-domain	リソースに関連付けるセキュリティードメイン。

表5.5 http-authentication-factory mechanism-configurations 属性

属性	説明
credential-security-factory	メカニズムで必要な認証情報の取得に使用するセキュリティーファクトリー。
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。
host-name	この設定が適用されるホスト名。
mechanism-name	この設定は、名前を指定したメカニズムが使用される場合にのみ適用されます。この属性を省略すると、メカニズム名に一致します。

属性	説明
mechanism-realm-configurations	メカニズムが理解するレルム名の定義のリストです。
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。
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。
protocol	この設定が適用されるプロトコル。
realm-mapper	メカニズムによって使用されるレルムマッパー。

表5.6 http-authentication-factory mechanism-configurations mechanism-realm-configurations 属性

属性	説明
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。
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。
realm-mapper	メカニズムによって使用されるレルムマッパー。
realm-name	メカニズムにより提示されるレルムの名前。

5.5. IDENTITY-REALM 属性

属性を設定することで、identity-realm を設定できます。

表5.7 identity-realm 属性

属性	説明
attribute-name	このアイデンティティーに関連付けられた属性の名前。
attribute-values	アイデンティティーの属性に関連付けられた値の一覧。
identity	セキュリティーレルムから利用可能なアイデンティティー。

5.6. JDBC-REALM 属性

属性を設定することで jdbc-realm を設定できます。

表5.8 jdbc-realm 属性

属性	説明
hash-charset	パスワード文字列をバイト配列に変換する際に使用する文字セット。デフォルトは UTF-8 です。
principal-query	特定の鍵タイプに基づいてユーザーを認証するために使用される認証クエリーのリスト。

表5.9 jdbc-realm principal-query 属性

属性	説明
attribute-mapping	このリソースに定義された属性マッピングのリスト。
bcrypt-mapper	SQL クエリーから返されるコラムを **Bcrypt** キータイプにマッピングするキーマッパー。
clear-password-mapper	SQL クエリーから返されるこらコラムをクリアパスワードキータイプにマッピングするキーマッパー。これには、ユーザーのパスワードを表す認証クエリーからのコラムインデックスである **password-index** 子要素があります。
data-source	データベースに接続するために使用されるデータソースの名前。
salted-simple-digest-mapper	SQL クエリーから返されたコラムを **Simple Digest** キータイプにマッピングするキーマッパー。
scram-mapper	SQL クエリーから返されるコラムを **SCRAM** キータイプにマッピングするキーマッパー。
simple-digest-mapper	SQL クエリーから返されたコラムを **Simple Digest** キータイプにマッピングするキーマッパー。
sql	特定のユーザーのテーブル列としてキーを取得し、そのタイプに応じてキーをマッピングするために使用される SQL ステートメント。

表5.10 jdbc-realm principal-query attribute-mapping 属性

属性	説明
index	マップされた属性を表す SQL クエリーからのカラムインデックス。
to	SQL クエリーから返されるカラムからマッピングされたアイデンティティー属性の名前。

関連情報

- パスワードマッパー属性

5.7. 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5.11 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。

属性	説明
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。
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。

5.8. LDAP-REALM 属性

属性を設定することにより、ldap-realm を設定できます。

表5.12 ldap-realm 属性

属性	説明
allow-blank-password	このレルムが、空のパスワードの直接検証に対応しているかどうか。この属性が設定されていない場合、空白のパスワードの試行は拒否されます。

属性	説明
dir-context	LDAP サーバーへの接続に使用される **dir-context** の名前。
直接検証	この属性が **true** に設定されている場合、このレルムは、認証されるアカウントとして LDAP に直接接続することによる認証情報の検証をサポートします。それ以外の場合は、LDAP サーバーからパスワードを取得し、JBoss EAP で検証します。有効にした場合、JBoss EAP サーバーはクライアントからプレーンユーザーパスワードを取得できる必要があります。これには、認証に **PLAIN** SASL または **BASIC** HTTP メカニズムのいずれかを使用する必要があります。デフォルトは **false** です。
hash-charset	パスワード文字列をバイト配列に変換する際に使用する文字セット。デフォルトは UTF-8 です。
hash-encoding	パスワードがプレーンテキストで保存されていない場合のパスワードの文字列形式。これは、以下のいずれかになります。 ● base64 ● hex デフォルトは base64 です。
identity-mapping	基礎となる LDAP サーバーで対応するエントリーにプリンシパルをマッピングする方法を定義する設定オプション。

表5.13 ldap-realm identity-mapping 属性

属性	説明
attribute-mapping	このリソースに定義された属性マッピングのリスト。
filter-name	名前でアイデンティティーを取得する LDAP フィルター。
iterator-filter	レルムのアイデンティティーを繰り返し処理する LDAP フィルター。
new-identity-属性	新規に作成された ID の属性のリスト。レルムの変更に必要です。これは、**name** と **value** ペアオブジェクトです。
new-identity-parent-dn	新しく作成された ID の親の DN。レルムの変更に必要。
otp-credential-mapper	OTP 認証情報の認証情報のマッピング。
rdn-identifier	LDAP エントリーからプリンシパル名を取得するために使用されるプリンシパルの DN の RDN 部分。これは新規アイデンティティーの作成時にも使用されます。

属性	説明
search-base-dn	アイデンティティーを検索するベース DN。
use-recursive-search	この属性が **true** に設定されている場合、ID 検索クエリーは再帰的になります。デフォルトは **false** です。
user-password-mapper	userPassword と同様の認証情報の認証情報マッピング。
x509-credential-mapper	X509 認証情報のストレージとして LDAP を使用できるようにする設定。**-from** 子属性が定義されていない場合は、この設定は無視されます。複数の **-from** 子属性が定義されている場合、ユーザー証明書は定義されたすべての基準に一致する必要があります。

表5.14 ldap-realm identity-mapping attribute-mapping 属性

属性	説明
extract-rdn	Raw 形式の値が X.500 形式の場合に、属性の値として使用する RDN キー。
filter	特定の属性の値を取得するために使用するフィルター。文字列 **{0}** はユーザー名に置き換えられ、`{1}` はユーザー ID の DN に置き換えられます。
filter-base-dn	フィルターが実行されるコンテキストの名前。
from	アイデンティティー属性にマッピングするための LDAP 属性の名前。定義されていない場合は、エントリーの DN が使用されます。
reference	値を取得するエントリーの DN を含む LDAP 属性の名前。
role-recursion	再帰的なロール割り当ての最大深度。**0** で、再起なしを指定します。デフォルトは **0** です。
role-recursion-name	ロールエントリーの LDAP 属性を確認します。これは、ロールのロール検索時に **filter-name** の "{0}" に代わるものです。
search-recursive	**true** 属性の場合、LDAP 検索クエリーは再帰的になります。デフォルトは **true** です。
to	特定の LDAP 属性からマップされたアイデンティティー属性の名前。指定されない場合、属性の名前は **from** の定義と同じになります。**from** も定義されていない場合は、**dn** 値が使用されます。

表5.15 ldap-realm identity-mapping user-password-mapper 属性

属性	説明
from	アイデンティティー属性にマッピングするための LDAP 属性の名前。定義されていない場合は、エントリーの DN が使用されます。
verifiable	**true** パスワードを使用してユーザーを検証できる場合。デフォルトは **true** です。
writable	**true** の場合は、パスワードを変更できます。デフォルトは **false** です。

表5.16 ldap-realm identity-mapping otp-credential-mapper 属性

属性	説明
algorithm-from	OTP アルゴリズムの LDAP 属性の名前。
hash-from	OTP ハッシュ関数の LDAP 属性の名前。
seed-from	OTP シードの LDAP 属性の名前。
sequence-from	OTP シーケンス番号の LDAP 属性名。

表5.17 ldap-realm identity-mapping x509-credential-mapper 属性

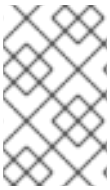
属性	説明
certificate-from	エンコードされたユーザー証明書にマップする LDAP 属性の名前。定義されていない場合は、エンコードされた証明書はチェックされません。
digest-algorithm	ユーザー証明書のダイジェストを計算するために使用される、ハッシュ関数のダイジェストアルゴリズム。**digest-from** が定義されている場合にのみ使用されます。
digest-from	ユーザー証明書ダイジェストにマップする LDAP 属性の名前。定義されていない場合、証明書のダイジェストはチェックされません。
serial-number-from	ユーザー証明書のシリアル番号にマップする LDAP 属性の名前。定義されていない場合は、シリアル番号はチェックされません。
subject-dn-from	ユーザー証明書のサブジェクト DN にマップする LDAP 属性の名前。定義されていない場合、サブジェクト DN はチェックされません。

5.9. パスワードマッパー属性

パスワードマッパーは、以下のアルゴリズムタイプのいずれかを使用してデータベースの複数のフィールドをもとにパスワードを作成します。

- クリアテキスト
- シンプルダイジェスト
- ソルトシンプルダイジェスト
- bcrypt
- SCRAM
- モジュール暗号化

パスワードマッパーには以下の属性があります。



注記

どのマッパーも最初のコラムのインデックスは、1 になります。

表5.18 パスワードマッパーの属性

マッパー名	属性	暗号化方法
clear-password-mapper	• password-index クリアテキストパスワードが含まれるコラムのインデックス。	暗号化なし

マッパー名	属性	暗号化方法
simple-digest	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **algorithm** 使用するハッシュアルゴリズム。以下の値がサポートされます。 ◦ **simple-digest-md2** ◦ **simple-digest-md5** ◦ **simple-digest-sha-1** ◦ **simple-digest-sha-256** ◦ **simple-digest-sha-384** ◦ **simple-digest-sha-512** • **hash-encoding** 表現ハッシュを指定します。使用できる値: ◦ **base64** (デフォルト) ◦ **hex**	簡単なハッシュメカニズムを使用します。

マッパー名	属性	暗号化方法
salted-simple-digest	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **algorithm** 使用するハッシュアルゴリズム。以下の値がサポートされます。 ◦ **password-salt-digest-md5** ◦ **password-salt-digest-sha-1** ◦ **password-salt-digest-sha-256** ◦ **password-salt-digest-sha-384** ◦ **password-salt-digest-sha-512** ◦ **salt-password-digest-md5** ◦ **salt-password-digest-sha-1** ◦ **salt-password-digest-sha-256** ◦ **salt-password-digest-sha-384** ◦ **salt-password-digest-sha-512** • **salt-index** ハッシュに使用するソルトを含むコラムのインデックス。 • **hash-encoding** ハッシュの表現を指定します。使用できる値: ◦ **base64** (デフォルト) ◦ **hex** • **salt-encoding** ソルトの表現を指定します。使用できる値: ◦ **base64** (デフォルト) ◦ **hex**	ソルトには単純なハッシュメカニズムを使用します。

マッパー名	属性	暗号化方法
bcrypt-password-mapper	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **salt-index** ハッシュに使用するソルトを含むコラムのインデックス。 • **iteration-count-index** 使用する反復数を含むコラムのインデックス。 • **hash-encoding** ハッシュの表現を指定します。使用できる値: ◦ **base64** (デフォルト) ◦ **hex** • **salt-encoding** ソルトの表現を指定します。使用できる値: ◦ **base64** (デフォルト) ◦ **hex**	ハッシュ化に使用する Blowfish アルゴリズム。

マッパー名	属性	暗号化方法
scram-mapper	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **algorithm** 使用するハッシュアルゴリズム。以下の値がサポートされます。 ◦ **scram-sha-1** ◦ **scram-sha-256** ◦ **scram-sha-384** ◦ **scram-sha-512** • **salt-index** ソルトを含むコラムのインデックスはハッシュに使用されます。 • **iteration-count-index** 使用する反復数を含むコラムのインデックス。 • **hash-encoding** ハッシュの表現を指定します。使用できる値: ◦ **base64** (デフォルト) ◦ **hex** • **salt-encoding** ソルトの表現を指定します。使用できる値: ◦ **base64** (デフォルト) ◦ **hex**	Salted Challenge Response Authentication メカニズムはハッシュに使用します。
modular-crypt-mapper	• **password-index** 暗号化されたパスワードを含むコラムのインデックス。	modular-crypt エンコーディングは、複数の情報を単一の文字列にエンコードすることをサポートしています。情報には以下が含まれます。 • パスワードタイプ • ハッシュまたはダイジェスト • salt • イテレーション数

5.10. PERIODIC-ROTATING-FILE-AUDIT-LOG 属性

表5.19 periodic-rotating-file-audit-log 属性

属性	説明
autoflush	監査イベントが発生するたびに、出力ストリームのフラッシュが必要かどうかを指定します。属性を定義しない場合、**synchronized** 属性値がデフォルトになります。
encoding	監査ファイルのエンコーディングを指定します。デフォルトは **UTF-8** です。指定可能な値は次のとおりです。 • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**
format	人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。
path	ログファイルの保存場所を定義します。
relative-to	任意の属性です。ログファイルの保存場所を定義します。
接尾辞	任意の属性です。ローテーションされたログに日付の接尾辞を追加します。**java.time.format.DateTimeFormatter** の形式を使用する必要があります。例: **.yyyy-MM-dd**。
synchronized	デフォルト値は **true** です。監査イベントが発生するたびにファイル記述子が同期されることを指定します。

5.11. PROPERTIES-REALM 属性

属性を設定することで、properties-realm を設定できます。

表5.20 properties-realm 属性

属性	説明
groups-attribute	アイデンティティーのグループメンバーシップ情報が含まれる必要がある、返された **AuthorizationIdentity** の属性の名前。
groups-properties	ユーザーおよびそれらのグループが含まれるプロパティーファイル。
hash-charset	クライアントから提供されたパスワード文字列をハッシュ計算用のバイトアレイに変換するときに使用する文字セットの名前を指定します。デフォルトでは **UTF-8** に設定されています。
hash-encoding	パスワードがプレインテキストで保存されていない場合に、ハッシュされたパスワードの文字列形式を指定します。**hex** または **base64** のいずれかを指定できます。**properties-realm** のデフォルトは **hex** に設定されています。
users-properties	ユーザーとパスワードが含まれるプロパティーファイル。

表5.21 properties-realm users-properties 属性

属性	説明
digest-realm-name	プロパティーファイルで検出されない場合に、ダイジェストされたパスワードに使用するデフォルトのレルム名。
path	ユーザーおよびパスワードを含むファイルへのパス。このファイルには、レルム名の宣言が含まれる必要があります。
plain-text	**true** の場合、プロパティーファイルのパスワードはプレインテキストで保存されます。**false** の場合は事前にハッシュ化され、**HEX(MD5(username ':' realm ':' password))** 形式になります。デフォルトは **false** です。
relative-to	パスが相対する、事前に定義されたパス。

表5.22 properties-realm groups-properties 属性

属性	説明
path	ユーザーおよびそれらのグループを含むファイルへのパスです。
relative-to	パスが相対する、事前に定義されたパス。

5.12. SASL-AUTHENTICATION-FACTORY 属性

属性を設定して sasl-authentication-factory を設定できます。

表5.23 sasl-authentication-factory 属性

属性	説明
mechanism-configurations	メカニズム固有の設定のリスト。
sasl-server-factory	このリソースに関連付ける SASL サーバーファクトリー。
security-domain	このリソースに関連付けるセキュリティードメイン。

表5.24 sasl-authentication-factory mechanism-configurations 属性

属性	説明
credential-security-factory	メカニズムで必要な認証情報の取得に使用するセキュリティーファクトリー。
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。
host-name	この設定が適用されるホスト名。
mechanism-name	この設定は、名前を指定したメカニズムが使用される場合にのみ適用されます。この属性を省略すると、メカニズム名に一致します。
mechanism-realm-configurations	メカニズムが理解するレルム名の定義のリストです。
protocol	この設定が適用されるプロトコル。
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。
realm-mapper	メカニズムによって使用されるレルムマッパー。

表5.25 sasl-authentication-factory mechanism-configurations mechanism-realm-configurations 属性

属性	説明
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。

属性	説明
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。
realm-mapper	メカニズムによって使用されるレルムマッパー。
realm-name	メカニズムにより提示されるレルムの名前。

5.13. SECRET-KEY-CREDENTIAL-STORE 属性

secret-key-credential-store は、その属性を設定することで設定できます。

表5.26 secret-key-credential-store 属性

属性	説明
create	まだ存在していない場合に Elytron に作成させたくない場合は、値を **false** に設定してください。デフォルトは **true** です。
default-alias	デフォルトで生成されるキーのエイリアス名です。デフォルト値は **key** です。
key-size	生成される鍵のサイズです。デフォルトのサイズは 256 ビットです。以下のいずれかの値を設定できます。 • **128** • **192** • **256**
path	クレデンシャルストアへのパス。
populate	クレデンシャルストアに **default-alias** が含まれていない場合、この属性は Elytron が作成するかどうかを示します。デフォルトは **true** です。
relative-to	属性 **path** と相対パスにある、以前に定義済みのパスへの参照。

5.14. 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。
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。

5.15. SIMPLE-ROLE-DECODER 属性

属性を設定することにより、単純なロールデコーダーを設定できます。

表5.27 simple-role-decoder 属性

属性	説明
attribute	直接ロールにマップするアイデンティティーの属性名。

5.16. SIZE-ROTATING-FILE-AUDIT-LOG 属性

表5.28 size-rotating-file-audit-log 属性

属性	説明
autoflush	監査イベントが発生するたびに、出力ストリームのフラッシュが必要かどうかを指定します。属性を定義しない場合、**synchronized** 属性値がデフォルトになります。
encoding	監査ファイルのエンコーディングを指定します。デフォルトは **UTF-8** です。指定可能な値は次のとおりです。 • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**
format	デフォルト値は **SIMPLE** です。人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。
max-backup-index	ローテーション時にバックアップするファイルの最大数です。デフォルト値は **1** です。
path	ログファイルの保存場所を定義します。
relative-to	任意の属性です。ログファイルの保存場所を定義します。

属性	説明
rotate-on-boot	デフォルトでは、サーバーの再起動時に新しいログファイルは Elytron により作成されません。この属性を **true** に設定すると、サーバーの再起動時にログを回転させることができます。
rotate-size	Elytron がログをローテーションする前にログファイルが到達する最大サイズ。デフォルトは **10m**(10メガバイト) です。また、ログの最大サイズを k、g、b、t の単位で定義することもできます。単位の指定は、大文字でも小文字でも可能です。
接尾辞	任意の属性です。ローテーションされたログに日付の接尾辞を追加します。**java.text.format.DateTimeFormatter** の形式を使用する必要があります。たとえば、**.yyyy-MM-dd-HH**。
synchronized	デフォルト値は **true** です。監査イベントが発生するたびにファイル記述子が同期されることを指定します。

5.17. SYSLOG-AUDIT-LOG 属性

表5.29 syslog-audit-log 属性

属性	説明
format	監査イベントが記録される形式。 サポートされる値: • **JSON** • **SIMPLE** デフォルト値: • **SIMPLE**
host-name	syslog サーバーに送信されるすべてのイベントに埋め込まれるホスト名。
port	**syslog** サーバーのリスニングポート。

属性	説明
reconnect-attempts	接続を閉じる前に Elytron が **syslog** サーバーへの連続メッセージの送信を試行する最大回数。この属性の値は、使用される送信プロトコルが UDP の場合にのみ有効です。 サポートされる値: ● 正の **integer** 値 ● **-1** は、再接続が無限に試行されることを示します。 デフォルト値: ● **0**
server-address	**syslog** サーバーの IP アドレス、または Java の **InetAddress.getByName()** メソッドで解決できる名前。
ssl-context	**syslog** サーバーへの接続時に使用する SSL コンテキスト。この属性は、**transport** が **SSL_TCP** に設定されている場合にのみ必要です。
syslog-format	監査イベントの記述に使用される RFC 形式。 サポートされる値: ● **RFC3164** ● **RFC5424** デフォルト値: ● **RFC5424**
transport	**syslog** サーバーへの接続に使用するトランスポート層プロトコル。 サポートされる値: ● **SSL_TCP** ● **TCP** ● **UDP** デフォルト値: ● **TCP**