Custom Authentication in native Android applications

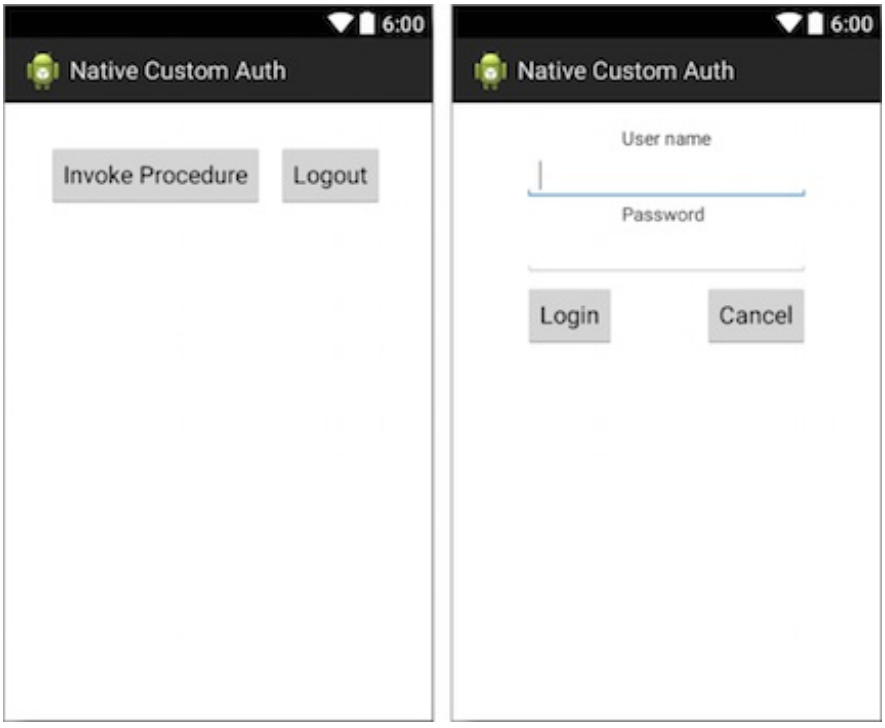
fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.1/authentication-security/custom-authentication/custom-authentication-in-native-android-applications.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

This tutorial explains how to implement the client side of a custom authentication in native Android.
Prerequisite: Make sure that you read the Custom 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`.

```
public class AndroidChallengeHandler extends ChallengeHandler
```

- Call the `super` method:

```
public AndroidChallengeHandler(String realm) {
 super(realm);
}
```

- 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`.

     ```
     public boolean isCustomResponse(WLResponse response) {
      if (response == null || response.getResponseJSON() == null) {
       return false;
      }
      if(response.toString().indexOf("authStatus") > -1){
       return true;
      }
      else{
       return false;
      }
     }
     ```

  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.

     ```
     public void handleChallenge(WLResponse response){
      try {
       if(response.getResponseJSON().getString("authStatus") == "complete"){
        submitSuccess(response);
       }
       else {
        cachedResponse = response;
        Intent login = new Intent(parentActivity, LoginCustomLoginModule.class)
     ;
        parentActivity.startActivityForResult(login, 1);
       }
      } catch (JSONException e) {
       e.printStackTrace();
      }
     }
     ```

  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.

```
public void onFailure(WLFailResponse response) {
 submitFailure(response);
}
public void onSuccess(WLResponse response) {
 submitSuccess(response);
}
```

submitLoginForm

When the user taps to submit the credentials, call the `submitLoginForm` method to send the the credentials to the MobileFirst Server.
For example, in here we implemented a `submitLogin method` that called by the MainActivity after the login process is completed.

```
public void submitLogin(int resultCode, String userName, String password, boolean back){
 if (resultCode != Activity.RESULT_OK || back) {
  submitFailure(cachedResponse);
 } else {
  HashMap<String, String> params = new HashMap<String, String>();
  params.put("username", userName);
  params.put("password", password);
  submitLoginForm("/my_custom_auth_request_url", params, null, 0, "post");
 }
}
```

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:

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

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

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

- Invoke the protected adapter procedure:

```
URI adapterPath = new URI("/adapters/AuthAdapter/getSecretData");
WLResourceRequest request = new WLResourceRequest(adapterPath,WLResourceRequest.GE
T);
request.send(new MyResponseListener());
```

Worklight Protocol

If your custom authenticator uses `WorklightProtocolAuthenticator`, some simplifications can be made:

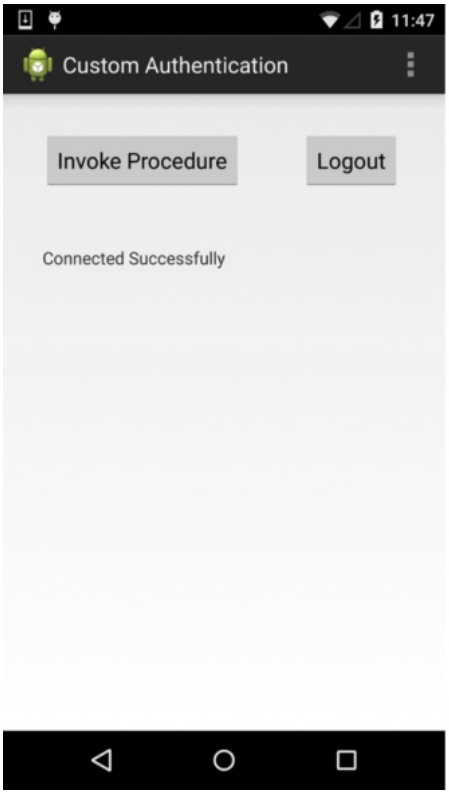
- Subclass your challenge handler using `WLChallengeHandler` instead of `ChallengeHandler`. Note the `WL`.
- You no longer need to implement `isCustomResponse` as the challenge handler will automatically check that the realm name matches.
- `handleChallenge` will receive the challenge as a parameter, not the entire response object.
- Instead of `submitLoginForm`, use `submitChallengeAnswer` to send your challenge response as a JSON.
- There is no need to call `submitSuccess` or `submitFailure` as the framework will do it for you.

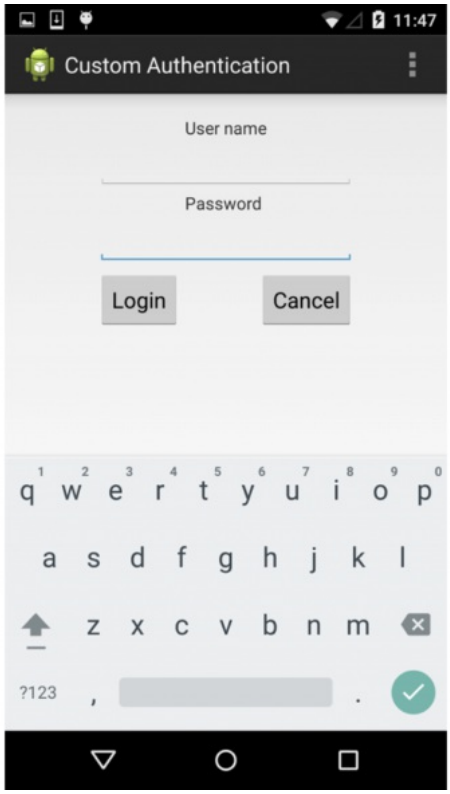
> For an example that uses `WorklightProtocolAuthenticator`, see the Remember Me (../../../advanced-topics/remember-me/) tutorial or this video blog post (file:////home/travis/build/MFPSamples/DevCenter/_site/blog/2015/05/29/ibm-mobilefirst-platform-foundation-custom-authenticators-and-login-modules/).

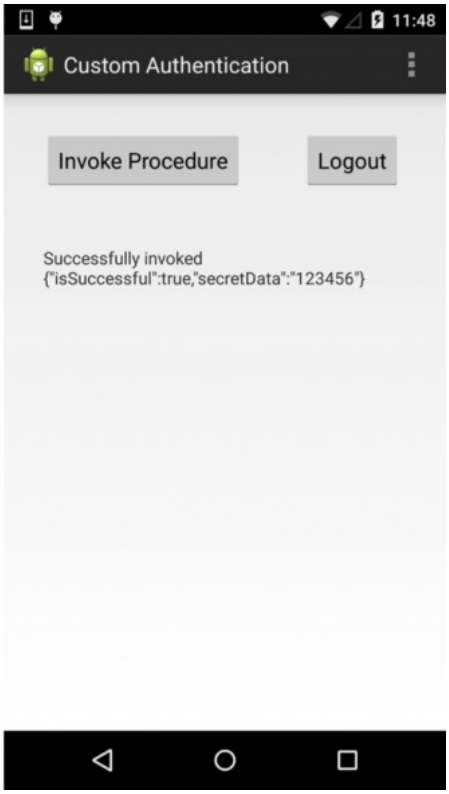
Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomAuth/tree/release71) the MobileFirst project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomAuthAndroid/tree/release71) the Native project.

- The `CustomAuth` project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The `CustomAuthAndroid` 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.


11:47
Custom Authentication
Invoke Procedure
Logout
Connected Successfully


11:47
Custom Authentication
User name
Password
Login
Cancel


11:48
Custom Authentication
Invoke Procedure
Logout
Successfully invoked
{"isSuccessful":true,"secretData":"123456"}