

Form-based authentication in native Android applications

Relevant to:



Native Android

Overview

This tutorial illustrates the native Android client-side authentication components for form-based authentication.

Prerequisite: Make sure that you read [Form-based authentication](#) first.

This tutorial covers the following topics:

- [Creating the client-side authentication components](#)
- [Sample application](#)

Creating the client-side authentication components

1. Create a native Android application and add the MobileFirst native APIs as explained in the documentation.
2. Add an activity, `LoginFormBasedAuth`, which handles and presents the login form.
3. Remember to add this activity to the `AndroidManifest.xml` file, too.

MyChallengeHandler

Create a `MyChallengeHandler` class as a subclass of `ChallengeHandler`.

Your `MyChallengeHandler` class must implement `isCustomResponse`, which checks every custom response received from MobileFirst Server to verify whether this is the expected challenge.

```
public boolean isCustomResponse(WLResponse response) {
    if (response == null || response.getResponseText() == null ||
        response.getResponseText().indexOf("j_security_check") ==
-1) {
        return false;
    }
    return true;
}
```

The `handleChallenge` method is called after the `isCustomResponse` method returns `true`. Here this method presents the login form.

```
public void handleChallenge(WLResponse response){
    if (!isCustomResponse(response)) {
        submitSuccess(response);
    } else {
```

```

        cachedResponse = response;
        Intent login = new Intent(parentActivity,
LoginFormBasedAuth.class);
        parentActivity.startActivityForResult(login, 1);
    }
}

```

The `submitLogin` method is called by the login form. If the user asked to abort this action, use the `submitFailure()` method, otherwise use the `submitLoginForm()` method to send input data to the authenticator.

```

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("j_username", userName);
        params.put("j_password", password);
        submitLoginForm("/j_security_check", params, null, 0, "post");
    }
}

```

Main activity

In the `MainActivity` class, connect to MobileFirst Server, register your `challengeHandler` object, and invoke the protected adapter procedure.

The procedure invocation triggers MobileFirst Server to send a challenge that will trigger the challenge handler.

```

final WLClient client = WLClient.createInstance(this);

client.connect(new MyConnectionListener());

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

invokeBtn = (Button) findViewById(R.id.invoke);
invokeBtn.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        //setMainText("Invoking...");
        WLProcedureInvocationData invocationData = new
WLProcedureInvocationData("DummyAdapter", "getSecretData");
        WLRequestOptions options = new WLRequestOptions();
        options.setTimeout(30000);
        client.invokeProcedure(invocationData, new
MyResponseListener(), options);
    }

});

```

Sample application

[Click to download](#) the MobileFirst project.

[Click to download](#) the Native project.

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

