Implementing the challenge handler in Android applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/authentication-and-security/security-check-with-attempts/android/index.md) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

When trying to access a protected resource, the server (the `SecurityCheck`) will send back to the client a list containing one or more **challenges** for the client to handle.
This list is received as a `JSON` object, listing the `SecurityCheck` name with an optional `JSON` of additional data:

```
{
  "challenges": {
    "SomeSecurityCheck1":null,
    "SomeSecurityCheck2":{
      "some property": "some value"
    }
  }
}
```

The client should then register a **challenge handler** for each `SecurityCheck`.
The challenge handler defines the client-side behavior that is specific to the security check.

Creating the challenge handler

A challenge handler is a class responsible for handling challenges sent by the MobileFirst server, such as displaying a login screen, collecting credentials and submitting them back to the `SecurityCheck`.

In this example, the `SecurityCheck` is `PinCodeAttempts` which was defined in Implementing Security Check with Attempts Security Adapter (../adapter). The challenge sent by this `SecurityCheck` contains the number of remaining attempts to login (`remainingAttempts`), and an optional `errorMsg`.

Create a Java class that extends `WLChallengeHandler`:

```
public class PinCodeChallengeHandler extends WLChallengeHandler {

}
```

Handling the challenge

The minimum requirement from the `WLChallengeHandler` protocol is to implement a constructor and a `handleChallenge` method, that is responsible for asking the user to provide the credentials. The `handleChallenge` method receives the challenge as a `JSONObject`.

> Learn more about the `WLChallengeHandler` protocol in the user documentation.

Add a constructor method:

```
public PinCodeChallengeHandler(String securityCheck) {
    super(securityCheck);
}
```

In this `handleChallenge` example, an alert is displayed asking to enter the PIN code:

```
@Override
public void handleChallenge(JSONObject jsonObject) {
    try{
        if (jsonObject.isNull("errorMsg")){
            alertMsg("This data requires a PIN code.\n Remaining attempts: " + jsonObject.getString("remaining
Attempts"));
        } else {
            alertMsg(jsonObject.getString("errorMsg") + "\nRemaining attempts: " + jsonObject.getString("remai
ningAttempts"));
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

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

If the credentials are incorrect, you can expect the framework to call `handleChallenge` again.

Submitting the challenge's answer

Once the credentials have been collected from the UI, use the `WLChallengeHandler`'s `submitChallengeAnswer(JSONObject answer)` method to send an answer back to the `SecurityCheck`. In this example `PinCodeAttempts` expects a property called `pin` containing the submitted PIN code:

```
submitChallengeAnswer(new JSONObject().put("pin", pinCodeTxt.getText()));
```

Cancelling the challenge

In some cases, such as clicking a "Cancel" button in the UI, you want to tell the framework to discard this challenge completely.

To achieve this, call:

```
submitFailure(null);
```

Handling failures

Some scenarios may trigger a failure (such as maximum attempts reached). To handle these, implement the `WLChallengeHandler`'s `handleFailure` method.
The structure of the `JSONObject` passed as a parameter greatly depends on the nature of the failure.

```
@Override
public void handleFailure(JSONObject jsonObject) {
    try {
        if (!jsonObject.isNull("failure")) {
            alertError(jsonObject.getString("failure"));
        } else {
            alertError("Unknown error");
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

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

Handling successes

In general successes are automatically processed by the framework to allow the rest of the application to continue.

Optionally you can also choose to do something before the framework closes the challenge handler flow, by implementing the `WLChallengeHandler`'s `handleSuccess` method. Here again, the content and structure of the `JSONObject` passed as a parameter depends on what the `SecurityCheck` sends.

In the `PinCodeAttempts` sample application, the `JSONObject` does not contain any additional data and so `handleSuccess` is not implemented.

Registering the challenge handler

In order for the challenge handler to listen for the right challenges, you must tell the framework to associate the challenge handler with a specific `SecurityCheck` name.

This is done by initializing the challenge handler with the `SecurityCheck` like this:

```
PinCodeChallengeHandler pinCodeChallengeHandler = new PinCodeChallengeHandler("PinCodeAttempts"
, this);
```

You must then **register** the challenge handler instance:

```
WLClient client = WLClient.createInstance(this);
client.registerChallengeHandler(pinCodeChallengeHandler);
```

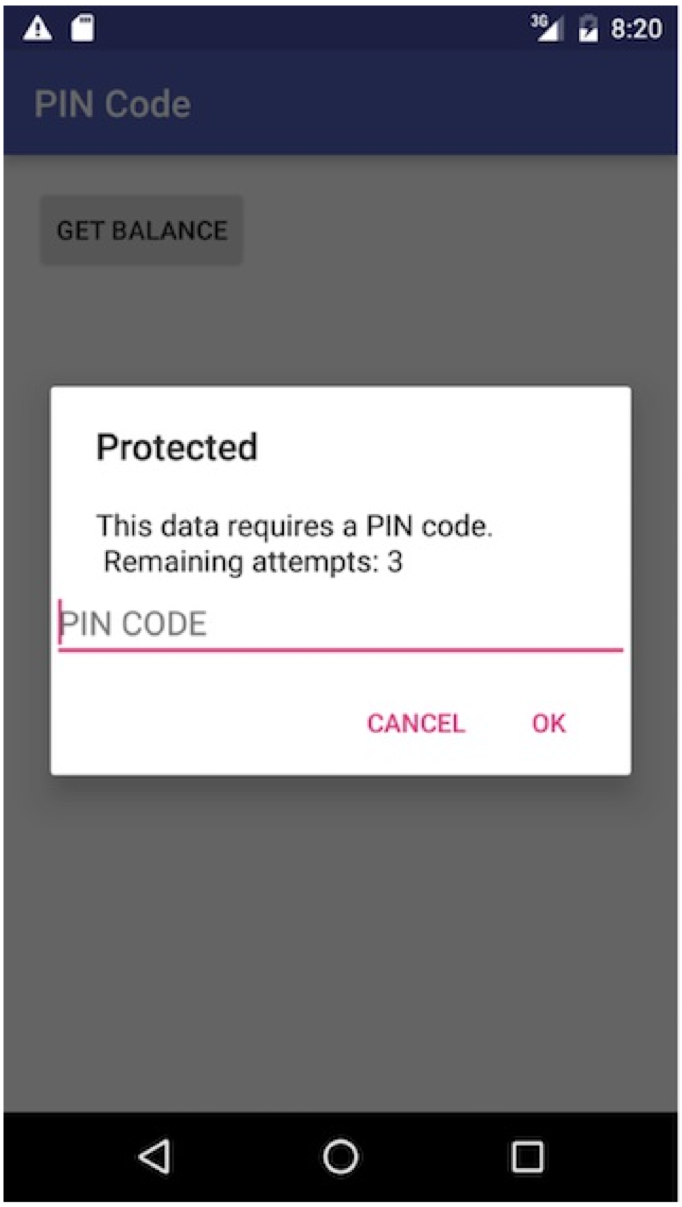
Sample application

The sample **PinCodeAndroid** is an Android application that uses `WLResourceRequest` to get a bank balance.
The method is protected with a PIN code, with a maximum of 3 attempts.

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/SecurityAdapters/tree/release80) the SecurityAdapters Maven project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/PinCodeAndroid/tree/release80) the Android Native project.

Sample usage

- Use either Maven or MobileFirst Developer CLI to build and deploy the available `ResourceAdapter` and `PinCodeAttempts` adapters (../../creating-adapters/).
- Ensure the sample is registered in the MobileFirst Server by running the command: `mfpdev app register`.
- In the MobileFirst console, under **Applications → PIN Code → Security → Map scope elements to security checks.**, add a mapping from `accessRestricted` to `PinCodeAttempts`.


PIN Code

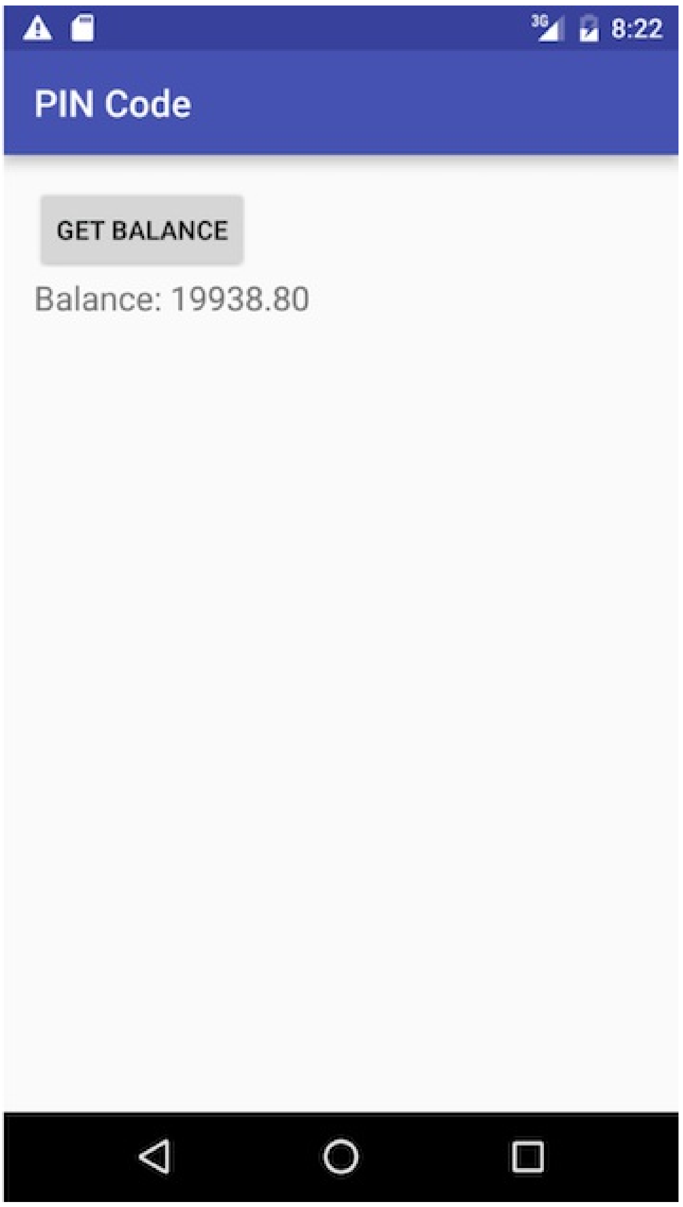
GET BALANCE

Protected

This data requires a PIN code.
Remaining attempts: 3

PIN CODE

CANCEL OK


PIN Code

GET BALANCE

Balance: 19938.80