

Trust Association Interceptor

Overview

MobileFirst Foundation provides a Java library to facilitate the authentication of external resources through IBM WebSphere's Trust Association Interceptors

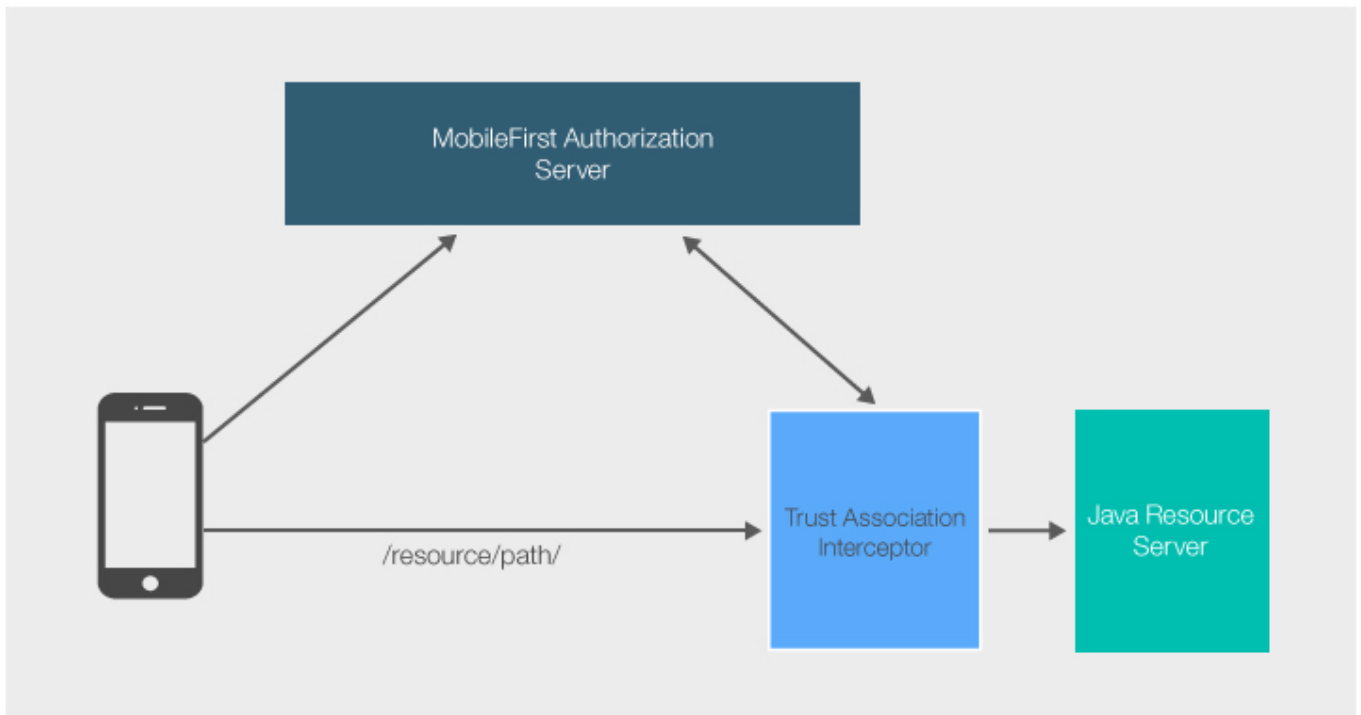
(https://www.ibm.com/support/knowledgecenter/SSHRKX_8.5.0/mp/security/sec_ws_tai.dita).

The Java library is provided as a JAR file (**com.ibm.mfp.oauth.tai-8.0.0.jar**).

This tutorial shows how to protect a simple Java Servlet, `TAI/GetBalance`, by using a scope (`accessRestricted`).

Prerequisite:

- Read the Using the MobileFirst Server to authenticate external resources (../) tutorial.
- Be familiar with the MobileFirst Foundation security framework (../..).



Server setup

1. Download the OAuth Security Extension from the **MobileFirst Operations Console** → **Download Center** → **Tools** tab.
- Unpack the `mfp-oauth-tai.zip` archive.
1. Add the `com.ibm.mfp.oauth.tai.jar` file to the WebSphere Application Server instance inside **usr/extension/lib**.
2. Add the `OAuthTai.mf` file to the WebSphere Application Server instance inside **usr/extension/lib/features**.

web.xml setup

Add a security constraint and a security role to the `web.xml` file of the WebSphere Application Server instance:

```

<security-constraint>
  <web-resource-collection>
    <web-resource-name>TrustAssociationInterceptor</web-resource-name>
    <url-pattern>/TAI/GetBalance</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 MobileFirst OAuthTAI uses to protect the resource, and it is mandatory to map it to 'All Authenticated in Application' in WebSphere Application Server full profile and to 'ALL_AUTHENTICATED_USERS' in WebSphere Application Server Liberty.</description>
  <role-name>TAIUserRole</role-name>
</security-role>

```

server.xml

Modify the WebSphere Application Server `server.xml` file to your external resource.

- Configure the feature manager to include the following features:

```

<featureManager>
  <feature>jsp-2.2</feature>
  <feature>appSecurity-2.0</feature>
  <feature>usr:OAuthTai-8.0</feature>
  <feature>servlet-3.0</feature>
  <feature>jndi-1.0</feature>
</featureManager>

```

- Add a security role as a class annotation in your Java servlet :

```
@ServletSecurity(@HttpConstraint(rolesAllowed = "TAIUserRole"))
```

If you are using servlet-2.x , you need to define the security role in your web.xml file:

```

<application contextRoot="TAI" id="TrustAssociationInterceptor" location="TAI.war" name="TrustAssociationInterceptor"/>
  <application-bnd>
    <security-role name="TAIUserRole">
      <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
  </application-bnd>
</application>

```

- Configure OAuthTAI. This is where URLs are set to be protected:

```

<usr_OAuthTAI id="myOAuthTAI" authorizationURL="http://localhost:9080/mfp/api" clientId="ExternalResourceId" clientSecret="ExternalResourcePass" cacheSize="500">
  <securityConstraint httpMethods="GET POST" scope="accessRestricted" securedURLs="/GetBalance"/></securityConstraint>
</usr_OAuthTAI>

```

- **authorizationURL**: Either your MobileFirst Server (`http(s)://your-hostname:port/runtime-name/api`), or an external AZ Server such as IBM DataPower.

- **clientID**: The Resource server must be a registered confidential client. To learn how to register a confidential client, read the Confidential Clients (../confidential-clients/) tutorial. *The confidential-client* ***MUST*** have the allowed scope `authorization.introspect` so that it can validate tokens.
- **clientSecret**: The Resource server must be a registered confidential client. To learn how to register a confidential client, read the Confidential Clients (../confidential-clients/) tutorial.
- **cacheSize (optional)**: TAI uses the Java-Token-Validator cache to cache tokens and introspection data as values, so that a token that comes in the request from the client won't need to be introspected again in a short time interval.

The default size is 50,000 tokens.

If you want to guarantee that the tokens are introspected on each request, set the cache value to 0.

- **scope**: The resource server authenticates against one or more scopes. A scope can be a security check or a scope element mapped to security checks.

Using the Token Introspection Data From the TAI

From your resource, you may want to access the token information that was intercepted and validated by the TAI. You can find the list of data found on the token in this user documentation topic:

https://www.ibm.com/support/knowledgecenter/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/html/refjava-mfp-java-token-validator/html/com/ibm/mfp/java/token/validator/data/package-summary.html To obtain this data, use the WSSubject API

(http://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/rwlp_sec_apis.html):

```
Map<String, String> credentials = WSSubject.getCallerSubject().getPublicCredentials(Hashtable.class).iterator().next();
JSONObject securityContext = new JSONObject(credentials.get("securityContext"));
...
securityContext.get('mfp-device')
```

Sample

You can deploy the project on supported application servers (WebSphere Application Server full profile and WebSphere Application Server Liberty profile).

Download the simple Java servlet (<https://github.com/MobileFirst-Platform-Developer-Center/TrustAssociationInterceptor/tree/release80>).

Sample usage

1. Make sure to update the confidential client (../confidential-client) and secret values in the MobileFirst Operations Console.
2. Deploy either of the security checks: **UserLogin** (../user-authentication/security-check/) or **PinCodeAttempts** (../credentials-validation/security-check/).
3. Register the matching application.
4. Map the `accessRestricted` scope to the security check.
5. Update the client application to make the `WLResourceRequest` to your servlet URL.
6. Set the scope of your securityConstraint scope to be the security check that your client needs to authenticate against.

