Implementing the CredentialsValidationSecurityCheck class

Overview

This abstract class extends `ExternalizableSecurityCheck` and implements most of its methods to simplify usage. Two methods are mandatory: `validateCredentials` and `createChallenge`.
The `CredentialsValidationSecurityCheck` class is meant for simple flows to validate arbitrary credentials in order to grant access to a resource. Also provided is a built-in capability to block access after a set number of attempts.

This tutorial uses the example of a hard-coded PIN code to protect a resource, and gives the user 3 attempts (after which the client app instance is blocked for 60 seconds).

Prerequisites: Make sure to read the Authorization concepts (../../) and Creating a Security Check (../../creating-a-security-check) tutorials.

Jump to:

Creating the Security Check

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

```
public class PinCodeAttempts extends CredentialsValidationSecurityCheck {

    @Override
    protected boolean validateCredentials(Map<String, Object> credentials) {
        return false;
    }

    @Override
    protected Map<String, Object> createChallenge() {
        return null;
    }
}
```

Creating the challenge

When the security check is triggered, it sends a challenge to the client. Returning `null` creates an empty challenge, which may be sufficient in some cases.
Optionally, you can return data with the challenge, such as an error message to display, or any other data that can be used by the client.

For example, `PinCodeAttempts` sends a predefined error message and the number of remaining attempts.

```
@Override
protected Map<String, Object> createChallenge() {
   Map challenge = new HashMap();
   challenge.put("errorMsg",errorMsg);
   challenge.put("remainingAttempts",getRemainingAttempts());
   return challenge;
}
```

> The implementation of `errorMsg` is included in the sample application.

`getRemainingAttempts()` is inherited from `CredentialsValidationSecurityCheck`.

Validating the user credentials

When the client sends the answer from the challenge, the answer is passed to `validateCredentials` as a `Map`. This method should implement your logic and return `true` if the credentials are valid.

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

      if(pinCode.equals("1234")){
         return true;
      }
      else {
         errorMsg = "The pin code is not valid.";
      }

   }
   else{
      errorMsg = "The pin code was not provided.";
   }

   //In any other case, credentials are not valid
   return false;

}
```

Configuration class

You can also configure the valid PIN code by using the adapter.xml file and the MobileFirst Operations Console.

Create a new Java class that extends `CredentialsValidationSecurityCheckConfig`. It is important to extend a class that matches the parent security check class, in order to inherit the default configuration.

```
public class PinCodeConfig extends CredentialsValidationSecurityCheckConfig {

    public String pinCode;

    public PinCodeConfig(Properties properties) {
        super(properties);
        pinCode = getStringProperty("pinCode", properties, "1234");
    }

}
```

The only required method in this class is a constructor that can handle a `Properties` instance. Use the `get[Type]Property` method to retrieve a specific property from the adapter.xml file. If no value is found, the third parameter defines a default value (`1234`).

You can also add error handling in this constructor by using the `addMessage` method:

```
public PinCodeConfig(Properties properties) {
    //Make sure to load the parent properties
    super(properties);

    //Load the pinCode property
    pinCode = getStringProperty("pinCode", properties, "1234");

    //Check that the PIN code is at least 4 characters long. Triggers an error.
    if(pinCode.length() < 4) {
        addMessage(errors,"pinCode","pinCode needs to be at least 4 characters");
    }

    //Check that the PIN code is numeric. Triggers warning.
    try {
        int i = Integer.parseInt(pinCode);
    }
    catch(NumberFormatException nfe) {
        addMessage(warnings,"pinCode","PIN code contains non-numeric characters");
    }
}
```

In your main class (`PinCodeAttempts`), add the following two methods to be able to load the configuration:

```
@Override
public SecurityCheckConfiguration createConfiguration(Properties properties) {
    return new PinCodeConfig(properties);
}
@Override
protected PinCodeConfig getConfiguration() {
    return (PinCodeConfig) super.getConfiguration();
}
```

You can now use the `getConfiguration().pinCode` method to retrieve the default PIN code.

You can modify the `validateCredentials` method to use the PIN code from the configuration instead of the hardcoded value.

```
@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
   if(credentials!=null && credentials.containsKey(PINCODE_FIELD)){
      String pinCode = credentials.get(PINCODE_FIELD).toString();

      if(pinCode.equals(getConfiguration().pinCode)){
         return true;
      }
      else {
         errorMsg = "Pin code is not valid. Hint: " + getConfiguration().pinCode;
      }

   }
   else{
      errorMsg = "The pin code was not provided.";
   }

   //In any other case, credentials are not valid
   return false;

}
```

Configuring the security check

In your adapter.xml, add a `<securityCheckDefinition>` element:

```
<securityCheckDefinition name="PinCodeAttempts" class="com.sample.PinCodeAttempts">
  <property name="pinCode" defaultValue="1234" description="The valid PIN code"/>
  <property name="maxAttempts" defaultValue="3" description="How many attempts are allowed"/>
  <property name="blockedStateExpirationSec" defaultValue="60" description="How long before the client
can try again (seconds)"/>
  <property name="successStateExpirationSec" defaultValue="60" description="How long is a successful
state valid for (seconds)"/>
</securityCheckDefinition>
```

The `name` attribute must the name of the security check. Set the `class` parameter to the class that you created previously.

A `securityCheckDefinition` can contain zero or more `property` elements. The `pinCode` property is the one defined in the `PinCodeConfig` configuration class. The other properties are inherited from the `CredentialsValidationSecurityCheckConfig` configuration class.

By default, if you do not specify those properties in the adapter.xml file, you receive the default values that are set by `CredentialsValidationSecurityCheckConfig`:

```
public CredentialsValidationSecurityCheckConfig(Properties properties) {
   super(properties);
   maxAttempts = getIntProperty("maxAttempts", properties, 1);
   attemptingStateExpirationSec = getIntProperty("attemptingStateExpirationSec", properties, 120);
   successStateExpirationSec = getIntProperty("successStateExpirationSec", properties, 3600);
   blockedStateExpirationSec = getIntProperty("blockedStateExpirationSec", properties, 0);
}
```

The `CredentialsValidationSecurityCheckConfig` class defines the following properties:

- `maxAttempts`: How many attempts are allowed before reaching a *failure*.
- `attemptingStateExpirationSec`: Interval in seconds during which the client must provide valid credentials, and attempts are counted.
- `successStateExpirationSec`: Interval in seconds during which the successful login holds.
- `blockedStateExpirationSec`: Interval in seconds during which the client is blocked after reaching `maxAttempts`.

Note that the default value for `blockedStateExpirationSec` is set to `0`: if the client sends invalid credentials, it can try again "after 0 seconds". This means that by default the "attempts" feature is disabled.

Sample Security Check

Download (https://github.com/MobileFirst-Platform-Developer-Center/SecurityCheckAdapters/tree/release80) the Security Checks Maven project.

The Maven project contains an implementation of CredentialsValidationSecurityCheck.