Implementing the UserAuthenticationSecurityCheck

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 will be 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 asking for a username and password and uses the username 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() {
    HashMap challenge = new HashMap();
    challenge.put("errorMsg",errorMsg);
    challenge.put("remainingAttempts",remainingAttempts);
    return challenge;
}
```

Validating the user credentials

When the client sends the challenge's answer, the answer is passed to `validateCredentials` as a `Map`. This method should implement your logic and return `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.containsKey("password")){
        String username = credentials.get("username").toString();
        String password = credentials.get("password").toString();
        if(!username.isEmpty() && !password.isEmpty() && username.equals(password)) {
            return true;
        }
        else {
            errorMsg = "Wrong Credentials";
        }
    }
    else{
        errorMsg = "Credentials not set properly";
    }
    return false;
}
```

Creating the AuthenticatedUser object

The `UserAuthenticationSecurityCheck` 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 receives a `id`, `displayName` and `securityCheckName`.

In this example, we are using the `username` for both the `id` and `displayName`.

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

```
private String userId, displayName;

@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
   if(credentials!=null && credentials.containsKey("username") && credentials.containsKey("passwor
d")){
      String username = credentials.get("username").toString();
      String password = credentials.get("password").toString();
      if(!username.isEmpty() && !password.isEmpty() && username.equals(password)) {
         userId = username;
         displayName = username;
         return true;
      }
      else {
         errorMsg = "Wrong Credentials";
      }
   }
   else{
      errorMsg = "Credentials 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` will call your `createUser()` implementation after a successful `validateCredentials`.

Adding Remember Me functionality

`UserAuthenticationSecurityCheck` by default uses the `successStateExpirationSec` property to determine how long does the success state last; this property was inherited from `CredentialsValidationSecurityCheck`.

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

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

The feature is also managed by overriding the method `rememberCreatedUser()`, which returns `true` by default. Meaning the feature is active by default (granted you changed the duration property).

In this example, the client decides to enable/disable the remember me feature by sending a `boolean` 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") && credentials.containsKey("passwor
d")){
      String username = credentials.get("username").toString();
      String password = credentials.get("password").toString();
      if(!username.isEmpty() && !password.isEmpty() && username.equals(password)) {
         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 SecurityCheck

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

```
<securityCheckDefinition name="UserLogin" class="com.sample.UserLogin">
  <property name="maxAttempts" defaultValue="3" displayName="How many attempts are allowed"/>
  <property name="failureStateExpirationSec" defaultValue="10" displayName="How long before the client
can try again (seconds)"/>
  <property name="successStateExpirationSec" defaultValue="60" displayName="How long is a successful
state valid for (seconds)"/>
  <property name="rememberMeDurationSec" defaultValue="120" displayName="How long is the user rem
embered when using RememberMe (seconds)"/>
</securityCheckDefinition>
```

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

In addition, a `rememberMeDurationSec` property can also be configured.

Sample application

To see a sample using this security check, review the below tutorials.
Select a platform:

- Implementing the challenge handler in Cordova applications (../cordova)
- Implementing the challenge handler in iOS applications (../ios)
- Implementing the challenge handler in Android applications (../android)
- Implementing the challenge handler in Windows 8.1 Universal and Windows 10 UWP applications (../windows-8-10)