Enabling Role-Based Access Control Using XACML

Many organizations expose their business capabilities through APIs. One of the key challenges is controlling access to these exposed APIs in such a way that all authorized users are able to access its APIs without any interruption, while at the same time making sure that any unauthorized users are kept out. In order to achieve this, parameters such as the user role can be used in determining whether to grant or deny access to an API for a given user. OAuth 2.0 scope is one way of controlling access to users. This section explains how an external extensible Access Control Markup Language (XACML) entitlement server can be integrated with WSO2 API Manager to provide role-based access control to APIs exposed via WSO2 API Manager. XACML is a declarative access control policy language based on XML that can provide a standardized way of validating authorization requests.

WSO2 API Manager provides the capability to authorize users based on OAuth 2.0 tokens and this mechanism can be extended to provide role-based access control using OAuth 2.0 scopes. However, as opposed to using OAuth 2.0 scope to provide authorization, XACML provides a standardized way of validating authorization requests. Authorization policies can be written in a standardized way using XACML and can be stored and managed through a policy administration point (PAP). Since the policies are standardized, policies written to one XACML engine can be ported to another engine from a different vendor without any issue. Similarly, XACML provides more control on how access should be enforced as different parameters and possibilities can be evaluated. XACML also provides ‘Obligations’ and ‘Advice’ as part of the XACML response that can be used by the API Manager when enforcing the policy decision to implement fine-grained access control for APIs.

How XACML is used with WSO2 API Manager

The diagram shown below depicts the scenario where WSO2 API Manager uses the XACML entitlement server to validate API requests that come into the API Manager. In this case, WSO2 Identity Server has been used as the XACML entitlement server.

The process is initiated by an administrator who creates the XACML policies and adds them to the PAP. The created policies are stored in a policy repository and promoted to the policy decision point (PDP) by an authorized user. Once the policy is deployed, authorization requests are evaluated against this policy. There can be more than one policy deployed in the PDP.

The API Manager acts as the policy enforcement point (PEP). Whenever an API invocation comes to the API Manager, an authorization request is sent to the PDP with the required attributes. In this case, it can be the name of the user, resource path and the HTTP verb. The PDP receives the request along with these attributes and evaluates the request against the existing policies deployed in the PDP. If the request requires more information, the PDP tries
to obtain that information from a policy information point (PIP). In this case, the request from the API Manager can contain the username and the policy that is deployed requires the role of the user. In such a scenario, the PDP gets this information from the user store that is defined as a PIP. Once the PDP has the required information to evaluate the request, a response is sent back to the API Manager with its policy decision.

**Enabling role-based access control**

The steps below demonstrate how WSO2 Identity Server, acting as a XACML entitlement server, can validate authentication requests from the API Manager based on a set of predefined XACML entitlement policies. This allows a standardized way of defining entitlement policies that can be enforced from WSO2 API Manager.

Let’s take the following requirement in exposing an API via the API manager.

<table>
<thead>
<tr>
<th>API</th>
<th>Resource Path</th>
<th>HTTP Verb</th>
<th>Who can access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/student/info</td>
<td>GET</td>
<td>Staff, Admin</td>
</tr>
<tr>
<td>api.educollege.com</td>
<td>POST</td>
<td>Add student info</td>
<td>Admin</td>
</tr>
<tr>
<td></td>
<td>/staff/info</td>
<td>GET</td>
<td>Student, Staff, Admin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POST</td>
<td>Admin</td>
</tr>
</tbody>
</table>

Based on the requirement, a single API is exposed to add or retrieve student and staff information. Each member type (staff or student) is identified from the resource path. The operation (GET or POST) that needs to be performed is distinguished by the HTTP verb. Follow the steps below to implement this kind of role-based access control.

1. Let’s start by creating the required users. First, you need to link both the API Manager and the Identity Server to the same user store in order to share the users, roles and other related information. This can be done by linking the API manager with the LDAP user store within WSO2 Identity Server. For more information, see Configuring Primary User Stores.

   In an actual deployment, both these servers can be linked to the user store of your organization.

2. Create user information with the following permission structure.

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>staff_user</td>
<td>staff</td>
</tr>
<tr>
<td>student_user</td>
<td>student</td>
</tr>
<tr>
<td>admin_user</td>
<td>college_admin</td>
</tr>
</tbody>
</table>

3. Install the required XACML features on the API Manager. The default API Manager distribution is not shipped with these features.
   a. Start the API Manager server and log in to its Management Console.
   b. In the Configure menu, click Features.
   c. Click the Repository Management tab and add a new repository with the following details:

      - **Name:** Wilkes
      - **URL:** http://product-dist.wso2.com/p2/carbon/releases/wilkes
d. Click **Find Features** in the **Available Features** tab to locate all the available features in the repository.

![Feature Management](image)

4. Once the features are installed, restart the server.
5. Start the WSO2 Identity Server and log in to its Admin Console.

Since API Manager and Identity Server run on the same server, offset the Identity Server by 1.

6. Under the **Entitlement** section, click **Policy Administration > Add New Entitlement Policy**.
7. You are redirected to a page listing all available policy editors. Select **Standard Policy Editor** from the list and add the values shown below in the policy editor.
   a. **Entitlement Policy Name**: EDUCollegePolicy
   b. **Rule Combining Algorithm**: Deny unless Permit

   When the rule combination algorithm is set to **Deny Unless Permit**, you need to set the permit criteria as a rule.

8. In the **Define Entitlement Rule(s)** area, set the following 3 rules to define the kind of requests and from which user they should be permitted.
   a. **AdminGrant** - grants full access to the admin user. Give the information below,
     **Rule Name**: AdminGrant
     **Conditions**: Subject is/are at-least-one-member college_admin
     Click the icon next to **END** shown below to configure the attribute value and attribute source to retrieve the user roles from the user store.
Select the attributes as given below. Note that this needs to be done for all the rules.

**Select Attribute ID:** Role  
**Select Attribute Data Type:** String  
**Entitlement Data Module:** Carbon Attribute Finder Module

b. GetStudent - allows staff users to get student information from the API. Give the information below,
   
   **Rule Name:** GetStudent  
   **Conditions:** Resource is equal /student/info AND Action is equal GET  
   **Subject** is/are at-least-one-member staff

Click the icon next to **END** shown below to configure the attribute value and attribute source to retrieve the user roles from the user store.
c. GetStaff - allows staff and student members to get staff information via the API. Give the information below,

**Rule Name:** GetStaff

**Conditions:**
- Resource is equal /staff/info
- Action is equal GET
- Subject is/are at-least-one-member staff
- Subject is/are at-least-one-member student

Click the icon next to **END** shown below to configure the attribute value and attribute source to retrieve the user roles from the user store.
9. Click **Add** once done.
10. The rules are added to the policy. Click **Finish** to save the policy.

Create XACML Policy

<table>
<thead>
<tr>
<th>Rule Id</th>
<th>Rule Effect</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminGrant</td>
<td>Permit</td>
<td><img src="edit-icon.png" alt="Edit" /> <img src="delete-icon.png" alt="Delete" /></td>
</tr>
<tr>
<td>GetStudent</td>
<td>Permit</td>
<td><img src="edit-icon.png" alt="Edit" /> <img src="delete-icon.png" alt="Delete" /></td>
</tr>
<tr>
<td>GetStaff</td>
<td>Permit</td>
<td><img src="edit-icon.png" alt="Edit" /> <img src="delete-icon.png" alt="Delete" /></td>
</tr>
</tbody>
</table>

11. In the Policy Administration page, click **Publish to My PDP** to publish the policy to the PDP. You can test the service by clicking **Try**.
12. Download the entitlement-1.0-SNAPSHOT JAR file and add it to the <APIM_HOME>/repository/components/lib directory. This JAR file contains the APIEntitlementCallbackHandler class which passes the username, HTTP verb and the resource path to the XACML entitlement server. If you want to view the source code of the JAR, go here.

13. Restart the server once the JAR file is added.

14. Now, you need to create a sequence containing the entitlement policy mediator that can be attached to each API required to authorize users with the entitlement server. Create an XML file with the following configuration and name it EntitlementMediator.xml.

```xml
<sequence xmlns="http://ws.apache.org/ns/synapse" name="EntitlementMediator">
  <entitlementService xmlns="http://ws.apache.org/ns/synapse" remoteServiceUrl="https://localhost:9444/services" remoteServiceUserName="admin" remoteServicePassword="admin" callbackClass="org.wso2.sample.handlers.entitlement.APIEntitlementCallbackHandler"/>
</sequence>
```

15. Log in to the API Publisher and create an API.

16. Attach the custom sequence to the inflow of the message as shown below.
17. Save, publish and test the API to make sure that the requests specified in the 3 rules defined in step 8 are accessible according to the user role specified. For example, the GET/staff/info operation is only available to users with the roles, student and staff. If an anonymous user tries to access the GET/staff/info operation, it should fail.

If you encounter an error stating "org.apache.axis2.transport.jms.JMSSender cannot be found by axis2_1.6.1.wso2v16" when publishing the API, comment out the following JMSSender configuration in the <APIM_HOME>/repository/conf/axis2/axis2_blocking_client.xml file and restart the server.

```
<transportSender name="jms"
class="org.apache.axis2.transport.jms.JMSSender"/>
```

18. If you want to debug the entitlement mediator, enable debug logs in the Management Console for the org.wso2.sample.handlers.entitlement.APIEntitlementCallbackHandler class.