

Understanding XACML Policy Language - XACML (Extended Assertion Markup Language) - Part 1

By amilaj.wso2.com

Created 2011-11-16 19:04

Introduction

There are number of articles written by WSO2 experts on XACML. In order to have an understanding on XACML evaluation model, you may go through ([1], [2]). In this article I will be more focusing on XACML policies and how we could define XACML policies.

Before going into details about XACML policies let me add some background knowledge on, how XACML engine works. It will give us a pellucid idea about XACML policy components. Then we will go through XACML policy elements, in detail and define a sample XACML policy.

The complete article consists of 2 parts. In the first part we will discuss about XACML policy language. In [Part 2](#) [1] we will define a sample XACML policy based on a defined use case. We will also discuss about UI support in "WSO2 Identity Server" for XACML.

Applies To

WSO2 Identity Server [2]	3.2.0
WSO2 Identity Server [2]	3.2.2
WSO2 Identity Server [2]	3.2.3

Contents

- [Background](#) [3]
- [XACML Policy Elements](#) [4]
- [Summary](#) [5]
- [References](#) [6]

Background

In XACML, core logic related to policy evaluation resides in a software component called XACML Engine. WSO2 Identity Server has a XACML (Sun XACML) engine embedded.

XACML engine takes two inputs and gives a single output. Inputs it is taking are XACML Policies and a so called XACML Request. Output per request can be one of the following;

- **Permit** Request is evaluated against all applicable policies given to XACML engine and request is authorized to carry on the operation/actions
- **Deny** Request is evaluated against all applicable policies given to XACML engine and request is not authorized to carry on the operation/actions
- **Not Applicable** XACML engine didn't find any applicable policy for given request and request is not evaluated in XACML engine

Let's discuss inputs given to XACML engine.

XACML Policies

XACML engine can take multiple policies and evaluate request against all those policies. XACML engine evaluates policies independently. Result of evaluating a single policy is same as the results stated in previous section (Permit, Deny, and Not Applicable). So now the problem is how output from multiple policies is combined together. For that we have something known as Policy Combining Algorithm. This algorithm is responsible for combining outcome from multiple policy evaluations.

- **permit-override** If at least one policy is evaluated as permit, the integrated output will also be permit.
- **deny-override** If at least one policy is evaluated as deny, the integrated output will also be deny.

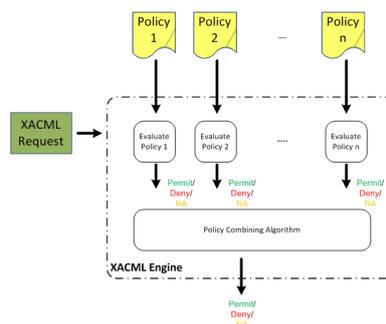


Figure 1

Figure 1, depicts the above mentioned behavior.

XACML Request

A XACML request contains information necessary to take authorization decision. Basically it contains attribute names and attributes values. When evaluating the policy, attribute names and attribute values are compared according to criteria defined in the policy. A sample XACML request is depicted in Figure 2.

```

<Request>
  <Subject SubjCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject-subject-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>am1a1c/AttributeValue</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource-resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>/sevice/EchoProxyService/echoString/AttributeValue</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action-action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>read/AttributeValue</AttributeValue>
    </Attribute>
  </Action>
  <Environment>
  </Environment>
</Request>

```

Figure 2

We will discuss more details about XACML request elements after discussing XACML policy model.

Applicability of XACML Policies

XACML engine will get all policies as inputs. But it might not be necessary to evaluate all of them for a given request. **So how does XACML engine filter applicable policies for a given request?**

The logic which says whether a given policy is applicable to a request is defined in the policy itself. Each policy has a XML element known as **Target**. The Target element's attribute values are matched with the incoming request attribute values. If those are matched with each other, XACML engine decides that the request should be evaluated against the complete policy.

XACML Policy Elements

Now you should have a basic idea how XACML engine processes XACML policies. Now let's focus on how actual evaluation is taking place.

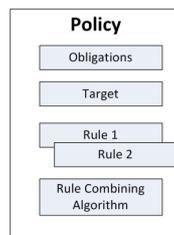


Figure 3

Mainly a XACML policy consists of 4 sub-elements. Those sub-elements are depicted in Figure 3.

Target Element

Every XACML policy should have a **Target** element. As described in **Applicability of XACML Policies** section, **Target** element decides whether a particular policy is applicable to a given request. The **Target** element contains 4 sub-elements. They are depicted in Figure 4.



Figure 4

The sub-element values are compared against the XACML (In Figure 2) request values. When comparing, all sub-elements are taken into account. If the request attributes match with ?Target?s attributes, the policy will be further evaluated else XACML engine decides that given XACML request is not applicable to the policy. Let?s look at what each of sub-elements represent.

Subjects ? This is a parent element which contains 1 or more ?**Subject**? elements. A **Subject** element usually represents the identity of the entity, which performs an action. Within the ?**Subject**? element we need to define matching criteria for policy. For that we are using another sub-element called ?**SubjectMatch**?. Within the ?**SubjectMatch**? we define the logic to match elements. A **SubjectMatch** element contains 2 parameters;

- Value of the subject attribute
- Name of the subject attribute

We can directly specify comparing attribute value with the relevant data type (in the policy). When XACML needs to retrieve attribute values from incoming Subject (in request), it uses ?**SubjectAttributeDesignator**?. Similar to SubjectAttributeDesignator, there are ?**ResourceAttributeDesignator**?s, ?**EnvironmentAttributeDesignator**?s and ?**ActionAttributeDesignator**?s. In the attribute designator we specify the fully qualified name of the attribute and its type. Though, this looks complicated it is essential retrieving attribute values from the request.

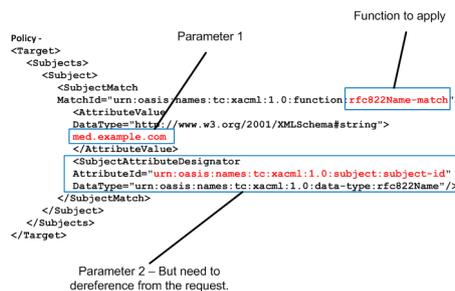


Figure 5

Figure 5 shows a sample policy target with a ?Subject? where you will also find a ?SubjectMatch? element. ?SubjectMatch? element defines a function called ?rfc822Name-match?. Function takes 2 parameters. Parameter 1 is directly specified in the policy, i.e. **med.example.com**, parameter 2 is specified as an ?**attribute designator**?.

So what are attribute designators?

Attribute designators instruct XACML run environment to look for values from the XACML request. As described in an earlier section, there are 4 types of attribute designators. They are ?SubjectAttributeDesignators?, ?ResourceAttributeDesignators?, ?EnvironmentAttributeDesignators? and ?ActionAttributeDesignators?. All attribute designators instruct XACML engine to look for values from the XACML request. ?SubjectAttributeDesignators? instructs XACML engine to look only for values within ?Subject? element (Figure 2). Similarly ?ResourceAttributeDesignators? will only look for ?Resource? and ?ActionAttributeDesignators? will only look for ?Action? in the XACML request. Same pattern is applied to ?EnvironmentAttributeDesignators?.

Assume I have following ?Subject? element in my XACML request.

```

Request -
<Subject>
  <Attribute AttributeId="urn:oasis:names:
tc:xacml:1.0:subject:subject-id"
  DataType="urn:oasis:names:
tc:xacml:1.0:data-type:rfc822Name">
    <AttributeValue>
      med.example.com
    </AttributeValue>
  </Attribute>
</Subject>

```

Figure 6

While evaluating Target element, XACML engine will first build an expression similar to following;

```

Evaluate function rfc822Name-Match (?med.example.com?,
[SubjectAttributeDesignator Value])

```

Now XACML ?run environment? will de-reference value in ?SubjectAttributeDesignator? after matching metadata in policy and XACML request (In other words XACML engine will retrieve appropriate values from request). After de-referencing XACML engine will evaluate a function similar to following;

```

Evaluate function rfc822Name-Match (?med.example.com?, ?med.example.com?)

```

Resources/Actions ??**Resources**?/? **Actions**?, is a parent element which represents a collection of ? **Resource**?/? **Action**? elements. A ?**Resource**? element usually represents the actual resource which user is trying to access. ?**Action**? element represents user?s activity on a resource (E.g.:- read, write, execute, etc ?). The process of comparing **Resource/Action** values in a policy and **Resource/Action** values in XACML request is quite similar to **Subject** (described above). Hence I will not discuss about matching **Resource/Action** elements.

Environment ? This element is used to define system wide attributes which effects the authorization.

E.g.:- If we want to apply certain policies only based on a domain, we can use this element. In ? **Environment**? element we can specify the domain name and the domain name will also be sent in the XACML request. We write the policy in such a way, where it will be evaluated only if domain names are equivalent (We can use XACML built-in functions to do comparison).

```

Policy -
<Environments>
  <Environment>
    <EnvironmentMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
        <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">wso2.domain</AttributeValue>
        <EnvironmentAttributeDesignator
          AttributeId="urn:oasis:names:tc:xacml:1.0:environment:envi
ronment-id" DataType="http://www.w3.org/2001/
XMLSchema#string"/>
        </EnvironmentMatch>
      </Environment>
    </Environments>

```

Figure 7

Figure 7 shows a sample policy (only a part of the policy) which has an **environment** defined. In this case we need the policy to be applied for requests only coming for ?**wso2.domain**?. Therefore we are comparing whether the values in ?**Environment**? element (in request) matches with the ?**wso2.domain**?. Again in order to refer values in the request, we use ?**environment attribute designator**?

Rule Element/s

A policy can have one or more rules. While target element evaluates the applicability of a policy, rule elements implement the actual authorization logic. Structure of a rule is depicted in Figure 8.

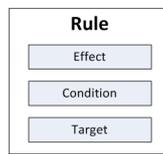


Figure 8

Target ? A policy can have multiple rules. But it is not necessary to evaluate all such rules for a given request. A rule has a target element similar to policy's target element. Role of this target element is to decide whether a rule should be evaluated or not for a given request. If there isn't a target, the rule will be evaluated for all requests applicable to policy.

Condition ? You can treat the condition as the core element of a rule. Within the condition we specify the exact authorization logic which always contains a Boolean expression. Based on the outcome of Boolean expression, the rule will be evaluated as true or false. Within the condition element we can use certain functions to implement authorization logic.

```

Policy-
<Rule Effect="Permit" RuleId="Rule_On_Withdraw">
  <Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Money</AttributeValue>
        <ResourceAttributeDesignator AttributeId="http://domain.wso2.org/resources/parents"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Withdraw</AttributeValue>
        <ActionAttributeDesignator AttributeId="http://domain.wso2.org/actions/parents"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Apply>
    </Apply>
  </Condition>
</Rule>

```

Figure 9

Figure 9 shows a sample XACML rule with a condition. On top of the rule we specify the **effect** of the rule. **Effect**, states the outcome of the rule if its condition is evaluated as true. In above example if rule's condition is evaluated as true, output of the rule will be **Permit**.

Then we have specified the **condition**. Within the condition we should have Boolean functions. In example given in Figure 9, we have an **and** Boolean function. Parameters of **and** function is specified within **and function element**. According to Figure 9 there are 2 parameters. Parameters are, function defined in green and function defined in blue. The first function (green) checks whether request's resource section contains a value called **Money**. Here we are using **ResourceAttributeDesignator** to check values in XACML request. The second function (blue), checks whether there is an attribute values in request's **Action** section with name **Withdraw**. So outputs from those 2 functions are given to **and** function.

Rule Combing Algorithm

In a policy we can have many rules. Each rule says whether a request is permitted/denied or **not applicable**. When we have multiple rules, how can we decide the final outcome of a policy? For that policy defines a **rule combining algorithm**. Rule combining algorithm is quite similar to **policy combining algorithm** described above (In XACML Policies section). In fact algorithms are also same as in **Policy combining algorithms**. i.e. **permit overrides**, **deny overrides** etc ?

Obligations

This is the least used element in a policy. This is an optional element and it is used to invoke certain actions upon a policy evaluation.

E.g.:- If policy is evaluated to ?permit? send a mail to head of the department. This can be used to audit certain authorization logic evaluations. But due to its performance reasons this is not widely used in practical scenarios.

Summary

Now you should have a basic idea about XACML policy language and its main components. In Part 2 we will define a sample XACML policy based on the concepts we learned from this article. Following I have summarized main points in this article.

- WSO2 Identity Server comes with a XACML run environment.
- XACML is used for fine grained authorization.
- An XACML run environment may consists of number of policies.
- Applicability of a policy is defined by the ?Target? of a policy.
- Attribute designators are used to refer XACML request and retrieve attribute values.
- A policy may contain one or more rules.
- Applicability of a rule is also decided by the ?Target? element within the rule.
- Rule condition specifies the exact authorization logic.
- Rules are combined using a ?rule combining algorithm?, similarly output of each policy is combined using a ?policy combining algorithm?.

References

- [Using XACML Fine Grained Authorization with the WSO2 Product Platform](#) [7]
- [Fine-Grained Authorization to RESTful Services with XACML](#) [8]
- [eXtensible Access Control Markup Language - Specification](#) [9]

[To Part 2](#)^[1]

[Articles Intermediate Cloud Computing Security](#)

© 2012 WSO2 Inc.

footer

- [Home](#)
- [Products](#)
- [Platforms](#)
- [Cloud](#)
- [Solutions](#)
- [OEM](#)
- [Events](#)
- [Contact](#)

• © 2013 WSO2 Inc

- [Legal](#)
- [Privacy](#)

• [Report a problem with this page](#)

```
var rw_ext_id=""; var pkBaseURL = (("https:" == document.location.protocol)
?"https://connect.wso2.com/wso2/" : "http://connect.wso2.com/wso2/");
document.write(unescape("%3Cscript src=" + pkBaseURL + "std/resource/script/rwts.js'
type='text/javascript'%3E%3C/script%3E")); rw_log(pkBaseURL, 4220);
```

Source URL: <http://wso2.org/library/articles/2011/10/understanding-xacml-policy-language-xacml-extended-assertion-markup-langue-part-1>

Links:

- [1] <http://wso2.org/library/articles/2011/10/understanding-xacml-policy-language-xacml-extended-assertion-markup-langue-part-2/>
- [2] <http://wso2.org/library/identity-server>
- [3] <http://wso2.org/library/articles/2011/10/understanding-xacml-policy-language-xacml-extended-assertion-markup-langue-part-1#background>
- [4] <http://wso2.org/library/articles/2011/10/understanding-xacml-policy-language-xacml-extended-assertion-markup-langue-part-1#xacml-policy-elements>
- [5] <http://wso2.org/library/articles/2011/10/understanding-xacml-policy-language-xacml-extended-assertion-markup-langue-part-1#summary>
- [6] <http://wso2.org/library/articles/2011/10/understanding-xacml-policy-language-xacml-extended-assertion-markup-langue-part-1#references>
- [7] <http://wso2.org/library/articles/2010/10/using-xacml-fine-grained-authorization-wso2-platform/>
- [8] <http://wso2.org/library/articles/2011/08/finegrained-authorization-restful-services-xacml/>
- [9] http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf