Implementing the UserAuthenticationSecurityCheck Class

Overview

This abstract class extends `CredentialsValidationSecurityCheck` and builds upon it to fit the most common use-cases of simple user authentication. In addition to validating the credentials, it creates a **user identity** that is accessible from various parts of the framework, allowing you to identify the current user. Optionally, `UserAuthenticationSecurityCheck` also provides **Remember Me** capabilities.

This tutorial uses the example of a security check that asks for a user name and password, and uses the user name to represent an authenticated user.

Prerequisites: Make sure to read the CredentialsValidationSecurityCheck (../../credentials-validation/) tutorial.

Jump to:

Creating the Security Check

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

```
public class UserLogin extends UserAuthenticationSecurityCheck {

    @Override
    protected AuthenticatedUser createUser() {
        return null;
    }

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

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

Creating the challenge

The challenge is exactly the same as the one described in Implementing the CredentialsValidationSecurityCheck (../../credentials-validation/security-check/).

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

Validating the user credentials

When the client sends the challenge answer, the answer is passed to `validateCredentials` as a `Map`. Use this method to implement your logic. The method returns `true` if the credentials are valid.

In this example, credentials are considered “valid” when `username` and `password` are the same:

```
@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
    if(credentials!=null && credentials.containsKey("username") && credentials.co
ntainsKey("password")){
        String username = credentials.get("username").toString();
        String password = credentials.get("password").toString();
        if(!username.isEmpty() && !password.isEmpty() && username.equals(passwor
d)) {
            return true;
        }
        else {
            errorMsg = "Wrong Credentials";
        }
    }
    else{
        errorMsg = "Credentials not set properly";
    }
    return false;
}
```

Creating the AuthenticatedUser object

The `UserAuthenticationSecurityCheck` class stores a representation of the current client (user, device, application) in persistent data, allowing you to retrieve the current user in various parts of your code, such as the challenge handlers or the adapters. Users are represented by an instance of the class `AuthenticatedUser`. Its constructor takes the `id`, `displayName`, and `securityCheckName` parameters.

This example uses `username` for both the `id` and `displayName` parameters.

1. First, modify the `validateCredentials` method to save the `username` argument:

```
private String userId, displayName;

@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
     if(credentials!=null && credentials.containsKey("username") && credenti
als.containsKey("password")){
         String username = credentials.get("username").toString();
         String password = credentials.get("password").toString();
         if(!username.isEmpty() && !password.isEmpty() && username.equals(p
assword)) {
             userId = username;
             displayName = username;
             return true;
         }
         else {
             errorMsg = "Wrong Credentials";
         }
     }
     else{
         errorMsg = "The credentials are not set properly.";
     }
     return false;
}
```

2. Then, override the `createUser` method to return a new instance of `AuthenticatedUser`:

```
@Override
protected AuthenticatedUser createUser() {
     return new AuthenticatedUser(userId, displayName, this.getName());
}
```

You can use `this.getName()` to get the current security check name.

`UserAuthenticationSecurityCheck` calls your `createUser()` implementation after a successful `validateCredentials`.

Storing attributes in the AuthenticatedUser

`AuthenticatedUser` has an alternate constructor:

```
AuthenticatedUser(String id, String displayName, String securityCheckName, Map
<String, Object> attributes);
```

This constructor adds a `Map` of custom attributes to be stored with the user representation. The map can be used to store additional information such as a profile picture, a website, etc. This information is accessible to the client side (challenge handler) and the resource (using introspection data).

> **Note:** The attributes `Map` must contain only objects of types/classes bundled in the Java library (such as `String`, `int`, `Map`, etc), and **not** custom classes.

Adding RememberMe functionality

By default, `UserAuthenticationSecurityCheck` uses the `successStateExpirationSec` property to determine how long the success state lasts. This property is inherited from `CredentialsValidationSecurityCheck`.

If you want to allow users to stay logged-in past the `successStateExpirationSec` value, `UserAuthenticationSecurityCheck` adds this capability.

`UserAuthenticationSecurityCheck` adds a property called `rememberMeDurationSec` whose default value is `0`: by default, users are remembered for **0 seconds**, which means that by default, the feature is disabled. Change this value to a number that makes sense for your application (a day, a week, a month…).

You can also manage the feature by overriding the `rememberCreatedUser()` method, which returns `true` by default, meaning that the feature is active by default (provided that you changed the duration property).

In this example, the client decides to enable/disable the **RememberMe** feature by sending a `boolean` value as part of the submitted credentials.

1. First, modify the `validateCredentials` method to save the `rememberMe` choice:

```
private String userId, displayName;
private boolean rememberMe = false;

@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
     if(credentials!=null && credentials.containsKey("username") && credenti
als.containsKey("password")){
         String username = credentials.get("username").toString();
         String password = credentials.get("password").toString();
         if(!username.isEmpty() && !password.isEmpty() && username.equals(p
assword)) {
             userId = username;
             displayName = username;

             //Optional RememberMe
             if(credentials.containsKey("rememberMe") ){
                 rememberMe = Boolean.valueOf(credentials.get("rememberMe").
toString());
             }

             return true;
         }
         else {
             errorMsg = "Wrong Credentials";
         }
     }
     else{
         errorMsg = "Credentials not set properly";
     }
     return false;
}
```

2. Then, override the `rememberCreatedUser()` method:

```
@Override
protected boolean rememberCreatedUser() {
     return rememberMe;
}
```

Configuring the security check

In the **adapter.xml** file, add a `<securityCheckDefinition>` element:

```
<securityCheckDefinition name="UserLogin" class="com.sample.UserLogin">
  <property name="maxAttempts" defaultValue="3" description="How many attempts
are allowed."/>
  <property name="blockedStateExpirationSec" defaultValue="10" description="How l
ong before the client can try again (seconds)."/>
  <property name="successStateExpirationSec" defaultValue="60" description="How l
ong is a successful state valid for (seconds)."/>
  <property name="rememberMeDurationSec" defaultValue="120" description="How long
is the user remembered by the RememberMe feature (seconds)."/>
</securityCheckDefinition>
```

As mentioned previously, `UserAuthenticationSecurityCheck` inherits all the `CredentialsValidationSecurityCheck` properties, such as `blockedStateExpirationSec`, `successStateExpirationSec`, etc.

In addition, you can also configure a `rememberMeDurationSec` property.

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 `UserAuthenticationSecurityCheck`.

Last modified on