Using the MobileFirst Server to authenticate external resources

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/authentication-and-security/using-mobilefirst-server-authenticate-external-resources.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Relevant to:

- Hybrid
- Download MobileFirst project (https://github.com/MobileFirst-Platform-Developer-Center/OAuthExternalServer)

Overview

By using MobileFirst Server, you can protect external resources by applying the OAuth 2.0 protocol. The OAuth 2.0 protocol is based on acquiring an access token, which encapsulates the authorization that is granted to the client.

- MobileFirst Server acts as an Authorization Server and issues an access token that can be validated by the external service.
- Client applications request the access token from MobileFirst Server and sends the token to the external services.
- The scope of the access token is a realm (or several realms), which are defined in the MobileFirst project.

This tutorial covers the following topics:

- Authenticating through an access token over OAuth 2.0
- Accessing external resource flows in different ways
- Configuring the MobileFirst project
- Configuring MobileFirst Server to use a keystore
- Using the WLResourceRequest API
- Reporting analytics
- Sample application

Authentication through an access token over OAuth 2.0

The OAuth 2.0 authorization framework enables an application from independent software vendors (also called third-party application) to obtain limited access to an HTTP service.

The implementation uses three roles of the OAuth protocol:

- **Resource Server**: third-party server
  The server that hosts the protected resources. It can accept, and respond to, protected resource requests by using access tokens.
- **Client**: app
  An application that requests protected resources.
- **Authorization Server**: MobileFirst Server
  The server that issues access tokens to the client after it has successfully authenticated the resource

owner and obtained authorization.

Overview of the Resource Server component

The **Resource Server** is an external server that hosts the available resource.

One use case is that services are deployed on a cloud, such as **Bluemix**. But this flow is not restricted to that case and works with any third-party server.

The public key that is necessary to verify the token must be configured for this component.

Java and `node.js` libraries are provided for offline validation.

Overview of the client component

As of IBM MobileFirst Platform Foundation 7.1, the **MobileFirst SDK** can now also be used to access the Resource Server through `WLResourceRequest`. This API handles the OAuth-based security model protocol and invokes the required challenges.

An **External SDK** can also be used to access the Resource Server. The external SDK must be able to attach a header to the request.

Overview of the MobileFirst Server component

MobileFirst Server uses the authentication infrastructure to issue an access token for the requested scope (MobileFirst realm).

Configuring the MobileFirst project

Configuring the MobileFirst project consists in configuring the scope for the access token. The scope of an access token must be a predefined realm in a MobileFirst project. You configure realms in the `*MobileFirst_project*/server/conf/authenticationConfig.xml` file.

The expiration period of a scope token (realm name) is defined by the expiration attribute of the login module that is associated with the realm.

If you want to configure a login module called `StrongDummy` with an expiration attribute, edit the `authenticationConfig.xml` file in the login module section as follows::

[code lang="xml"]
<loginModule name="StrongDummy" expirationInSeconds="1800">
<className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>
[/code]

If you want to configure a realm called `SampleAppRealm` that is associated with `StrongDummy`, edit the `authenticationConfig.xml` file in the realm section as follows:

[code lang="xml"]
<realm name="SampleAppRealm" loginModule="StrongDummy">
<className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
</realm>
[/code]

Configuring MobileFirst Server to use a keystore

To use this feature, preferably use or create your own keystore and configure MobileFirst Server to use it.

For an example in an unrelated context, see the topic about configuring device auto provisioning, in the product documentation.

Important: Using the default MobileFirst Server keystore is **NOT** secure!

Configuring the external service

For your external service to accept the access token, you must add a validation library to your service, which must be able to validate the token with either online or offline validation.

You can use one of two libraries for this purpose:

- `com.ibm.imf.oauth.common_1.0.0.jar` - Java lib
- The `node.js` library, which can be downloaded through `npm`.

For MobileFirst Server installation - You can find the necessary java library and files in: `*installation_dir*/MobileFirstServer/external-server-libraries`

For MobileFirst Studio - When you create a new project, you can find the java library and files in: `*project_dir*/externalServerLibraries`

External service configuration - Using Java

The purpose of this module is to enable offline validation of access tokens that are generated by MobileFirst Server for Java web applications.

Validation of access tokens that are generated by MobileFirst Server is also possible for `node.js` servers.

To use the Java library, you need three files:

- `Certificate` - For the associated sample, the certificate that has been exported from the MobileFirst Server Keystore. You can use the Java keytool. In production, preferably use your own keystore as explained in Configuring MobileFirst Server to use a keystore.
- `com.ibm.imf.oauth.common_1.0.0.jar`
- `OAuthTai-1.0.mf`
- `server.env` - It may also be useful to create this file in the same directory as the `server.xml` in the external server. In this file you will add the URL to the authorization server so this way it will be included in the environment variables of your deployment.

External service configuration - Using Java: servlet filter

1. Add the `com.ibm.imf.oauth.common_1.0.0.jar` file to your server under this path: `usr/extension/lib`
2. Add the `OAuthTai-1.0.mf` file to the path `usr/extension/lib/features`
3. Add following entry in environment variables of your deployment. One of the ways is to add it into server.env, which locates in the same directory of your server.xml.
   `publicKeyServerUrl=URL_to_authorization_server`
   where `URL_to_authorization_server` is:
   `http://<ip-address-to-MobileFirst-server>:<port-number>/<project-context-root>`
4. Add a security constraint and a security role to the `web.xml` file of your external server, as shown below:
   [code lang="xml"]
   <security-constraint>
   <web-resource-collection>

<web-resource-name>RESTServlet</web-resource-name>
<url-pattern>/api/protected</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>TAIUserRole</role-name>
</auth-constraint>
</security-constraint>
<security-role id="SecurityRole_TAIUserRole" >
<description>This is the role that MFP OAuthTAI uses to protect the resource, and it is required to be mapped to 'All Authenticated in Application' in WAS and 'ALL_AUTHENTICATED_USERS' in Liberty</description>
<role-name>TAIUserRole</role-name>
</security-role>
[/code]

5. You must modify the external server `server.xml` file, too. Configure the feature manager to include these features:
   [code lang="xml"]
   <featureManager>
   <feature>jsp-2.2</feature>
   <feature>appSecurity-2.0</feature>
   <feature>usr:OAuthTai-1.0</feature>
   <feature>servlet-3.0</feature>
   <feature>jndi-1.0</feature>
   </featureManager>
   [/code]
6. In your application, add a security role:
   [code lang="xml"]
   <application ...>
   ...
   <application-bnd>
   <security-role name="TAIUserRole">
   <special-subject type="ALL_AUTHENTICATED_USERS"/>
   </security-role>
   </application-bnd>
   </application>
   [/code]
7. Configure OAuthTAI, where you set which URLs are to be protected:
   [code lang="xml"]
   <usr_OAuthTAI id="myOAuthTAI">
   <securityconstraint httpMethods="GET POST" scope="SampleAppRealm" securedURLs="/REST-Server/api/protected"></securityconstraint>
   </usr_OAuthTAI>
   [/code]
   The scopes that you provide are all defined in the `authenticationConfig.xml` file of the MobileFirst

Project. You can provide multiple scopes.

Using the WLResourceRequest API

The MobileFirst `WLResourceRequest` API provides built-in support fur using MobileFirst access tokens for the following platforms:

- Hybrid - JavaScript™
- Android
- iOS
- Windows Phone 8.1

Use of WLResourceRequest: JavaScript example

This JavaScript example shows how to use the client-side API for access to an external service.

[code lang="javascript"]
function callProtectedRestAPI() {
var url = EXTERNAL_SERVER_URL + REST_CONTEXT_ROOT+PATH_PROTECTED;
var request = new WLResourceRequest(url, WLResourceRequest.GET);
request.send().then(
function(response){
showResult(response.responseJSON.description);
},
function(error){
showErrorResult("failed to get scope!");
}
);
}
[/code]

Reporting analytics

Java

The **TAI** library is capable of reporting analytic events to IBM MobileFirst Platform Operational Analytics. To do this, you must configure the Resource Server that contains the **TAI** with your Analytics URL and credentials.

The MobileFirst Operational Analytics server is protected by basic authentication. When you installed this server, you configured the data entry point and basic authentication credentials. To configure the **Resource Server**, you must provide the Analytics credentials, specifically the URL to the data entry point, the user name, and password. These properties are set with the following property names:
- `imf.analytics.url`
- `imf.analytics.username`
- `imf.analytics.password`

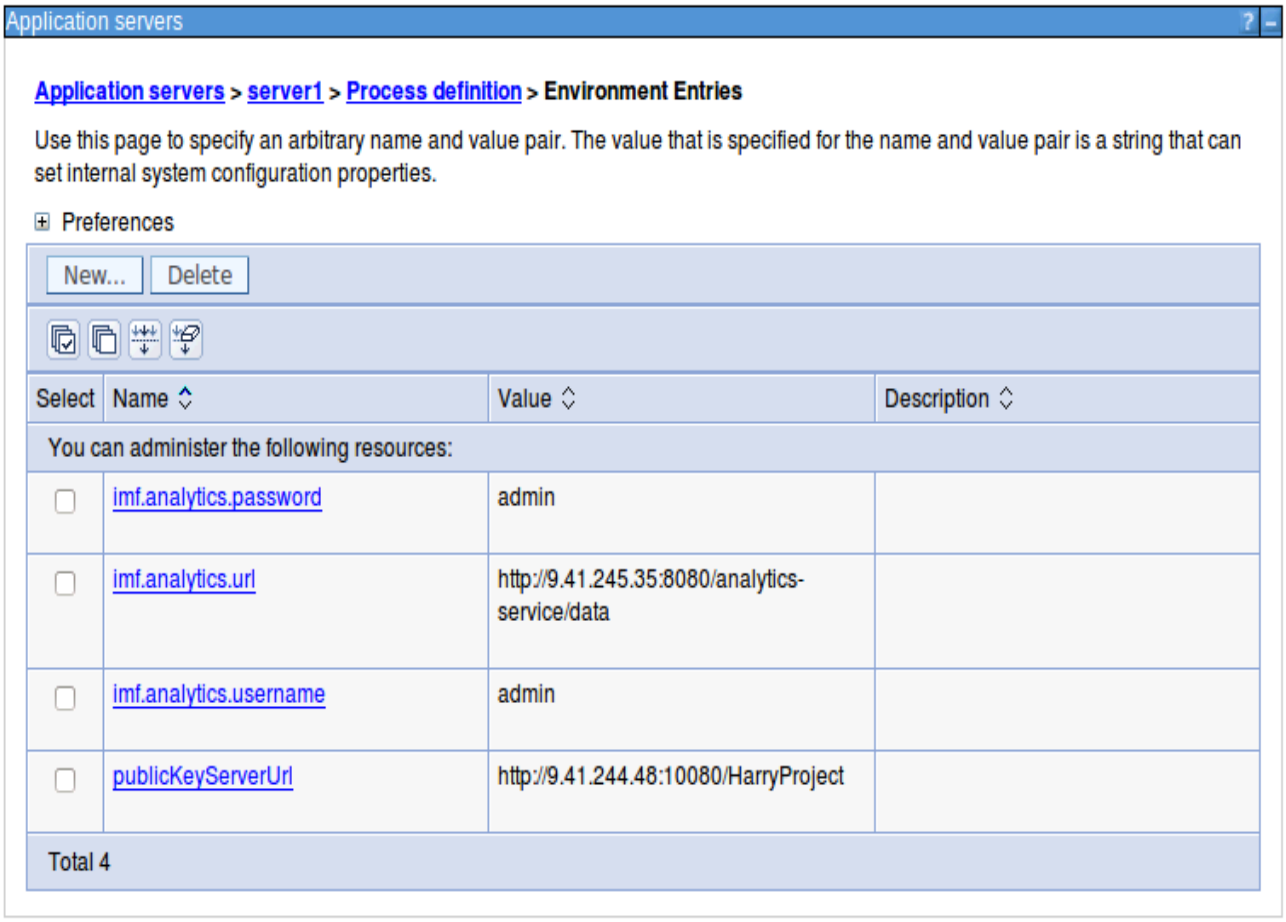
- If your Resource Server is running on Liberty, you must configure these properties by using JNDI. You can do this by adding entries to your `server.xml` file. For example:

  [code]

  <jndiEntry jndiName="imf.analytics.username" value="admin"/>

  [/code]

- If your Resource Server is running on WebSphere Application Server, configure these properties as environment entries.
  To do this, navigate from the WebSphere Application Server administration console to **Servers > Server Type > WebSphere application servers** > *{your_server}* > **Server Infrastructure** > **Java and Process Management** > **Process Definition** > **Additional Properties** > **Environment Entries** > **New**.
  This is where you had to set `publicKeyServerUrl` when you set up OAuthTAI.



  (https://developer.ibm.com/mobilefirstplatform/wp-content/uploads/sites/32/2014/12/WasEnvironmentEntries.png)

After these properties are set, the **TAI** will post its events to MobileFirst Operational Analytics.

Node.js

You can protect your resources that are running on `Node.js` servers with OAuth-based MobileFirst security. Like the Java implementation, the MobileFirst Operational Analytics server is protected by basic authentication. You must configure the **Resource Server** by connecting the `passport-mfp-token-validation` module to the Analytics server.

The `passport-mfp-token-validation` npm module provides passport validation strategy and a verification function to validate access tokens and ID tokens that are issued by the MobileFirst Server.

The following example shows how to use `mfpStrategy` in a node application:
[code lang="javascript"]
var express = require('express'),
passport = require('passport-mfp-token-validation').Passport,

```
mfpStrategy = require('passport-mfp-token-validation').Strategy;
//the configuration ('config') is optional if you wish to report
//events to the Analytics Server.
var config = {
url : 'http://localhost:9080/worklight-analytics-service/data',
username : 'admin',
password : 'admin'
};

passport.use(new mfpStrategy({publicKeyServerUrl:'http://localhost:9080/WLProject', analytics : {onpremise: config}}));

var app = express();
app.use(passport.initialize());

//protect api with MFP strategy using scope Realm1 Realm2 Realm3
app.get('/v1/apps/:appid/service', passport.authenticate('mobilefirst-strategy', {session: false, scope: 'Realm1 Realm2 Realm3' }),
function(req, res){
res.send(200, req.securityContext);
}
);

app.listen(3000);
[/code]

To start this example, issue the following commands:
`

$ npm install express
$ npm install passport
$ npm install passport-mfp-token-validation
`
```

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/OAuthExternalServer) the MobileFirst project.