Community TechNote

Single-sign-on between MWS custom portlets and IS services

Version 2 - Updated 22 Sep 2009

Abstract

This article describes how to use Single-Sign-On in the authentication of MWS portlets against Integration Server, when invoking services inside it as web services. This allows to utilize the user currently logged in in MWS to authenticate in IS, which allows to avoid the issue of configuring an user and password to this goal, and a proper way to configure permissions in a SOA.

This Single-Sign-On is achieved by means of the Central Users and the SAML authentication mechanisms. The article describes these mechanisms, and also provides a tutorial to make a quick demonstration of the feature.

Authenticating portlets against IS services

It is most common that portlets running inside MWS call IS services to perform a large part of its duties. One of the issues to address in this typical scenario is the one of authentication of these calls, i.e. of the credentials that IS must use when executing these calls.

A popular solution to this issue is to have a generic user and password, that all portlets will always use, no matter which is the end user that is using the portlet. While this is a conventional solution used from old in many IT scenarios, for a SOA scenario exhibits a series of defects:

- Using this generic user, portlets are allowed to access any service of the application, which effectively switches the responsibility of authorization in the runtime environment from the administrator of the services, to the portlet developer. Authorization policies become hardwired in the portlet source code, and the administrator has no means to define or change them.
- The authorization rules lie entangled with many other conditions in the source code, and it is difficult to examine and understand them.
- The developer may easily make a mistake allowing the end user to invoke a service for which he may not have permission. This may be difficult to spot, and also difficult to change since the authorization rules need of changes to the source code and redeployment of the application.
- If the generic user credentials become compromised, every service becomes accessible and compromised too.

In short, the net effect of this common solution is close to disabling IS security and yielding it to the portlet developers, which is not a good practice in general, but less for a SOA in which the services are a first-grade element that may be used from many other parts different than a portlet application.

Applicability

Products: My webMethods Server Integration Server
In a SOA, security should be defined at service level, in a way that is effective for all potential usages of the service. A useful way of doing this is to define authorization rules for end users and groups directly, and not for generic or artificial users, so no matter which is the means the user utilizes to reach the service (a MWS portlet, an Office tool or just an HTTP URL).

This is possible in webMethods 7.1.2 thanks to the Central Users feature and the **SAML authentication between IS and MWS**, which makes it possible that IS trusts MWS regarding the authentication of a given request. This effectively provides a **Single-Sign-On between IS and MWS**, so that requests made by a portlet in behalf of its end user are authenticated in IS with the credentials of that user. This makes it possible to define different authorization to services to different users, improving manageability and auditing.

**It is easy**

Enabling this mechanism is really easy, although as of webMethods 7.1.2 it is not yet documented. Briefly, the steps are:

1. Configuring the portlet to use SAML hybrid authentication
2. Configuring the ACL of the services in IS

This SAML authentication is easily enabled in the advanced or expert properties of the web service connector used by the portlet, which are shown by clicking on the icon in the **Bindings** view of webMethods Designer. Once we click this icon, if we open the tree for the web service connector, we will see that under the **Authentication Info** node there is a new entry named **Authentication Method**, which is an integer value. Apparently, the valid values for this property are:

- **1**: Basic authentication, i.e. user and password go encoded in Base64 inside the **Authorization** HTTP header
- **2**: SAML authentication, where a SAML token goes inside XML SAML markup in the SOAP header
- **3**: Hybrid authentication, a mix of the two above, in which the SAML token goes inside the **Authorization** header
- **4**: Nonce. This probably refers to a **unique number** generated by either IS or MWS to encrypt a password, but the details are unknown.
From these, Integration Server supports 1 and 3 (not 2). Hence, to enable SAML authentication, it suffices to set a value of 3 in this property:

And that's about it as what is related to the MWS side (watch out: for portlets created before Designer 7.1.2, there may be additional issues, described later.) After that, when the portlet issues the call through this web service connector, it will generate a new SAML token, valid only for that request, that will travel in the request made to IS (in the same Authorization HTTP header), and when IS receives it it will ask MWS which user corresponds to this token. Further details are given later in this article. Of course, the user known to MWS will not be a internal IS user, but rather a global user defined in the Central Users directory. Hence, for all this Single-Sign-On to work, the following must be also ready:

- IS must be using the Central Users feature, so that it is able to see the users defined in MWS
- The ACL for executing the service must include the MWS user

Besides, since IS must call MWS back in order to resolve the SAML token into a valid identity, the URL for this call must also be configured. All this is described below, followed by some troubleshooting tips.

**Hands on: tutorial**

To illustrate this feature, we will perform a simple tutorial in which a MWS portlet will call a IS service. It is simple, although it is long because the whole process of creating
a brand new portlet is described, so developers with CAF experience may want to skip most of it.

The steps of the tutorial are:

- Configure Central Users in IS
- Check the samIResolver
- Have a service to be called
- Creating the portlet
- Choosing a MWS user
- Configuring the IS ACL
- Running the portlet

Configure Central Users in IS
First we need to make sure that the Central Users directory is configured. For this, it suffices to go the IS administration console and then to the Security > User Management option. It should show something like:

```
Security > User Management

Central User Management Configuration

General
Central User Management Configured
```

If not, you have to configure Central Users, which is also easy. You just have to go to again to the the IS administration console and then to the Settings > JDBC Pools option, and then define the CentralUsers pool so that it points to the MWS database. For more information, see section “Configuring Central User Management” in chapter 17 of the Integration Server Administrators Guide.
Check the samlResolver

Integration Server must know to which address has it to connect to resolve the SAML tokens received into valid user identities. After IS installation, this address points to MWS, but it is worthy to check that it is right. This can be done by reviewing the Extended Settings in the IS administration console, checking the value of the watt.server.auth.samlResolver property:

- Edit Extended Settings
- Show and Hide Keys

Extended Settings

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>watt.server.auditDir</td>
<td>audit</td>
</tr>
<tr>
<td>watt.server.auditGuaranteed</td>
<td>true</td>
</tr>
<tr>
<td>watt.server.auth.connector</td>
<td>ftpBase</td>
</tr>
<tr>
<td>watt.server.auth.samlResolver</td>
<td><a href="http://localhost:8800/services/SAML">http://localhost:8800/services/SAML</a></td>
</tr>
<tr>
<td>watt.server.kecAliveTimeout</td>
<td>180000</td>
</tr>
<tr>
<td>watt.server.tspace.location</td>
<td>c:/largeDoc</td>
</tr>
<tr>
<td>watt.server.tspace.max</td>
<td>500000000</td>
</tr>
<tr>
<td>watt.server.userFtpRootDir</td>
<td>packages\AdvTrainingIS\pub\ftpBase</td>
</tr>
</tbody>
</table>

Extended settings are typically provided by webMethods Support

If you cannot see this property, remember that you can use the Show and Hide Keys option to show it.

Have a service to be called

After that we need an IS service to be called from the portlet, that may be as simple as you want. For the tutorial you may use the enclosed helloSupport.zip package, which provides a simple hello service that accepts a name as parameter and returns a greeting message. This message includes also the name of the actual user logged in IS, so it is easy to verify that IS used the right user to execute the service.

To import this package in IS, copy HelloSupport.zip to the <Integration Server>/replicate/inbound directory, go to the IS administration console, and then to Packages > Management > Install Inbound Releases option. Then select the
Once we have a service to call, we will create the portlet that calls it.

Creating the portlet
Now follows the step-to-step instructions to create the portlet that will call our service; if you already know about webMethods CAF probably you will know them perfectly already.
You should have both My webMethods Server and Integration Server running. Then:

1. Open webMethods Designer
2. Open the My webMethods perspective
3. If you have not configured the IS server, go to Window > Preferences and then to webMethods > Process Development > Logical Servers:

4. Double click on the Default logical server, enter the correct URL, user and password, and then click Connect:
5. Close the preferences dialogs by clicking the OK buttons. Then, in the Solutions view, right-click on User Interfaces and New Portlet Application.
6. Enter a name for the project (e.g. helloSAML) and then click on Finish:
7. In the Solutions view, open the User Interfaces node, then right-click on the name of your portlet, and select the New Portlet option.
8. Click on Next, then enter a portlet name (e.g. hello) and then click on Finish.

9. A new portlet view will be open in your workspace. Then go to the IS Browser view (it usually lies at the bottom left of the workspace) and navigate to the IS.
10. Then click on it, and drag it to the smaller rectangle on the new view, when the tooltip reads “Place inside HTML Form (id=defaultForm)”: 

11. Designer will take some time in generating several Java classes, the Web Service Connector and the UI to call the service. When it finishes our view
12. At the bottom right of the screen we should have the **Bindings** view. In it we will navigate until the Web Service Connector named **Hello 2**, and also to its **Authentication Info**:
13. If we then click on the button to show the Expert / Advanced properties, we will see the **Authentication Method** option. The tooltip is providing the values accepted for this property:

![Authentication Method](image)

14. As described before, all we have to do to enable SAML is to put a 3 in this property, in the **Data Binding** tab of the **Properties** view:

![Data Binding](image)

As a small caveat, after typing the value, press the TAB key in order for Designer to really take it.

15. A curiosity here is that, after changing the value, Designer will update the associated Java code, and it will save it without telling us about. However, it is good to double check and save the portlet by clicking on the save icon.
Once we have our portlet, we have to deploy it to My webMethods Server. For this,

1. We should open the Servers view, usually at the bottom of the workspace

![Servers view](image)

2. There, if we have not a Remote server already, we should create one by right-clicking on the view and selecting the New > Server option:

![Server creation](image)

3. Then we provide the server host name and select the My webMethods Server (Remote) server type. If the server is in the standard port (8585) we can just
If the server is not in that port or has some other differences, we should rather use the Next button (not shown here).
4. Once we have the server defined, we have to add our project to it by right-clicking on it and selecting the Add and Remove Projects option:

5. Once we have added the project, we click on Finish:
6. To deploy the portlet to the server we just have to click on the publish button. We will be asked for credentials authorized to publish in MWS; usually they are user `sysadmin`, password `manage`:

![Authentication Request](image)

After that, we will have our portlet deployed in MWS and ready to run.

Choosing a MWS user

Once the portlet is deployed in MWS, we have to choose a MWS user that we will be using to run the portlet. Typically, the `Administrator` user is chosen, but in this case we won't be doing so because this Single-Sign-On feature should be most useful for regular users.

Now we will describe now how to create a new regular user and role inside MWS, so if you already have such user you can skip it:

1. First we log to MWS in as `Administrator` (usually the password is `manage`):
2. Then we go to the Administration menu

![Administration menu](image)

3. There we click on the Users tab

![Users tab](image)

4. And then on the Add User button

![Add User button](image)

5. We can put any user id and password we want, and then click on the Create button:
To give permissions to this user, it must first pertain to a role or group, but by default our new user will already be a member of the *My webMethods Users* role. We will use this one. In more realistic situations, this user would be a member of a custom role (e.g. one specific of an application), which would be used later to configure the ACL inside IS.

**Configuring the IS ACL**

We will leave the default ACLs in IS as they are, and instead we will create a new one, assign it to the service, and add the MWS role to it:

1. We log in to the IS Administration console, go to *Security > ACLs* and then click on *Add and Remove ACLs*
2. There we enter the name of our new ACL (e.g. HelloApp) and click on Create ACLs

Security > Access Control Lists > Add an

* Return to ACLs

3. Then we select our ACL in the Select ACL list and then click on the Add button

Security > Access Control Lists

1 ACL(s) added

* Add and Remove ACLs

4. A Select Role/Group dialog will appear. On the Provider list inside it we have to choose the Central option, and then search for e.g. webMethods, and results
like these should show up:

<table>
<thead>
<tr>
<th>Role/Group Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>My webMethods Users</td>
<td>Role</td>
</tr>
<tr>
<td>My webMethods Administrators</td>
<td>Role</td>
</tr>
</tbody>
</table>
5. There we just have to click on the My webMethods Users, and it will be added to our ACL:

![ACL Membership](image)

6. It would be useful to add also the local administrators group to the ACL, so that the IS Administrator account can execute the service (e.g. from Developer). If we want, we can also add the global My webMethods Administrators role, so that the MWS Administrator user can also execute the portlet.

7. After that, we click on the Save changes button and we will have our ACL created.

Once we have our ACL, we should assign it to our service. For this, we have to:

1. Enter in webMethods Developer, or refresh it with the button if we had it already open (so that the new ACL is shown)
2. Lock for edition and edit the service we are calling from the portlet (e.g. helloSupport:hello), and click on its white canvas to show its properties at the
3. We will click on the combo at the right of **Execute ACL**, and select the name of the ACL we created before (e.g. **HelloApp**):

4. Then we save the service by clicking on the ![button](image)

**Running the portlet**

Now everything is set up to run our portlet and see whether our SAML authentication works. For this, the most direct way is to:

1. Log in to My webMethods Server with the user we created previously (e.g. **harry_potter**)
2. Once logged in, a quick way to access the portlet is entering a URL such as:

   http://portlet/projectName___viewName

   Note there are 3 underscores (_) between **productName** and **viewName**. E.g. in the case of our tutorial, it would be:

   http://portlet/helloSAML___hello
If you have problems with this way of accessing the portlet, you may use a more canonical means, which consists in creating a public folder and adding the portlet to it. You can do this easily from the MWS Admin view of Designer, or from the sysadmin account in MWS, but at any rate it is not described in this article. You may also try to log in as sysadmin, navigate to the portlet through System > Portlets and then copy the portlet URL to be used when logged in as the regular user.
3. Once you have accessed the portlet, we just have to enter a name and click on its Refresh button:

If everything goes well, after clicking Refresh you will receive the message from the service, greeting the name you entered in the input box. Besides, the hello service provided in this tutorial is also showing the actual user logged in IS to execute the service, that should be the same one you are using in MWS. Otherwise, look at the Troubleshooting section below.

**Troubleshooting**

The truth is that, although using this feature is quite easy, still there are a number of trouble that may arise, making the situation complex. Besides, normally the error messages shown are not much useful.

If your user does not have the right IS permissions to execute the service, you may receive an error such as org.xml.sax.SAXParseException: Premature end of file. or java.net.SocketTimeoutException: Read timed out. You may also receive a java.rmi.RemoteException: HTTP error response: [ISS.0084.9001] Invalid credentials, which at least is close to the cause, but it is not really true (IS is complaining about an invalid SAML token, and not about a not authorized user). In that case, verify that the
ACL is assigned to the service, and that the ACL is associated to a role or group which in turn contains the MWS user you are using.

Another thing to verify, as described before, is that you have the following property set in the IS Extended Settings of the IS Administration console:

```
watt.server.auth.samlResolver=http://localhost:8800/services/SAML
```

This URL should point to a valid address in your MWS; if you just open it in a browser and you should receive a message about *Hi there, this is an AXIS service!*! If the URL is right but this message is not shown, you may have some issue with your MWS installation.

There is also another property of the web service connector, *Requires Auth*, that should have a value of *true*:

![Screenshot of Requires Auth property](image)

It seems that Designer 7.1.2 sets properly this in the Java code when the connector is created, but that older versions did not so. Hence, when trying for a portlet created with an older version of Designer to use the SAML feature, it may be that this value is not set, and then although we do all the rest of things the SAML token will not be sent and the identification will not work.
When in doubt about the true values of the properties, it may be useful to check the Java code in which they are actually stored. It may be quickly viewed just by clicking in the Source link shown at the upper right corner of the view:

```
public class HelloDefaultViewView extends com.webmethods.caf.faces.bean.BasePageBean {
    /**
     * Determines if a de-serialized file is compatible with this class.
     * Maintainers must change this value if and only if the new version
     * of this class is not compatible with old versions. See Sun docs
     * for a <a href=http://java.sun.com/j2ee/1.5.0/docs/guide/serialization/spec/version.html>
     * details.</a>
     */
    private static final long serialVersionUID = 1L;
    private static final String[][] INITIAL_PROPERTY_EXPRESSIONS = new String[][] {
    }
    private com.webmethods.caf.hello.Hello hello = null;
    private com.webmethods.caf.is.excellent.hello.Hello hello = null;
    private static final String[][] HELLO_PROPERTY_EXPRESSIONS = new String[][] {
        //$(hello, authCredentials.requiresAuth), false
        //$(hello, authAliases, authenticationMethod), "Hi"
        //$(hello, endpointAddress), $(environment["excellent-endpointAddress"]) /"
    },
    };
    /**
     * Initialise page
     */
```

If this Requires Auth value is not set in the Java code, by default it is taken from the web application environment variables, inside the web.xml file. This variables may be set at runtime using the CAF Application Properties page in MWS (the shortcut is http://<MWS>/folder.admin.caf.app.runtime.config).

However, mind that you reboot MWS after changing this value, since apparently there is some kind of cache inside MWS which is storing the parameters for web service connectors, and I don't know how to refresh it. Specifically, if the Authentication Type or Requires Auth properties are changed after running the portlet, MWS must be rebooted in order for this change to take effect.

You may also examine the IS server or error logs for more information. You may find errors such as [ISC.0088.0001E] SOAPException: [ISS.0088.9134] Exception occurred while processing the body of the message or [ISS.0053.0002C] Access denied for user SAMLart on port 5555 -> 'soap/rpc' from 127.0.0.1. Both point to a wrongly configured ACL, or maybe to one that has not been assigned to the Execute ACL of the service.

If you have fixed everything, and logged out and in from MWS, and still there are still errors, before rebooting MWS or IS you may try to kill the IS session of the user. MWS is establishing a session with IS, and reusing it for subsequent calls with the same user; you may go to the IS administration console, at Statistics, and then click on the count of Total Sessions. You will be given a list of active sessions; you may kill the ones established with the global user you are using in MWS.
How does SAML hybrid authentication work

You may see it if you enable TRACE logging in IS, and use the `wm_soapmonitor` portlet to trace the SOAP calls between MWS and IS, or some other HTTP sniffer (the SOAP monitor does not show every actual request being made).

When MWS is making the call to IS, it is using the same headers that for Basic authentication, with a special user called `SAMLart`. The password however is a SAML token, not a real password, which has been automatically generated by CAF/Glue and that is associated to that request from that user. To validate this token, IS does a call back to the MWS SAML service (to the URL specified in `watt.server.auth.samlResolver`, described above). This SAML service responds with the user to which this token corresponds, and then IS trusts this assertion, and uses this identity to authenticate the session.

This token is unique for every request, and once validated once it is not valid anymore. One can see this in the trace logs, when the user has no permissions, since the request is issued twice and only the first one passes SAML validation.

Apparently, when the service is stateless, this call back to MWS is done for every request, which does not sound like good for performance. However, when the service is stateful, since the MWS request comes with a session id inside, the IS already uses the authentication inside that session, and does not attempt to validate the SAML token. Hence, the call back to MWS is done only the first time, when the session is being established. Therefore, regarding performance, it looks like advisable to define the services as stateful, instead of stateless. This is worth mentioning because standard recommendations for IS performance demand precisely that services are stateless when possible, because this saves resources in the IS side. Also, please note that although service caching is usually used only for stateless services, it can be also
safely used for services marked as stateful (as long as the service is not truly stateful, i.e. its results do not depend on the previous activities made in the session).

Please note that this mechanism is applicable only to custom portlets, and not to existing applications running in MWS (e.g. Monitor). Also the TN portlets inside MWS make use of this feature.

**Summary**

This article provides details about how to enable Single-Sign-On between custom portlets and IS services, which provides a convinient way of avoiding IS user and password configuration issues, and at the same time results in a proper degree of authorization control from a SOA standpoint. The mechanism is easily enabled by setting a property of Web Service Connectors in webMethods Designer, and by configuring security in Integration Server.

---

The information in this document is provided as-is and with no warranty whatsoever. Software AG cannot be held responsible for the results of any inaccuracies.