Java Extensions to OA4MP for Authentication

Overview

The Java code in OA4MP can easily be extended to add users. A use case is that you have an existing user management system you can access via Java and you want all authentication requests to use that. The basic way is to create a Maven project that uses OA4MP and extends OA2AuthenticationServer. There is precisely one method, checkUser, to override.

Setup

The easiest way is to get the OA4MP extension example from GitHub. This has a fully functional extension to OA4MP with a single user, **`me,`** and password **`124567890`**. Do read the readme.md there for more particulars.

and extend the class OA2AuthenticationServer.java and override the **`checkUser`** method there.

```
public void checkUser(String username,
                      String password)
                      throws GeneralSecurityException;
```

This takes the username and password that the user entered on the webpage. The contract is to do whatever you need to and if the user passes muster, do nothing, otherwise throw the **`GeneralSecurityException`** or a **`RuntimeException`** of your choice.

The server configuration

In the standard OA4MP configuration, you *do not* need to configure anything. The system will automatically find the authorization page (very minimal) and do the authentication. That is to say, there is no **`authorizedServlet`** element.

Tomcat configuration

The only configuration needed is in the Tomcat web.xml file where the standard authentication service:

```
<servlet>
    <servlet-name>authorize</servlet-name>
    <servlet-class>org.oa4mp.server.proxy.OA2AuthenticationServer</servlet-class>
</servlet>
```

and device flow service (if you want OA4MP to support that):

```
<servlet>
    <servlet-name>device</servlet-name>
    <servlet-class>org.oa4mp.server.proxy.RFC8628AuthenticationServer</servlet-class>
</servlet>
```

have the servlet-class elements replaced by your implementation.

How does it work?

OA4MP has a default page, authorize-init.jsp, for the authorization code flow that will do the authentication. (It is so named because, as per the OAuth spec., it sits at the **`/authorize`** endpoint for the service, but its function is authenticating users). When a user comes to the endpoint, the page is populated then the values read. You may just (optionally) replace this too if you like and this section details how.

Values set in the form

Parameter	Display?	Description
clientName	Y	The name of the client
client Home	Y	The URL that is registered as the home
clientScopes	Y	Blank delimited string of scopes the user requested
retryMessage	Y	If the user tries to submit the form and it fails, the form is redisplayed with the message. It is empty on the first attempt.
authorizationGrant	N	This is the unique identifier for this transaction. It is needed in internally an must be sent back to the server on submit

Values to be returned

Parameter		Description
AuthUsername		The username used in authentication
AuthPassword		The user's password
authorizationGrant		The transaction identifier sent. If you do not return this, there is no way to track which request this goes to and an error will be raised.

In the **`checkUser`** method overridden above, the **`AuthUsername`** and **`AuthPassword`** are read and passed to it.

So you may replace that page as you like with whatever you like. If you have enabled the device code flow, then there is a similar page device-init.jsp that accepts the same parameters including a **`userCode`** (if the user sends that as a parameter).

*But but but I really want to add some of my own parameters to the **`authorization-init.jsp`** page and process them…*

Then override

```
protected void createRedirect(HttpServletRequest request,
                              HttpServletResponse response,
                              ServiceTransaction trans)
```

along with **`checkUser`**. The parameters are in the request. Be sure to call super on this method when you are done. It is invoked *after* **`checkUser`** and will redirect the user's browser, so nothing can execute after it is called.

For the device code flow, you would override

```
protected void processRequest(HttpServletRequest request,
                              PendingState pendingState,
                              boolean checkCount)  throws Throwable
```

And call super on this first (since it does all of the work of managing the number of user retries as well as tracking timeouts), then add your own code, *viz.*,

```
@Override
protected void processRequest(HttpServletRequest request,
                              PendingState pendingState,
                              boolean checkCount) throws Throwable{
    super.processRequest(request, pendingState, checkCount);
    // now start getting parameters from the request and using them
}
```

The method will either throw and exception if there is an issue (such as the user's code has expired) or return, so you don't have to do anything special and your code won't run unless the flow is working. This is important, since if you do change hwo the flow works, your OA4MP extension might not be OAuth compliant or even insecure.