Enrollment

Overview

This sample demonstrates a custom enrollment process and step-up authorization. There is a one time enrollment process during which the user is required to enter username and password and define a PIN code.

Prerequisites: Make sure to read the ExternalizableSecurityCheck (../externalizable-security-check/) and Step-up (../step-up/) tutorials.

Jump to:

- Application Flow
- Storing Data in Persistent Attributes
- Security Checks
- Sample Applications

Application Flow

- When the application starts for the first time (before enrollment) it shows the UI with two buttons: "Get public data" and "Enroll".
- When the user starts enrollment (taps on the "Enroll" button) he is prompted with login form and is then requested to set a PIN code.
- After enrolling successfully, the UI includes four buttons: "Get public data", "Get balance", "Get transactions" and "Logout". The user can access all four buttons without entering the PIN code.
- When the application is launched for a second time (after enrollment) the UI includes all four buttons however after clicking the "Get transactions" button the user will be required to enter the PIN code.

After three failing attempts at entering the PIN code the user is prompted to authenticate again with a username and password and re-setting a PIN code.

Storing Data in Persistent Attributes

You can choose to save protected data in the `PersistentAttributes` object which is a container for custom attributes of a registered client. The object can be accessed either from a security check class or from an adapter resource class.

In the provided sample application the `PersistentAttributes` object is used in the adapter resource class to store the PIN code:

- The **isEnrolled** resource returns `true` if the **pinCode** attribute exist and `false` otherwise.

  ```
  @GET
  @Path("/isEnrolled")
  public boolean isEnrolled(){
    PersistentAttributes protectedAttributes = adapterSecurityContext.getClientRegistrationData().getProtectedAttributes();
    return (protectedAttributes.get("pinCode") != null);
  }
  ```

- The **setPinCode** resource adds the **pinCode** attribute and calls the `AdapterSecurityContext.storeClientRegistrationData()` method to store the changes.
  If the security check sets an `AuthenticatedUser`, the `ClientData` object will contain the user's properties. In our sample we get the authenticated user's display name and adds it to the `Response` object.

  ```
  @POST
  @Produces("application/json")
  @OAuthSecurity(scope = "setPinCode")
  @Path("/setPinCode/{pinCode}")
  public Response setPinCode(@PathParam("pinCode") String pinCode){
      ClientData clientData = adapterSecurityContext.getClientRegistrationData();
      clientData.getProtectedAttributes().put("pinCode", pinCode);
      adapterSecurityContext.storeClientRegistrationData(clientData);
      AuthenticatedUser authenticatedUser = null;
      if (clientData.getUsers() != null) {
          authenticatedUser = clientData.getUsers().get("EnrollmentUserLogin");
      }
      Map<String, Object> user = new HashMap<String, Object>();
      user.put("userName", authenticatedUser.getDisplayName());
      return Response.ok(user).build();
  }
  ```

  Here, `users` has a key called `EnrollmentUserLogin` which itself contains the `AuthenticatedUser` object.

- The **unenroll** resource deletes the **pinCode** attribute and calls the `AdapterSecurityContext.storeClientRegistrationData()` method to store the changes.

  ```
  @DELETE
  @OAuthSecurity(scope = "unenroll")
  @Path("/unenroll")
  public Response unenroll(){
      ClientData clientData = adapterSecurityContext.getClientRegistrationData();
      if (clientData.getProtectedAttributes().get("pinCode") != null){
          clientData.getProtectedAttributes().delete("pinCode");
          adapterSecurityContext.storeClientRegistrationData(clientData);
      }
      return Response.ok().build();
  }
  ```

Security Checks

The Enrollment sample contains three security checks:

EnrollmentUserLogin

The `EnrollmentUserLogin` security check is protecting the **setPinCode** resource so that only authenticated users could set a PIN code. It should expire quickly and hold only for the duration of "first time experience".
It is identical to the `UserLogin` security check explained in the Implementing the

UserAuthenticationSecurityCheck (../user-authentication/security-check) tutorial except for an extra `isLoggedIn` method. The `isLoggedIn` method returns `true` if the security check state equals SUCCESS and `false` otherwise:

```
public boolean isLoggedIn(){
   return getState().equals(STATE_SUCCESS);
}
```

EnrollmentPinCode

The `EnrollmentPinCode` security check is protecting the **Get transactions** resource and is similar to the `PinCodeAttempts` security check explained in the Implementing the CredentialsValidationSecurityCheck (../credentials-validation/security-check) tutorial except for a few changes.

- In this tutorial's example, `EnrollmentPinCode` **depends on** `EnrollmentUserLogin`. The user should only be asked to enter a PIN code if a successful login to `EnrollmentUserLogin` previously happened.

  ```
  @SecurityCheckReference
  private transient EnrollmentUserLogin userLogin;
  ```

- When the application starts **for the first time** and the user is successfully enrolled, we want him to be able to access the **Get transactions** resource without having to enter the PIN code he just set. To do so, in our `authorize` method, we use the `EnrollmentUserLogin.isLoggedIn` method to check if the user is logged in. This means that as long as `EnrollmentUserLogin` is not expired the user can access **Get transactions**.

  ```
  @Override
  public void authorize(Set<String> scope, Map<String, Object> credentials, HttpServletRequest re
  quest, AuthorizationResponse response) {
     if (userLogin.isLoggedIn()){
        setState(STATE_SUCCESS);
        response.addSuccess(scope, userLogin.getExpiresAt(), getName());
     }
  }
  ```

  When the user fails to enter the PIN code after three attempts, we want to delete the **pinCode** attribute before he is prompted to authenticate with the username and password and re-setting a PIN code.

```
@Override
public void authorize(Set<String> scope, Map<String, Object> credentials, HttpServletRequest request, AuthorizationResponse response) {
    PersistentAttributes attributes = registrationContext.getRegisteredProtectedAttributes();
    if (userLogin.isLoggedIn()){
        setState(STATE_SUCCESS);
        response.addSuccess(scope, userLogin.getExpiresAt(), getName());
    } else {
        super.authorize(scope, credentials, request, response);
        if (getState().equals(STATE_BLOCKED)){
            attributes.delete("pinCode");
        }
    }
}
```

- The `validateCredentials` method is the same as in `PinCodeAttempts` security check except that in here we compare the credentials to the stored **pinCode** attribute.

```
@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
    PersistentAttributes attributes = registrationContext.getRegisteredProtectedAttributes();
    if(credentials!=null && credentials.containsKey("pin")){
        String pinCode = credentials.get("pin").toString();

        if(pinCode.equals(attributes.get("pinCode"))){
            errorMsg = null;
            return true;
        }
        else {
            errorMsg = "Pin code is not valid. Hint: " + attributes.get("pinCode");
        }
    }
    else{
        errorMsg = "Pin code was not provided";
    }
    //In any other case, credentials are not valid
    return false;
}
```

IsEnrolled

The `IsEnrolled` security check is protecting:

- The **getBalance** resource so that only enrolled users can see the balance.
- The **unenroll** resource so that deleting the **pinCode** will be possible only if it has been set before.

Creating the Security Check

Create a Java adapter (../../adapters/creating-adapters/) and add a Java class named `IsEnrolled` that extends `ExternalizableSecurityCheck`.

```
public class IsEnrolled  extends ExternalizableSecurityCheck{
   protected void initStateDurations(Map<String, Integer> durations) {}

   public void authorize(Set<String> scope, Map<String, Object> credentials, HttpServletRequest
request, AuthorizationResponse response) {}

   public void introspect(Set<String> scope, IntrospectionResponse response) {}
}
```

The IsEnrolledConfig Configuration Class

Create an `IsEnrolledConfig` configuration class that extends `ExternalizableSecurityCheckConfig`:

```
public class IsEnrolledConfig extends ExternalizableSecurityCheckConfig {

   public int successStateExpirationSec;

   public IsEnrolledConfig(Properties properties) {
      super(properties);
      successStateExpirationSec = getIntProperty("expirationInSec", properties, 8000);
   }
}
```

Add the `createConfiguration` method to the `IsEnrolled` class:

```
public class IsEnrolled  extends ExternalizableSecurityCheck{
   @Override
   public SecurityCheckConfiguration createConfiguration(Properties properties) {
      return new IsEnrolledConfig(properties);
   }
}
```

The initStateDurations Method

Set the duration for the SUCCESS state to `successStateExpirationSec`:

```
@Override
protected void initStateDurations(Map<String, Integer> durations) {
   durations.put (SUCCESS_STATE, ((IsEnrolledConfig) config).successStateExpirationSec);
}
```

The authorize Method

In our sample we simply check if the user is enrolled and return success or failure respectively:

```
public void authorize(Set<String> scope, Map<String, Object> credentials, HttpServletRequest request,
AuthorizationResponse response) {
  PersistentAttributes attributes = registrationContext.getRegisteredProtectedAttributes();
  if (attributes.get("pinCode") != null){
    setState(SUCCESS_STATE);
    response.addSuccess(scope, getExpiresAt(), this.getName());
  } else {
    setState(STATE_EXPIRED);
    Map <String, Object> failure = new HashMap<String, Object>();
    failure.put("failure", "User is not enrolled");
    response.addFailure(getName(), failure);
  }
}
```

- In case the "pinCode" attribute exist:
  - Set the state to SUCCESS by using the `setState` method.
  - Add success to the response object by using the `addSuccess` method.
- In case the "pinCode" attribute doesn't exist:
  - Set the state to EXPIRED by using the `setState` method.
  - Add failure to the response object by using the `addFailure` method.

Sample Applications

Security check

The `EnrollmentUserLogin`, `EnrollmentPinCode` and `IsEnrolled` security checks are available in the SecurityChecks project under the Enrollment Maven project. Click to download (https://github.com/MobileFirst-Platform-Developer-Center/SecurityCheckAdapters/tree/release80) the Security Checks Maven project.

Applications

Sample applications are available for iOS (Swift), Android, Cordova and Web.

- Click to download (https://github.com/MobileFirst-Platform-Developer-Center/EnrollmentCordova/tree/release80) the Cordova project.
- Click to download (https://github.com/MobileFirst-Platform-Developer-Center/EnrollmentSwift/tree/release80) the iOS Swift project.
- Click to download (https://github.com/MobileFirst-Platform-Developer-Center/EnrollmentAndroid/tree/release80) the Android project.
- Click to download (https://github.com/MobileFirst-Platform-Developer-Center/EnrollmentWeb/tree/release80) the Web app project.

Deploy adapters

- Use either Maven, MobileFirst CLI or your IDE of choice to build and deploy (../../adapters/creating-adapters/) the available **Enrollment** and **ResourceAdapter** adapters.

Register applications

Cordova
From a **Command-line** window, navigate to the project's root folder and:

- Add a platform by running the `cordova platform add` command.
- Registering the application: `mfpdev app register`.

Native
From a **Command-line** window, navigate to the application's root folder and run the command: `mfpdev app register`.

Web
Make sure you have Node.js installed.
Register the application in the MobileFirst Operations Console.

Scope mapping

In the **MobileFirst Operations Console → EnrollmentWeb → Security**, map the following scopes:

- `setPinCode` scope to `EnrollmentUserLogin` security check
- `accessRestricted` scope to `IsEnrolled` security check
- `unenroll` scope to `IsEnrolled` security check
- `transactions` scope to `EnrollmentPinCode` and `IsEnrolled` security checks

Alternatively, from the **Command-line**, navigate to the project's root folder and run the command: `mfpdev app push`.

> **Note for web sample:**
> Start the reverse proxy by running the commands: `npm install` followed by: `npm start`.
> In a browser, load the URL http://localhost:9081/sampleapp (http://localhost:9081/sampleapp).