Authentication in OA4MP

Authentication – the means by which users are identified as valid[1] -- is not as simple as one would like in OA4MP. Since it is to be “open authorization for many people”, it does not, in fact, have a native concept of a user, but relies on some added mechanism. This permits institutions to use their existing infrastructure with OA4MP rather than having to give every user yet another set of credentials on an OA4MP server. There are 4 main ways this integration can be done.

1. *Proxy case*. Use a proxy to a trusted OAuth service that has all of your users. Their login is accepted as a valid authentication on your server, effectively allowing you to make your system and an extension of theirs. A very common use case is to get a client on CILogon which gives access to most major academic and educational research institutions. Logins to your site are redirected to the proxy site and if successful there, the user is accepted in OA4MP.
2. *Header case*. Configure OA4MP to use an HTTP header from a web server. The standard is the REMOTE_USER header, but you may use any other header of your choice). Most commonly if Tomcat is hosting OA4MP, then Tomcat users can be accepted as valid OA4MP users. This also works with reverse proxies such as Apache or Nginx. Note that since it is easy to forge headers, you must manage the trust relation with the authenticating service.
3. *Extension case*. Extend OA4MP (in Java) directly to use your existing user management system. In this case there is already a system in place. Note that the default “out of the box” behavior for OA4MP is to reject all requests.
4. *Independent case*. Write your own Authentication Service (AS) which is wholly independent of OA4MP then use callouts to OA4MP to notify it of progress. This lets you deploy your service along with OA4MP. CILogon is a famous example of this. Note that your application is wholly responsible for handling user logins in a specification compliant way.
5. *Dedicated issuer case*. Write your own service that handles all requests up to and including token issuance. OA4MP then may be used for all subsequent OAuth token calls (such as refreshes, exchanges, token introspection etc.). You may handle vetting requests and authentication (if any) in any way you deem fit, giving enormous flexibility in this case. OA4MP just mints (*i.e.* issues) tokens and then manages them. This is of particular interest to institutions that are replacing their X509 certificate infrastructure with tokens.

You should also read the documentation for configuring the authorizationServlet.

Supplied Deployment Descriptors (web.xml)

In a standard OA4MP install, there are a fully function DDs (deployment descriptors, aka the web.xml file) supplied in

`$OA4MP_SERVER/etc/WEB-INF`

1 In OA4MP, authorization – what a user is allowed to do – is set by policies.

A list of these is here:

Name	***Description***
dedicated-web.xml	DD for a running OA4MP as a dedicated issuer.
di-service-web.xml	DD for running the DIService (Case 4)
header-web.xml	DD for using using headers from Tomcat (Case 2)
proxy-web.xml	DD for using a proxy service (Case 1)
sas-web.xml	Partial DD for enabling the SAS service. Integrate into other DDs as needed.
web.xml	Deployed DD at install.

For Case 5, this is a use pattern and can coexist with any other deployment. The supplied DD is a minimal one.

For Case 3, see the project at https://github.com/ncsa/oa4mp-extension-example which is a complete service. That has the required extensions and DD.

Nota Bene: If you update OA4MP, your changes to web.xml in **`$CATALINA_HOME/oauth2/WEB-INF/web.xml`** will be lost, so we suggest you adopt a best practice of keeping the deployment descriptor in the OA4MP directory and copying it as needed. Alternately, you can create a context file in Tomcat that points to your $OA4MP_SERVER/etc/WEB-INF DD, but this is not completely trivial hence left to the more experienced reader.

6.