Using LDAP Login Module to authenticate users with LDAP server in hybrid applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/authentication-security/using-ldap-login-module-to-authenticate-users-with-ldap-server-in-hybrid-applications.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

You can use the `LdapLoginModule` class to authenticate users with LDAP servers such as OpenLDAP or Active Directory.
The `LdapLoginModule` class implements the `UserNamePasswordLoginModule` interface. Therefore, you must use it in conjunction with an authenticator that implements the `UsernamePasswordAuthenticator` interface. For example: `FormBasedAuthenticator`.
For more information about how to implement the `UsernamePasswordAuthenticator` interface, see Custom Authenticator and Login Module (../../authentication-security/custom-authenticator-login-module/).

In this tutorial, you learn how to configure and use the `LdapLoginModule` class to protect various IBM MobileFirst Platform Foundation entities.

- Configuring the authenticationConfig.xml file
- Protecting a JavaScript adapter
- Protecting a Java adapter
- Creating the client-side authentication components
- Sample application

Configuring the authenticationConfig.xml file

Realm

1. Add an authentication realm to the `realms` section of the `authenticationConfig.xml` file and call it `LDAPRealm`.
2. Use `FormBasedAuthenticator` in the `className` element because it implements the required `UsernamePasswordAuthenticator` interface.
   This realm uses `LDAPLoginModule` as a login module, which you define in the Login module section.

```
<realm loginModule="LDAPLoginModule" name="LDAPRealm">
    <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className
>
    <onLoginUrl>/console</onLoginUrl>
</realm>
```

Login module

1. Add a login module to the `loginModules` section and call it `LDAPLoginModule`.

```
<loginModule name="LDAPLoginModule">
   <className>com.worklight.core.auth.ext.LdapLoginModule</className>
   <parameter name="ldapProviderUrl" value="ldap://10.0.1.2" />
   <parameter name="ldapTimeoutMs" value="2000" />
   <parameter name="ldapSecurityAuthentication" value="simple" />
   <parameter name="validationType" value="searchPattern" />
   <parameter name="ldapSecurityPrincipalPattern" value="{username}@myserver.com" />
   <parameter name="ldapSearchFilterPattern" value="(&amp;amp;(objectClass=user)(sAMAccountNa
me={username})(memberof=CN=Sales,OU=Groups,OU=MyCompany,DC=myserver,DC=com))" />
   <parameter name="ldapSearchBase" value="dc=myserver,dc=com" />
   <!-- Optional LDAP parameter "REFERRAL", available values are: ignore, follow, throw -->
   <parameter name="ldapReferral" value="ignore" />
</loginModule>
```

2. Use `com.worklight.core.auth.ext.LdapLoginModule` in the `className` element.
   - The `ldapProviderUrl` parameter is mandatory. It defines the URL of your LDAP server.
   - The `ldapTimeoutMs` parameter is mandatory. It defines the timeout for LDAP server requests (in milliseconds).
   - The `ldapSecurityAuthentication` parameter is mandatory. It defines the type of authentication that is required by LDAP server. The usual value is `simple`, but you might need to contact LDAP administrator for a more appropriate value.
   - The `validationType` parameter is mandatory. It defines the type of validation that is performed. `LdapLoginModule` supports three types of validation:
     - **`exists`**: The login module tries to establish the LDAP binding by using the supplied credentials. The validation of the credentials is considered successful when binding is successfully established.
     - **`searchPattern`**: The login module first tries to run the `exists` validation. After such validation is successful, the login module issues a search query to the LDAP server context according to the `ldapSearchFilterPattern` and `ldapSearchBase` parameters. Credential validation is considered successful if a search query returns one or more entries.
     - **`custom`**: Use this value to enable a custom validation logic. The login module tries to run the `exists` validation. After such validation is successful, the login module calls the `public boolean doCustomValidation(LdapContext ldapCtx, String username)` method. You can override this method by creating a custom Java class in your project and extending the `com.worklight.core.auth.ext.UserNamePasswordLoginModule` class.
       For more information about custom LDAP validation types, see the user documentation.
   - The `ldapSecurityPrincipalPattern` parameter is mandatory. It defines the pattern in which LDAP security principal is sent to the LDAP server. You can use a *`{username}`* placeholder to inject the user name from the authenticator.
   - The `ldapSearchFilterPattern` and `ldapSearchBase` parameters are optional. They apply only to the `searchPattern` validation type.

Security test

Add a security test to the `securityTests` section of the `authenticationConfig.xml` file.
You use this security test to protect the adapter procedure. Therefore, use the `customSecurityTest` element.
Remember the security test name because you will reuse it to protect adapters.

```
<customSecurityTest name="LDAPSecurityTest">
   <test realm="wl_directUpdateRealm" step="1" />
   <test isInternalUserID="true" realm="LDAPRealm" /
>
</customSecurityTest>
```

Protecting a JavaScript adapter

1. Create an adapter and name it **DummyAdapter**.
2. Add a `getSecretData` procedure and protect it with the security test that you created previously.

   ```
   <procedure name="getSecretData" securityTest="LDAPSecurityTest"/>
   ```

   In this module, the `getSecretData` procedure returns some hardcoded value:

   ```
   function getSecretData(){
      return {
         secretData: '123456'
      };
   }
   ```

Protecting a Java adapter

1. Create a Java adapter.
2. Add a `getSecretData` method and protect it with the realm that you created previously.
   In this module, the `getSecretData` procedure returns some hardcoded value:

   ```
   @GET
   @Produces("application/json")
   @OAuthSecurity(scope="LDAPRealm")
   public JSONObject getSecretData(){
    JSONObject result = new JSONObject()
  ;
    result.put("secretData", "123456");
    return result;
   }
   ```

3. To set the new realm as the default user identity for the application, in the application descriptor, add this option:

   ```
   <br />
  <userIdentityRealms>LDAPRealm</userIdentityRealms><br />
   ```

Creating the client-side authentication components

The application consists of two main `div` elements:

- The `AppDiv` element displays the application content.
- The `AuthDiv` element is used for authentication forms.

When authentication is required, the application hides the `AppDiv` element and shows the `AuthDiv` element. When authentication is complete, it does the opposite.

```
 <br />
<div id="AppDiv">
   <input type="button" class="appButton" value="Call protected adapter proc" onclick="getSecretData()" />
   <input type="button" class="appButton" value="Logout" onclick="WL.Client.logout('LDAPRealm',{onSucces
s: WL.Client.reloadApp})" />
   <p id="resultDiv"></p>
</div>
```

The buttons are used to call the `getSecretData` procedure and to log out.

```
<div id="AuthDiv" style="display:none">
   <div id="loginForm">
      <input type="text" id="usernameInputField" placeholder="Enter username" />
      <input type="password" placeholder="Enter password" id="passwordInputField" /
>
      <br/>
      <input type="button" class="formButton" id="loginButton" value="Login" />
      <input type="button" class="formButton" id="cancelButton" value="Cancel" />
   </div>
</div>
```

The `AuthDiv` element is styled as `display:none` because it must not be displayed before authentication is requested by server.

Challenge handler

Use the `WL.Client.createChallengeHandler` method to create a challenge handler object. Supply a realm name as a parameter.

```
var LDAPRealmChallengeHandler = WL.Client.createChallengeHandler("LDAPRealm");
```

isCustomResponse

The `isCustomResponse` function of the challenge handler is called each time a response is received from the server. That function is used to detect whether the response contains data that are related to this challenge handler. It must return `true` or `false`.
The default login form that is returned from the MobileFirst server contains the `j_security_check` string. If the challenge handler detects it in the response, it returns `true`.

```
LDAPRealmChallengeHandler.isCustomResponse = function(response) {
    if (!response || !response.responseText) {
        return false;
    }
    var idx = response.responseText.indexOf("j_security_check");
    if (idx >= 0){
        return true;
    }
    return false;
};
```

handleChallenge

If the `isCustomResponse` method returns `true`, the framework calls the `handleChallenge` function. This function is used to perform required actions, such as hide the application screen or show the login screen. After the client application detects that the server sent a login form, which means that the server is requesting authentication, the application hides the `AppDiv`, shows the `AuthDiv`, and cleans up the `passwordInputField` element.

```
LDAPRealmChallengeHandler.handleChallenge = function(response){
        $('#AppDiv').hide();
        $('#AuthDiv').show();
        $('#passwordInputField').val('');
};
```

Other methods

In addition to the methods that the developer must implement, the challenge handler contains functionality that the developer might want to use:

- The `submitLoginForm` function sends collected credentials to a specific URL. The developer can also specify request parameters, headers, and callbacks.
- The `submitSuccess` function notifies the framework that the authentication process completed successfully. The framework then automatically issues the original request that triggered authentication.
- The `submitFailure` function notifies the framework that the authentication process completed with failure. The framework then disposes of the original request that triggered authentication.

Important: You **must** attach each function to its object. For example: `myChallengeHandler.submitSuccess()`

Login button

A click on a **Login** button triggers a function that collects the user name and password from the HTML input fields and submits them to the server.
It is possible to set request headers here, and to specify callbacks.
The form-based authenticator uses the hardcoded `j_security_check` URL component. You cannot have more than one instance of it.

```
$('#loginButton').bind('click', function () {
    var reqURL = '/j_security_check';
    var options = {};
    options.parameters = {
          j_username : $('#usernameInputField').val(),
          j_password : $('#passwordInputField').val()
    };
    options.headers = {};
    LDAPRealmChallengeHandler.submitLoginForm(reqURL, options, LDAPRealmChallengeHandler.submitLo
ginFormCallback);
});
```

Cancel button

A click on a **Cancel** button hides `AuthDiv`, shows `AppDiv`, and notifies the framework that authentication failed.

```
$('#cancelButton').bind('click', function () {
    $('#AppDiv').show();
    $('#AuthDiv').hide();>
    LDAPRealmChallengeHandler.submitFailure()
;
});
```

submitLoginFormCallback

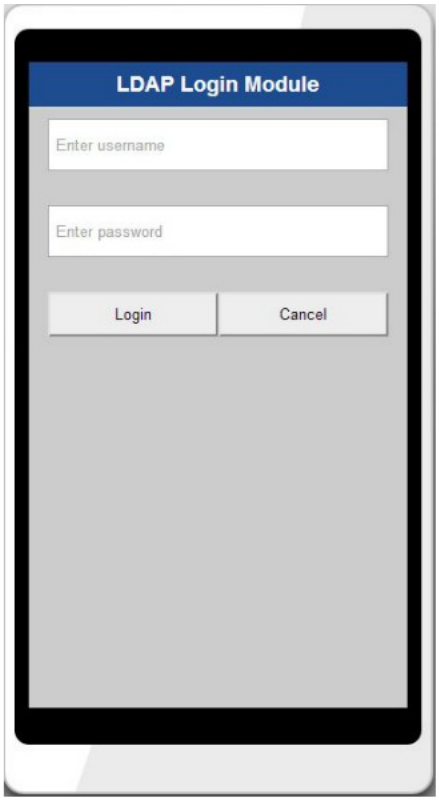
The callback function checks the response for the containing server challenge again. If a challenge is found, the `handleChallenge` function is called again.
No challenge in the server response means that authentication completed successfully. In this case, `AppDiv` is shown, `AuthDiv` is hidden, and the framework is notified about authentication success.

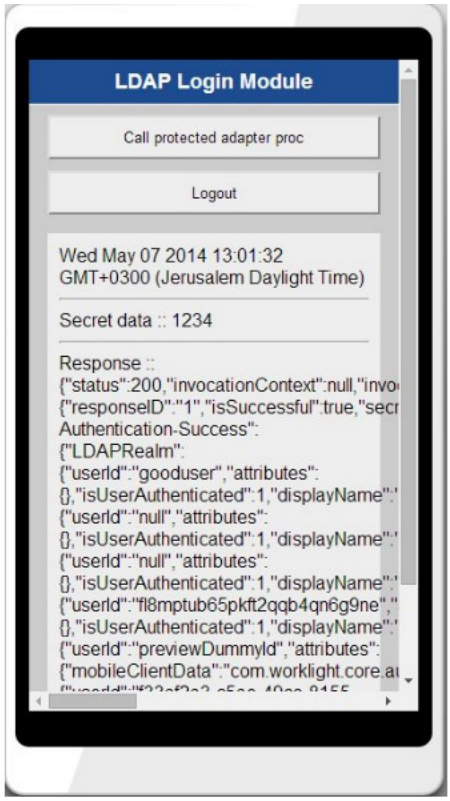
```
LDAPRealmChallengeHandler.submitLoginFormCallback = function(response) {
    var isLoginFormResponse = LDAPRealmChallengeHandler.isCustomResponse(response)
;
    if (isLoginFormResponse){
        LDAPRealmChallengeHandler.handleChallenge(response);
    } else {
        $('#AppDiv').show();
        $('#AuthDiv').hide();
        LDAPRealmChallengeHandler.submitSuccess();
    }
};
```

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/LDAPLoginModuleHybridProject.zip) the Studio project.

LDAP Login Module
Call protected adapter proc
Logout


LDAP Login Module
Enter username
Enter password
Login
Cancel


LDAP Login Module
Call protected adapter proc
Logout
Wed May 07 2014 13:01:32
GMT+0300 (Jerusalem Daylight Time)
Secret data :: 1234
Response ::
{"status":200,"invocationContext":null,"invo
{"responseID":"1","isSuccessful":true,"secr
Authentication-Success":
{"LDAPRealm":
{"userId":"gooduser","attributes":
{},"isUserAuthenticated":1,"displayName":"
{"userId":"null","attributes":
{},"isUserAuthenticated":1,"displayName":"
{"userId":"null","attributes":
{},"isUserAuthenticated":1,"displayName":"
{"userId":"fl8mptub65pkft2qqb4qn6g9ne","
{},"isUserAuthenticated":1,"displayName":"
{"userId":"previewDummyId","attributes":
{"mobileClientData":"com.worklight.core.au