

# Two-Step adapter authentication

## Overview

This tutorial demonstrates how to implement "Two-Step" adapter-based authentication.

*Two-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 ([../authentication-security/adapter-based-authentication/](#)). The changes extend the application from *single-step* to *Two-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
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Sample application

## 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 (<https://cloudant.com/sign-up/>) and create a database.

Then proceed as follows:

- Change the database permissions - Follow the instructions in the Changing Database Permissions (<https://cloudant.com/changing-database-permissions-tutorial/>) 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

(<https://docs.cloudant.com/authentication.html#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:

```
1 <realm loginModule="AuthLoginModule" name="TwoStepAuthRealm">
2   <className>com.worklight.integration.auth.AdapterAuthenticator</className>
3   <parameter name="login-function" value="AuthAdapter.onAuthRequired"/>
4   <parameter name="logout-function" value="AuthAdapter.onLogout"/>
5 </realm>
```

### Security tests

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

```
1 <customSecurityTest name="TwoStepAuthAdapter-securityTest">
2   <test isInternalUserID="true" realm="TwoStepAuthRealm"/>
3 </customSecurityTest><br />
```

To review the remaining/existing sample components, see the Adapter-based authentication ([../authentication-security/adapter-based-authentication/](https://docs.cloudant.com/authentication-security/adapter-based-authentication/)) tutorial.

## Creating the server-side authentication components

To put in place the Two-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:

```
1 <domain>${USERNAME}.cloudant.com</domain>
```

2. Add the following procedure:

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

3. Protect the `getSecretData` method with the new `TwoStepAuthAdapter`-`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:

```
1 | var auth = "Basic REPLACE_ME_WITH_THE_BASE-64_ENCODED_STRING";
```

2. Create a variable to save your database name:

```
1 | var dbName = "REPLACE_ME_WITH_THE_DATABASE_NAME";
```

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

```
1 | function onAuthRequired(headers, errorMessage){  
2 |   errorMessage = errorMessage ? errorMessage : null;  
3 |   return {  
4 |     authRequired: true,  
5 |     authStep: 1,  
6 |     errorMessage: errorMessage  
7 |   };  
8 | }
```

4. Update the `submitAuthenticationStep1` function:

- Add the following line to get the client ID:

```
1 | function submitAuthenticationStep1(username, password){  
2 |   if (username === "user" && password === "password"){  
3 |     WL.Logger.debug("Step 1 :: SUCCESS");  
4 |     var clientId = WL.Server.getClientId();  
5 |     var userIdentity = {  
6 |       userId: username,  
7 |       displayName: username,  
8 |       attributes: {}  
9 |     };  
10 |   }
```

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

```

1 | //Validate that the DB doesn't already contains the ClientId
2 | var response = deleteUserIdentityFromDB(dbName, null);
3 | //Write ClientId to DB
4 | var response = writeUserIdentityToDB(dbName, {_id:clientId, "userIdentity":userIdentity});

```

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

```

1 | if (response){
2 |   return {
3 |     authRequired: true,
4 |     authStep: 2,
5 |     question: "What is your pet's name?",
6 |     errorMessage : ""
7 |   };
8 | } else {
9 |   return onAuthRequired(null, "Database ERROR");
10 | }
11 | } else{
12 |   WL.Logger.debug("Step 1 :: FAILURE");
13 |   return onAuthRequired(null, "Invalid login credentials");
14 | }
15 | }

```

5. Add submitAuthenticationStep2 function to handle the second authentication step:

- Get the client ID and read it from the database:

```

1 | function submitAuthenticationStep2(answer){
2 |   var clientId = WL.Server.getClientId();
3 |   var response = readUserIdentityFromDB(dbName, clientId);

```

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

```

1  if (response){
2  if (answer === "Lassie"){
3    var doc = JSON.parse(response.text);
4    var userIdentity = doc.userIdentity;
5    WL.Logger.debug("Step 2 :: SUCCESS");
6    WL.Server.setActiveUser("TwoStepAuthRealm", userIdentity);
7    WL.Logger.debug("Authorized access granted");
8    var response = deleteUserIdentityFromDB(dbName, doc);
9    return {
10     authRequired: false
11    };
12  } else{
13    WL.Logger.debug("Step 2 :: FAILURE");
14    return onAuthRequired(null, "Wrong security question answer");
15  }
16  } else {
17    WL.Logger.debug("Step 1 :: FAILURE");
18    return onAuthRequired(null, "Database ERROR");
19  }
20 }

```

## Database actions

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

- Write to the database:

```

1  function writeUserIdentityToDB(db, document){
2    var input = {
3      method : 'post',
4      returnedContentType : 'plain',
5      path : db,
6      headers: {
7        "Authorization":auth
8      },
9      body:{
10       contentType:'application/json; charset=UTF-8',
11       content:JSON.stringify(document)
12     }
13   };
14
15   var response = WL.Server.invokeHttp(input);
16   var responseString = "" + response.statusCode;
17
18   //Checking if the invocation was successful - status code = 2xx
19   if (responseString.indexOf('2') === 0){
20     return response;
21   }
22   return null;
23 }

```

- Read from database:

```

1  function readUserIdentityFromDB(db, key){
2      var input = {
3          method : 'get',
4          returnedContentType : 'plain',
5          path : db + "/" + key,
6          headers: {
7              "Authorization":auth
8          }
9      };
10
11     var response = WL.Server.invokeHttp(input);
12     var responseString = "" + response.statusCode;</p>
13
14     //Checking if the invocation was successful - status code = 2xx
15     if (responseString.indexOf('2') === 0){
16         return response;
17     }
18     return null;
19 }

```

- Delete from the database:

```

1  function deleteUserIdentityFromDB(db, document){
2      var doc = document;
3
4      if (!doc){
5          var clientId = WL.Server.getClientId();
6          var response = readUserIdentityFromDB(dbName, clientId);
7
8          if (!response){
9              return;
10         } else {
11             doc = JSON.parse(response.text);
12         }
13     }
14
15     var id = doc._id; // The id of the doc to remove
16     var rev = doc._rev; // The rev of the doc to remove
17     var input = {
18         method : 'delete',
19         returnedContentType : 'plain',
20         path : db + "/" + id + "?rev=" + rev,
21         headers: {
22             "Authorization":auth
23         }
24     };
25     return WL.Server.invokeHttp(input);
26 }

```

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

# Creating the client-side authentication components

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

```
1 <div id="AppDiv">
2   ...
3   <input type="button" class="appButton" value="Logout" onclick="WL.Client.logout('TwoStepAuthI
4   <div id="ResponseDiv"></div>
5 </div>
```

2. Add a second authentication screen:

```
1 <div id="AuthStep2Div">
2   <h3>Authentication Step 2</h3>
3   <p id="AuthQuestion"></p>
4   <input type="text" placeholder="Enter answer" id="AuthAnswer"/><br />
5   <input type="button" class="formButton" value="Submit" id="AuthStep2Submit" /><input type="bu
6 </div>
```

3. Finally, update the challenge handler accordingly.

In this example, a new challenge handler (a new `.js` file), called

`TwoStepAuthRealmChallengeProcessor.js`, is created for this purpose.

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

```
1 var TwoStepAuthRealmChallengeHandler = WL.Client.createChallengeHandler("TwoStepAuth
2
3 TwoStepAuthRealmChallengeHandler.isCustomResponse = function(response) {
4   if (!response || !response.responseJSON || response.responseText === null) {
5     return false;
6   }
7
8   if (typeof(response.responseJSON.authRequired) !== 'undefined'){
9     return true;
10  } else {
11    return false;
12  }
13  };
```

- Add another case for the second authentication step:

```

1 TwoStepAuthRealmChallengeHandler.handleChallenge = function(response){
2   var authRequired = response.responseJSON.authRequired;</p>
3
4   if (authRequired == true){
5     $("#AppDiv").hide();
6     $("#AuthDiv").show();
7     $("#AuthInfo").empty();
8     $("#AuthStep1Div").hide();
9     $("#AuthStep2Div").hide();
10
11    switch (response.responseJSON.authStep) {
12      case 1:
13        $("#AuthStep1Div").show();
14        $("#AuthPassword").val("");
15        break;
16      case 2:
17        $("#AuthStep2Div").show();
18        $("#AuthAnswer").val("");
19        $("#AuthQuestion").html(response.responseJSON.question);
20        break;
21    }
22
23    if (response.responseJSON.errorMessage)
24      $("#AuthInfo").html(response.responseJSON.errorMessage);
25  } else if (authRequired == false){
26    $("#AppDiv").show();
27    $("#AuthDiv").hide();
28
29    TwoStepAuthRealmChallengeHandler.submitSuccess();
30  }
31 };

```

- Perform the second authentication step:



```

1  $("#AuthStep1Submit").bind('click', function () {
2      var username = $("#AuthUsername").val();
3      var password = $("#AuthPassword").val();
4      var invocationData = {
5          adapter : "AuthAdapter",
6          procedure : "submitAuthenticationStep1",
7          parameters : [ username, password ]
8      };
9
10     TwoStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {});
11 });
12
13 $("#AuthStep2Submit").bind('click', function () {
14     var answer = $("#AuthAnswer").val();
15     var invocationData = {
16         adapter : "AuthAdapter",
17         procedure : "submitAuthenticationStep2",
18         parameters : [ answer ]
19     };
20
21     TwoStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {});
22 });
23
24 $(".AuthCancelButton").bind('click', function () {
25     $("#AppDiv").show();
26     $("#AuthDiv").hide();
27
28     TwoStepAuthRealmChallengeHandler.submitFailure();
29 });

```

To review the remaining/existing sample client-side implementation, see the [Adapter-based authentication in hybrid applications \(../authentication-security/adapter-based-authentication/adapter-based-authentication-hybrid-applications/\)](#) tutorial.

## Sample application

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/TwoStepAuth>) the sample application.