

Adapter-based authentication in native Android applications

Overview

This tutorial explains how to implement the client-side of adapter-based authentication in native Android.

Prerequisite: Make sure that you read the Adapter-based authentication (../) tutorial first.

Implementing the client-side authentication

- Create a native Android application and add the MobileFirst native APIs as explained in the Configuring a native Android application with the MobileFirst Platform SDK (../hello-world/configuring-a-native-android-application-with-the-mfp-sdk/) tutorial.
- Add an activity which handles and presents a login form.



Challenge Handler

- Create a MyChallengeHandler class as a subclass of ChallengeHandler.

```
1 | public class AndroidChallengeHandler extends ChallengeHandler
```

- Call the super method:

```

1 public AndroidChallengeHandler(String realm) {
2     super(realm);
3 }

```

- Add an implementation of the following ChallengeHandler methods to handle the form-based challenge:

1. isCustomResponse method:

The isCustomResponse method is invoked each time a response is received from the MobileFirst Server. It is used to detect whether the response contains data that is related to this challenge handler. It must return either true or false.

```

1 public boolean isCustomResponse(WLResponse response) {
2     try {
3         if(response!= null &&
4             response.getResponseJSON()!=null &&
5             !response.getResponseJSON().isNull("authStatus") &&
6             response.getResponseJSON().getString("authStatus") != ""){
7             return true;
8         }
9     } catch (JSONException e) {
10        e.printStackTrace();
11    }
12    return false;
13 }

```

2. handleChallenge method:

If isCustomResponse returns true, the framework calls the handleChallenge method. This function is used to perform required actions, such as hiding the application screen and showing the login screen.

```

1 public void handleChallenge(WLResponse response){
2     cachedResponse = response;
3     try {
4         if(response.getResponseJSON().getString("authStatus").equals("credentialsRequired")){
5             MainAdapterBasedAuth.setMainText("handleChallenge->credentialsRequired");
6             Intent login = new Intent(parentActivity, LoginAdapterBasedAuth.class);
7             parentActivity.startActivityForResult(login, 1);
8         }
9         else if(response.getResponseJSON().getString("authStatus").equals("complete")){
10            submitSuccess(cachedResponse);
11        }
12    } catch (JSONException e) {
13        // TODO Auto-generated catch block
14        e.printStackTrace();
15    }
16 }

```

3. onSuccess and onFailure methods:

At the end of the authentication flow, `onSuccess` or `onFailure` will be triggered. Call the `submitSuccess` method in order to inform the framework that the authentication process completed successfully and for the `onSuccess` handler of the invocation to be called. Call the `submitFailure` method in order to inform the framework that the authentication process failed and for the `onFailure` handler of the invocation to be called.

```
1 public void onFailure(WLFailResponse response) {
2     submitFailure(response);
3 }
4 public void onSuccess(WLResponse response) {
5     submitSuccess(response);
6 }
```

submitLoginForm

When the user taps to submit the credentials, call the `submitLoginForm` method to send the credentials to the adapter procedure.

For example, in here we implemented a `submitLogin` method that called by the `MainActivity` after the login process is completed.

```
1 public void submitLogin(int resultCode, String userName, String password, boolean back){
2     if (resultCode != Activity.RESULT_OK || back) {
3         submitFailure(cachedResponse);
4     } else {
5         Object[] parameters = new Object[]{userName, password};
6         WLProcedureInvocationData invocationData = new WLProcedureInvocationData("AuthAdapter", "submit
7         invocationData.setParameters(parameters);
8         WLRequestOptions options = new WLRequestOptions();
9         options.setTimeout(30000);
10        submitAdapterAuthentication(invocationData, options);
11    }
12 }
```

The Main Activity

In the sample project, in order to trigger the challenge handler we use the `WLClient` `invokeProcedure` method.

The protected procedure invocation triggers MobileFirst Server to send the challenge.

- Create a `WLClient` instance and use the `connect` method to connect to the MobileFirst Server:

```
1 final WLClient client = WLClient.createInstance(this);
2 client.connect(new MyConnectionListener());
```

- In order to listen to incoming challenges, make sure to register the challenge handler by using the

registerChallengeHandler method:

```
1 challengeHandler = new AndroidChallengeHandler(this, realm);
2 client.registerChallengeHandler(challengeHandler);
```

- Invoke the protected adapter procedure:

```
1 URI adapterPath = new URI("/adapters/AuthAdapter/getSecretData");
2 WLResourceRequest request = new WLResourceRequest(adapterPath,WLResourceRequest.GET)
3 request.send(new MyResponseListener());
```

Sample application

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/AdapterBasedAuth>) the MobileFirst project.

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/AdapterBasedAuthAndroid>) the Native project.

- The AdapterBasedAuth project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The AdapterBasedAuthAndroid project contains a native Android application that uses a MobileFirst native API library.
- Make sure to update the `worklight.plist` file in the native project with the relevant server settings.

