MobileFirst Platform {dev}

Double-step adapter authentication

Relevant to:
Hybrid

Overview

This tutorial demonstrates how to implement “double-step” adapter-based authentication.

Double-step means that after the initial authentication that uses, for example, a username and a password, an additional authentication step is required, such as a login pin, a secret word, or similar identification. In this example, a secret word is implemented for the second authentication step. The code snippets and sample application in this tutorial are based on the existing adapter-based authentication sample. The changes extend the application from *single-step* to *double-step*.

Session-independent mode

By default, MobileFirst Platform Foundation 7.1 applications run in a session-independent mode, meaning that you can no longer use HTTP sessions or global variables to persist data across requests. Instead, MobileFirst apps must use a third-party database to store applicative states.

> To learn more about the session-independent mode, see its topic in the user documentation.

To demonstrate how to store user data, the tutorial uses the `WL.Server.getClientId` API and a Cloudant database.

Agenda

Prerequisite – Creating an IBM Cloudant account

This sample uses IBM Cloudant Database to save user data. To run the sample and understand how to work with Cloudant, first sign up for a free account and create a database.
Then proceed as follows:

- Change the database permissions – Follow the instructions in the Changing Database Permissions tutorial.
- Basic authentication – The basic authentication value is passed as part of every request to the database. Instead of using your username and password to identify, use base-64 encoding to generate a string that is created by concatenating the API `key` and `password`, separated by a column character in the following manner: `key:password`. You use it later to send requests to the database.
  For more information, read the Cloudant Basic Authentication documentation.

Configuring the authenticationConfig.xml file

Realms

Add a realm or replace the existing `AuthLoginModule` realm in the `realms` section of the `authenticationConfig.xml` file:

```
<realm loginModule="AuthLoginModule" name="DoubleStepAuthRealm">
  <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="AuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="AuthAdapter.onLogout"/>
</realm>
```

Security tests

Add a security test or replace the existing `AuthSecurityTest` in the `securityTests` section of the `authenticationConfig.xml` file:

```
<customSecurityTest name="DoubleStepAuthAdapter-securityTest">
    <test isInternalUserID="true" realm="DoubleStepAuthRealm"/>
</customSecurityTest>
```

> To review the remaining/existing sample components, see the Adapter-based authentication tutorial.

Creating the server-side authentication components

To put in place the double-step authentication process, several changes are necessary to the adapter file (whether XML or JavaScript) and to the database.

Adapter XML file

Edit the `AuthAdapter.xml` file:

1. Change the domain name to your Cloudant domain:

```
<domain>$USERNAME.cloudant.com</domain>
```

2. Add the following procedure:

```
<procedure name="submitAuthenticationStep2" securityTest="wl_unprotected"/>
```

3. Protect the `getSecretData` method with the new `DoubleStepAuthAdapter-securityTest`

Adapter JavaScript file

Edit the `AuthAdapter-impl.js` file:

1. Create a variable to save the basic authentication encoded string you have generated before:

```
var auth = "Basic REPLASE_ME_WITH_THE_BASE-64_ENCODED_STRING";;
```

2. Create a variable to save your database name:

```
var dbName = "REPLACE_ME_WITH_THE_DATABASE_NAME";
```

3. Update the `onAuthRequired` function to return that authentication step 1 is required:

```
function onAuthRequired(headers, errorMessage){
    errorMessage = errorMessage ? errorMessage : null;
    return {
        authRequired: true,
        authStep: 1,
        errorMessage: errorMessage
    };
}
```

4. Update the `submitAuthenticationStep1` function:
   - Add the following line to get the client ID:

```
function submitAuthenticationStep1(username, password){
    if (username === "user" && password === "password"){
        WL.Logger.debug("Step 1 :: SUCCESS");
        var clientId = WL.Server.getClientId();
        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {}
        };
```

   - To save the `userIdentity` for the next authentication step, write it to the database. Use the `clientId` variable as the document `_id` key:

```
//Validate that the DB doesn't already contains the ClientId
var response = deleteUserIdentityFromDB(dbName, null);

//Write ClientId to DB
var response = writeUserIdentityToDB(dbName, {_id:clientId, "userIdentity":userIdentity});
```

   - If step 1 authentication was successful, return that step 2 is required:

```
        if (response){
            return {
                authRequired: true,
                authStep: 2,
                question: "What is your pet's name?",
                errorMessage : ""
            };
        } else {
            return onAuthRequired(null, "Database ERROR");
        }
```

```
        } else{
            WL.Logger.debug("Step 1 :: FAILURE");
            return onAuthRequired(null, "Invalid login credentials");
        }
    }
```

5. Add `submitAuthenticationStep2` function to handle the second authentication step:
   - Get the client ID and read it from the database:

```
    function submitAuthenticationStep2(answer){
        var clientId = WL.Server.getClientId();
        var response = readUserIdentityFromDB(dbName, clientId);
```

   - If step 2 authentication was successful, delete the client document from database:

```
        if (response){
            if (answer === "Lassie"){
                var doc = JSON.parse(response.text);
                var userIdentity = doc.userIdentity;
                WL.Logger.debug("Step 2 :: SUCCESS");
                WL.Server.setActiveUser("DoubleStepAuthRealm", userIdentity);
                WL.Logger.debug("Authorized access granted");

                var response = deleteUserIdentityFromDB(dbName, doc);

                return {
                    authRequired: false
                };
            } else{
                WL.Logger.debug("Step 2 :: FAILURE");
                return onAuthRequired(null, "Wrong security question answer");
            }
        } else {
            WL.Logger.debug("Step 1 :: FAILURE");
            return onAuthRequired(null, "Database ERROR");
        }

    }
```

Database actions

To handle the database actions, use the `WL.Server.invokeHttp` method and Cloudant REST API.

- Write to the database:

```
function writeUserIdentityToDB(db, document){
       var input = {
                method : 'post',
                returnedContentType : 'plain',
                path : db,
                headers: {
                    "Authorization":auth
                },
                body:{
                    contentType:'application/json; charset=UTF-8',
                    content:JSON.stringify(document)
               }};

         var response = WL.Server.invokeHttp(input);
         var responseString = "" + response.statusCode;

        //Checking if the invocation was successful - status code = 2xx
         if (responseString.indexOf('2') === 0){
             return response;
         }
         return null;
}
```

- Read from database:

```
function readUserIdentityFromDB(db, key){
     var input = {
                method : 'get',
                returnedContentType : 'plain',
                path : db + "/" + key,
                headers: {
                    "Authorization":auth
```

```
                    }
                };
        var response = WL.Server.invokeHttp(input);
        var responseString = "" + response.statusCode;

        //Checking if the invocation was successful - status code = 2xx
        if (responseString.indexOf('2') === 0){
            return response;
        }
        return null;
    }
```

- Delete from the database:

```
    function deleteUserIdentityFromDB(db, document){
        var doc = document;
        if (!doc){
            var clientId = WL.Server.getClientId();
            var response = readUserIdentityFromDB(dbName, clientId);
            if(!response){
                return;
            } else {
                doc = JSON.parse(response.text);
            }
        }
        var id = doc._id; // The id of the doc to remove
        var rev = doc._rev; // The rev of the doc to remove
        var input = {
                    method : 'delete',
                    returnedContentType : 'plain',
                    path : db + "/" + id + "?rev=" + rev,
                    headers: {
                        "Authorization":auth
                        }
                };
        return WL.Server.invokeHttp(input);
    }
```

> To learn more about IBM Cloudant REST API, see the Cloudant documentation.

Creating the client-side authentication components

1. In `index.html`, use the `DoubleStepAuthRealm` instead of the existing realm:

```
<div id="AppDiv">
      ...
      <input type="button" class="appButton" value="Logout"
 onclick="WL.Client.logout('DoubleStepAuthRealm', {onSuccess:WL.Client.reloadApp})" />
      <div id="ResponseDiv"></div>
</div>
```

2. Add a second authentication screen:

```
<div id="AuthStep2Div">
    <h3>Authentication Step 2</h3>
    <p id="AuthQuestion"></p>
    <input type="text" placeholder="Enter answer" id="AuthAnswer"/><br />
    <input type="button" class="formButton" value="Submit" id="AuthStep2Submit" /><input
 type="button" class="AuthCancelButton" value="Cancel" />
</div>
```

3. Finally, update the challenge handler accordingly.
   In this example, a new challenge handler (a new `.js` file), called `DoubleStepAuthRealmChallengeProcessor.js`, is created for this purpose.

   - The response is checked as in the original sample application:

```
var doubleStepAuthRealmChallengeHandler =
WL.Client.createChallengeHandler("DoubleStepAuthRealm");

doubleStepAuthRealmChallengeHandler.isCustomResponse = function(response) {
    if (!response || !response.responseJSON || response.responseText === null) {
        return false;
    }
```

```
    if (typeof(response.responseJSON.authRequired) !== 'undefined'){
        return true;
    } else {
        return false;
    }
};
```

- Add another case for the second authentication step:

```
doubleStepAuthRealmChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;

    if (authRequired == true){
        $("#AppDiv").hide();
        $("#AuthDiv").show();
        $("#AuthInfo").empty();

        $("#AuthStep1Div").hide();
        $("#AuthStep2Div").hide();
        switch (response.responseJSON.authStep) {
            case 1:
                $("#AuthStep1Div").show();
                $("#AuthPassword").val('');
                break;
            case 2:
                $("#AuthStep2Div").show();
                $("#AuthAnswer").val('');
                $("#AuthQuestion").html(response.responseJSON.question);
                break;
        }

        if (response.responseJSON.errorMessage)
            $("#AuthInfo").html(response.responseJSON.errorMessage);

    } else if (authRequired == false){
        $("#AppDiv").show();
        $("#AuthDiv").hide();
        doubleStepAuthRealmChallengeHandler.submitSuccess();
    }
};
```

- Perform the second authentication step:

```
$("#AuthStep1Submit").bind('click', function () {
    var username = $("#AuthUsername").val();
    var password = $("#AuthPassword").val();

    var invocationData = {
        adapter : "AuthAdapter",
        procedure : "submitAuthenticationStep1",
        parameters : [ username, password ]
    };

    doubleStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {});
});

$("#AuthStep2Submit").bind('click', function () {
    var answer = $("#AuthAnswer").val();

    var invocationData = {
        adapter : "AuthAdapter",
        procedure : "submitAuthenticationStep2",
        parameters : [ answer ]
    };

    doubleStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {});
});

$(".AuthCancelButton").bind('click', function () {
    $("#AppDiv").show();
    $("#AuthDiv").hide();
    doubleStepAuthRealmChallengeHandler.submitFailure();
});
```

> To review the remaining/existing sample client-side implementation, see the Adapter-based authentication in hybrid

> applications tutorial.

Sample application

Click to download the sample application.