Creating a XACML Policy

At the root of all XACML policies is a Policy or a PolicySet. A Policy represents a single access control policy, expressed through a set of rules. A PolicySet is a container that can hold other Policies or PolicySets, as well as references to policies found in remote locations. Each XACML policy document contains exactly one Policy or PolicySet root XML tag. Because a Policy or PolicySet may contain multiple policies or Rules, each of which may evaluate to different access control decisions, XACML needs some way of reconciling the decisions each makes. This is done through a collection of Combining Algorithms. Each algorithm represents a different way of combining multiple decisions into a single decision.

WSO2 Identity Server XACML support will provide two methods of writing an XACML Policy.

- Some predefined commonly used XACML Policy template samples where you can use by editing according to your use case.

  To get more details on available XACML templates. Read me.

- A set of UI editors to create an XACML policy using UI configurations.

The following steps list how you can create an XACML policy using the management console.

1. Log in to Management Console by entering your username and password.
2. Navigate to the Main menu to access the Entitlement menu. Click Policy Administration under PAP.
3. Click Add New Entitlement Policy.

The Add New Policy page appears which gives the 6 ways of writing an XACML 3.0 policy. You can select one out of six methods to create the policy using UI according to your preference as follows.

- Simple Policy Editor
- Basic Policy Editor
- Standard Policy Editor
- Policy Set Editor
- Import Existing Policy
- Write Policy in XML
Simple Policy Editor

- This simple editor provides you the opportunity to create XACML 3.0 policies without any knowledge of XACML policy language. You can define your rules in a simpler manner and create an XACML 3.0 policy out of them. However, you need to have some knowledge about access control rules.

This editor is based on four categories which we are mostly talking about access control rules. i.e User, Resource, Action, and Environment where **User** is the person who is going to access the resource, **Resource** is an entity that we are trying to protect with access control rules, **Action** is, what user is going to perform on Resource and **Environment** is the time, domain or any other factors that could cause to control the user’s access.
In the Simple Editor, you can see the following,

**Entitlement Policy Name:** Name of the policy.

**Entitlement Policy Description:** A description of the policy.

**This policy is based on:** Define based on what entity, that you are going to write this policy.

---

**Note**

If you are writing policy based on web service, you can select “Resource” category and continue. Or less, if you are writing policy based on email domain of users. You can select “Subject” category and select the “Email” attribute Id and then define the email.

**You can define multiple permit rules:** As an example, “Only users in admin role can do GET” This rule can be defined as follows. You need to select “Role” attribute id for “User” and fill the text box with the role name (admin) and then fill the text box of near “Action” with action name (GET).

**Deny rule is automatically created as the final rule.** Permitted rules are evaluated from top to blow.

---

**Important**

- If you want to define value as java regexp expression, you need to embedded value in the curly brackets “{ }”

  Ex: `{ ^([a-zA-Z0-9_.-])+@wso2.com }`

  - If you want to define multiple values as OR or AND value sets. you can separate those multiple values with “|” or “&” separates

  Ex: `read | write | delete`

  Ex: `ReadRole & WriteRole`

  - If you want to define value as a greater or lesser than value. you can use “<” or “>” (We do not support “>=” or “<=”)

  Ex: `< 34`

  - If you want to define two values that are in a range, you can use square brackets “[ ]” and round brackets “( )”. And two values are comma “, “ separated.

  Ex: `[09:00:00+05:00, 16:00:00+05:00]` time between 09.00am and 04.00 pm

  Ex: `(18, 30]` greater than 18 and less than or equal to 30

---

A sample policy example: This policy is defined for accessing “foo” resource

**Following are the access control rules that policy needs to be satisfied**

- Rule1: Resources under “foo” collection can be read, written, edited and deleted by the users in the admin role.
- Rule 2: Resources under “foo/wso2 collection can be read by only the users in the WSO2 (whose email address is wso2.com domain) regex for email (^([a-zA-Z0-9_.-]+)@wso2.com) between 9.00 am and 4.00 pm (09:00:00+05:00, 16:00:00+05:00)
- All other access request to “foo” resource must be denied.

You can build the above-mentioned policy example using Simple Policy Editor as shown below. Here, “foo” can be the main resource and the other resource “foo/wso2 can be the child resource. If you have further resources to evaluate you can add them as child resources by clicking on the icon and create separate rules.

Basic Policy Editor

- This editor is based on four categories which we are mostly talking about access control rules. i.e Subject, Resource, Action and Environment.
- You can define a target and multiple rules in the policy. Rules can be ordered.
- You can plug any attribute value sources and select those attribute values when creating the policy; rather than filling text boxes by your own. By default, WSO2 registry resources, Roles of the underline user store and some pre-defined actions are the attribute value sources for the resource, subject and action attributes respectively. There are extension points that you can use to extend and bring more attribute values on to the policy editor UI.
A sample policy requirement:

- This access control policy is written for the patient records of Medi Hospital. Patient records are stored in a location under “/patient/” directory. Therefore we are defining access control rules for “/patient/” directory.
- Users can only access patient records from 09.00pm to 04.00pm.
- Patient records can be created, deleted by users in MedAdminstrator role.
- Patient records can be updated and read by users in MediStaff role.
- All other access requests to patient records must be denied.

Let's implement this sample policy using Basic Policy Editor,

**Step1:** Define a name for the policy.

**Step 2:** This is similar to defining the policy target element. Configure it such as “*policy is applied for resource attribute value /patient/* with reg-ex match*”. 
Step 3: Define the 1st rule. The rule is “Users can only access patient records from 09.00pm to 04.00pm”. It means that if the user tries to access records any other time, the user must be denied. Therefore, we are writing a deny rule for users who are not accessing patient records from 09.00pm to 04.00pm. Therefore first, let's define a rule name and also define the rule effect as “Deny”. Then Select “Time” from the environment. Select functions as “is not” and “greater than and less than”. Write the time with GMT offset value. After defining you can add this rule into the policy.
Step 4: Define the 2nd rule. Rule name must be given. Then select “Role” as user’s attribute. You can select your “MedAdminstrator” role name from user attribute source. So just click on the icon.
**Step 5:** You can do a search for attributes values.

**Step 6:** Select only the “MediAdminstrator” role from the attribute source.
**Step 7:** You can see a text box has been filled with the selected “MediAdministrator” role name. Now let define actions. Here let us make the function name as “at-least-one”. Then this rule would be satisfied even when at least one action is going to perform. Finally, let us add this rule into the policy.
Step 8: Let’s define the 3rd rule. Which allows “MediStaff” roles to access the resource with action read and edit. You can follow same steps 4, 5, 6, 7. Then finally, add this rule in to the policy.
Step 9: Finally define the rule to deny all other access, as following
Step 10: We have defined the target and rules. Now it is time to define the rule-combining algorithm. Let select is as “first applicable” Then rule effect of the 1st rule that is evaluated properly, would be the final result of the policy.

You can click on “finish” and finish policy creation.

Standard Policy Editor

- The standard policy editor is little similar to Basic Policy Editor. But it is basically designed for creating XACML 3.0 policy rules. Because there are several improvements with Obligation in XACML 3.0 when compared to 2.0. In XACML 2.0, obligations can only be added to policies and policy sets. But with XACML 3.0, rules can also contain Obligations. Therefore, if you want to get details of the obligations after the policy evaluated, It is good to use Standard Policy Editor.

- As in Basic Policy Editor, there is a place to define the conditions which make the rules evaluated as “The policy is evaluated only when following are matched” and a place to define entitlement rules.

- Advice is a newly introduced feature with XACML 3.0 which is similar to Obligations. But only different, when compared to Obligations, PEPs do not have to comply with advice statements. PEPs can consider or discard the statement. A common use case is to explain why something was denied. “User Bob is denied because he has not a valid email”

- Here the attribute "Define Policy Obligation or Advice" is not mandatory.
A sample policy requirement:

- This access control policy is written for the patient records of Medi Hospital. Patient records are stored in a location under “/patient/” directory. Therefore we are defining access control rules for “/patient/” directory.
- Users can only access patient records from 09.00pm to 04.00pm.
- Patient records can be created, deleted by users in MedAdministrator role.
- Patient records can be updated and read by users in MediStaff role.
- All other access requests to patient records must be denied.

Since this editor is very similar to Basic Policy Editor we can use the same steps from **Step 1** to **Step 9** to configure the above requirement in Advance Policy Editor.

**Step 10**: In advance, if we want to see the details of the obligation after the policy evaluated, you can

You can define a policy obligation or advice as follows:
Step 11: We have defined the target, rules, and obligation. Now it is time to define the rule-combining algorithm. Let select is as “first applicable” Then rule effect of the 1st rule that is evaluated properly, would be the final result of the policy.

You can click on “finish” and finish policy creation.

Note

There are Policy Combining Algorithms which are used by Policy Sets and Rule Combining Algorithms which are used by Policies. Each of the algorithms mentioned below has its Policy Combining algorithm and its Rule Combining algorithms as follows:

- Standard combining algorithms defined in XACML 3.0:
  - urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable
  - urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:first-applicable
  - urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:only-one-applicable
  - urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny

These algorithms are explained in detail as follows,

- **Deny Overrides:**
  This combining algorithm combines decisions in such a way that if any decision is a Deny, then that decision wins.
  Deny overrides is one of the safest combining algorithms since it favors a Deny decision. However, if none of the children return a Deny decision, then the combining algorithm will never produce a Deny.

- **Permit Overrides:**
  This combining algorithm combines decisions in such a way that if any decision is a Permit, then that decision wins.

The permit overrides combining algorithm can be interesting when:

At least one child must return a Permit for access to be granted overall regardless of restrictions.

One wants to return all the reasons why access is being denied. This is what one could call a “greedy deny overrides”. For instance, if the reason for not being able to view a resource is that (a) you are not the owner and (b) you are in the wrong department, then we could rework the previous example as follows. When any of the deny reason triggers, the response would be denied with all the applicable reasons for access being denied:
• Policy Set (deny overrides): role==manager AND action==view AND resourceType==resource
  • Policy 1 (permit overrides)
    • Rule 1: deny if resourceOwner != userId + Advice("you are not the owner of the resource")
    • Rule 2: deny if resourceDepartment != userDepartment + Advice("you are not in the same department as the resource")
  • Policy 2
    • Rule 1: permit

• First Applicable:
  This combining algorithm combines decisions in such a way that the final decision returned is the first one produced either of Permit or Deny.

  First applicable is useful to shortcut policy evaluation. For instance, if a policy set contains a long series of not applicable policies and one applicable policy which returns either of Permit or Deny, then if that policy comes first and does produce Permit or Deny, the PDP will stop there and not process the other siblings.

• Deny Unless Permit | Permit Unless Deny:

  In XACML there are 4 possible decisions: Permit, Deny, NotApplicable, and Indeterminate. Sometimes, it is desirable to hide the NotApplicable and Indeterminate decisions to only allow for Permit or Deny. It makes the PEP logic potentially simpler.

• Only One Applicable:
  This combining algorithm exists only for policy sets to combine policy sets and policies. It cannot be used to combine rules. With this combining algorithm, in order for either permit Permit or Deny to be returned, then only one of the children must produce a valid decision – whether Deny or Permit.

• Ordered Deny Overrides | Ordered Permit Overrides:
  The ordered combining algorithms combine decisions in the same way as their (unordered) cousins. In addition they bring the guarantee that policies, policy sets, and rules are considered in the order in which they are defined. The need to define an ordered combining algorithm stems from the fact the XACML specification does not specify whether order matters in the deny-overrides and permit-overrides combining algorithms.
When you want to create a set of policies to evaluate at one time, you can create a **Policy Set**. You can add policies as shown in the figure and Click "Finish" to create the policy set.

**Import Existing Policy**

You can add a policy by using a policy XML file.

- Write a policy in an XML file and upload it.

**Write Policy in XML**

Click **Choose File** and browse to the location of the policy in your local machine.
Create your own policy in XML.

Click **Finish/Upload** depending on the option you chose to create your policy.

**Related Topics**

To evaluate the policy you just created and see a sample request and response of it, see *Evaluating a XACML Policy*. 