**XACML Architecture**

See the following topics for information on XACML and also on the architecture of XACML.

- Introducing XACML
- Why XACML
- XACML system architecture
- XACML terminology
- XACML engine architecture (PDP)
- Improvements in XACML 3.0

**Introducing XACML**

XACML (eXtensible Access Control Markup Language) is an XML-based language for access control that has been standardized by the Technical Committee of the OASIS consortium. XACML is popular as a fine grain authorization method among the community. However, there are aspects of XACML which surpass it being just a fine grain authorization mechanism.

Although XACML was introduced as a standard by OASIS in 2003, not many organisations that have been adapted it yet. This is because most of the organizations have not shown interest in moving towards a XACML solution for authorization. The following may be some reasons for this:

- Many software designer and developers do not have a clear understanding on the importance and advantages of XACML.
- It is a bit hard to implement a XACML solution rather than a typical JDBC or hard coded authorization system.
- Performances of XACML-based authorization system may be less than adequate.
- Complexity of defining and managing XACML policies.

However, current market trends indicate that there are some motivation for XACML-based authorization systems. This section includes some architectural and implementation details on XACML with an existing XACML engine.

If you are interested in going into the depth in XACML, You can find XACML specifications and other information here.

To summarize, XACML describes both an access control policy language, request/response language and reference architecture. The policy language is used to express access control policies (who can do what when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries(responses). The reference architecture proposes a standard for deployment of necessary software modules within an infrastructure to allow efficient enforcement of policies.

**Why XACML**

Most of the organizations still use legacy systems with inbuilt authorization logic. Sometimes, one organization contains a large number of information systems and applications that each system or application uses for their own process of authorization. Today, authorization has become more complex because users within organizations as well as outside the organization need access to shared data and have the need to collaborate efficiently. Therefore It has been very hard to manage those legacy, custom authorization systems. However, XACML offers a solution to this problem.

When we think about a traditional authorization system, it would mostly look like as follows:

- Authorization logics are hard coded into the programming source code
- Authorization logics are stored in a databases that could be only readable and understandable by the underlying application

The following are the some goals that current organizations are looking at from an authorization system.

- Can business managers (who should determine how access controls would be implemented) define and modify authorization logics?
• Can we find new IT technicians to manage these legacy systems? Especially when people who developed the system, left the organization.
• Can authorization logics be modified without any source code changes in a dynamic way?
• Is authorization system capable for evaluating following rule? “X resource can be accessed by the Users who are from example.com domain and whose age is greater than 21 years old”
• If we are going to implement a new information system with the organization, can we re-use the authorization logics of a legacy system?
• Can achieve find-grant authorization without defining large number of static combinations?
• Is authorization systems capable of answering following questions: “Can a user, Bob, transfer X amount from Y current account at 1.00pm?”

As we can understand, the above goals can not be achieved from a traditional authorization system. However XACML (eXtensible Access Control Markup Language) based authorization solution can provide all of these goals. This is because XACML:

• is a standard which is ratified by OASIS standards organization.
• is a policy language implemented using XML.
• supports Attribute Based Access Control (ABAC) and evaluation can be done with the additional data retrieved from Policy Information Point (PIP) which is defined by the XACML reference architecture.
• contains reference architecture which is provided to externalize the authorization system. The Policy Decision Point (PDP) offers authorization as a service in your infrastructure. Authorization algorithms can be removed from the application logic and applications can query the PDP via their own Policy Enforcement Points (PEP).
• provides fine-grained authorization with higher level of abstraction by means of policies and rules.
• supports dynamic evaluation of policies by using the Policy Information Point (PIP).

**XACML system architecture**

Policy Decision Point (PDP) evaluates policies against access requests provided by Policy Enforcement Points (PEP). To provide the decisions, PDP may also need to query a Policy Information Point (PIP) to gather descriptive attributes about the user or any other missing attribute in the request. Policy Administration Point (PAP) is used to manage the PDP and PIP functionality.

**XACML terminology**

The XACML reference architecture illustrated above is comprised of the following four ‘building blocks’.
- **Policy Enforcement Point (PEP):** The system entity that performs access control by making decision requests and enforcing authorization decisions. Basically this is the entity that sends the XACML request to the Policy Decision Point (PDP) and receives an authorization decision.
- **Policy Decision Point (PDP):** The system entity that evaluates an applicable policy and returns an authorization decision.
- **Policy Information Point (PIP):** The system entity that acts as a source of attribute values. Basically if there are missing attributes in the XACML request that is sent by PEP, PIP would find them for the PDP to evaluate the policy.
- **Policy Administration Point (PAP):** The system entity that creates a policy or policy set and manages them.

**XACML engine architecture (PDP)**

The WSO2 Identity Server is a major player in the XACML and open source world. The Identity Server supports XACML 3.0, which is based on Balana XACML implementation. As the source code, distribution and documentation are available for free, it is possible to analyze and understand the architecture behind it. This section provides some information regarding the architecture of the XACML engine (or the entitlement engine) of the WSO2 Identity Server.

The XACML engine of the WSO2 Identity Server has two major components, i.e., PAP and PDP. This section focuses on Policy Decision Point (PDP) only.

The following diagram represents the component architecture of the PDP.

![Diagram of XACML engine architecture (PDP)](image)

The following information provides more details on the various components available in the PDP architecture.

**Entitlement Admin Service**
All PDP configurations have been exposed via this API as a Web service. You need to have certain privilege to access this API. The following functions can be done by calling this API.

- Invalidating caches
- Refreshing policy, attribute, resource finder modules
- Retrieving PDP configurations
- Includes the function for testing the Test PDP with a given sample request or a sample request with given policies

**Entitlement Service**

Actual XACML authorization API has been exposed via this API as a Web service. This service supports the following three communication methods with PEP.

- SOAP-based Web service
- Thrift binary protocol (this is much faster than SOAP)
- WS-XACML (This is the standard way but not popular)

**Balana PDP**

This is the core of the engine. Balana PDP has been initialized with all the attribute finders, resource finders and policy finder modules that have been plugged with WSO2 Identity Server.

**Balana Test PDP**

This is same as Balana PDP. However, this instance has been initialized with the PAP policy store. It means, policies that are defined by PAP can be evaluated by using this PDP instance. There is no policy caching or decision caching with this. Therefore this can be only used for testing your policies in the PAP store. Web service API for Balana Test PDP has been exposed via the Entitlement Admin Service with method name “doTestRequest()”.

**Carbon Policy Finder**

This module finds policies from different policy stores to evaluate a XACML request. The following is a high level diagram of how this works.
Any number of policy finder modules can be plugged with the Carbon policy finder. You need to implement the CarbonPolicyFinderModule interface to write a new module and register it using the entitlement.properties configuration file. By default, WSO2 Identity Server is shipped with a Carbon registry-based policy finder module that can retrieve policies from a registry collection, where the collection path is configurable using a property value in the entitlement.properties configuration file.

All modules would be initialized and policies would be loaded into memory when the Carbon policy finder is used. Loaded policies are kept as a policy collection where you can use your own implementation to access them. A written policy collection implementation can be registered using the entitlement.properties configuration file. The default implementation uses an in-memory cache and creates an effective policy for a given XACML request.

Therefore, the Carbon policy finder always finds effective policies from the policy collection where it is recommended to be kept in-memory. However, if any update in the policy store happens, the Carbon policy finder can be re-initialized automatically by the module that you have written, or you can re-initialize it using the Web service API (Entitlement Admin Service). Also you can manually do it using the Identity Server Management Console.

When you are writing a new module, you need to consider the following in addition to policy retrieving and re-initializing.

- Policy ordering
- Policy combining algorithm
- How to retrieve referenced policies

**Carbon Attribute Finder**

This module helps to find the missing attribute for a given XACML request. The following is the high level diagram for both the Carbon attribute and resource finders.

Any number of PIP attribute finder modules can be plugged with the Carbon attribute finder. You need to implement the PIPAttributeFinder interface to write a new module and register it using the entitlement.properties configuration file. By default, WSO2 Identity Server is shipped with a DefaultAttributeFinder that communicates with the underlying user store of the Identity Server. The default user store of the Identity Server is ApacheDS where it can be changed using user-mgt.xml file. See [Configuring the Realm](#) for more information.

All modules would be initialized and supported attribute Ids are retrieved for each module. During runtime, the Carbon attribute finder checks for the support attribute Id and hands it over to the proper module to handle. A caching machine is used by the Carbon attribute finder for caching the findings or the attribute finder module itself can handle it.

**Carbon Resource Finder**

This is used to retrieve children or descendant resources of a given root level resource value. Basically it is used to fulfill the requirement of a multiple decision profile. The implementation is same as the PIP attribute finder module. By default, WSO2 Identity Server is shipped with DefaultResourceFinder (which is the implementation of PIPResourceFinder) that communicates with Carbon registry resources.

**Improvements in XACML 3.0**

The XACML 3.0 core specification highlights the following main changes in comparison with XACML 2.0.
Custom attribute categories

Custom attribute categories can be defined with XACML 3.0. However, in XACML 2.0, attributes have been organized into subject, resource, environment or action. For example, consider that you want to create an attribute category called "foo" in your policy and request. You can do it with XACML 3.0 without any issue. According to the XACML 3.0 policy schema, the category of XACML element is identified by a XML attribute called "Category".

In XACML 2.0 Policy, you can define the attribute designator element as follows, However, it must be a pre-defined category such as subject, resource, environment or action.

```xml
<ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
```

In a XACML 3.0 Policy, you can define it as follows. Category can be any thing as it is defined as an attribute of the `AttributeDesignator` element.

```xml
<AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Category="resource" DataType="http://www.w3.org/2001/XMLSchema#string"/>
```

Improvements in Obligation

There are several improvements with Obligation in XACML 3.0 when compared to 2.0.

One of the main improvements is the introduction of **Obligation Expressions**. This adds dynamic expressions into the obligation statements. For a more indepth understanding, see the following example:

Let assume that you want to do following with the Obligation: "On deny, inform the PEP to send an email to the user".

In XACML 2.0, you need to define the obligation element with the user email statically.

```xml
<Obligation ObligationId="send-email" FulfillOn="Deny">
    <AttributeAssignment AttributeId="email" DataType="http://www.w3.org/2001/XMLSchema#string">user@foo.com</AttributeAssignment>
</Obligation>
```

However, the user would not be same for each XACML request that is evaluated. Therefore it is not possible to configure the email statically in the `Obligation` element. Obligation can only inform PEP to send an email to user (it lets the PEP figure out the value of user's email).

```xml
<Obligation ObligationId="send-email" FulfillOn="Deny">
    <AttributeAssignment AttributeId="text" DataType="http://www.w3.org/2001/XMLSchema#string">please send email to user</AttributeAssignment>
</Obligation>
```

However, in XACML 3.0, the email of each user can be retrieved using PIP in a dynamic manner as we can define an expression element inside the `ObligationExpression`. Therefore, obligation can inform PEP to send an email to user@foo.com address.
In XACML 2.0, obligations can only be added to policies and policy sets. However, with XACML 3.0, rules can also contain obligations.

**Introducing Advice**

Advice is a newly introduced feature with XACML 3.0 which is similar to obligations. PEPs do not have to comply with advice statements; PEPs can consider or discard the statement. A common scenario is to explain why something was denied: “User bob is denied because he does not have a valid email”.

**Improvements in Target**

Since it is possible to define custom attribute categories, there are improvements in some related elements in the XACML 3.0 policy, when compared to 2.0. With XACML 3.0, the Target element can be used to define more complex authorization logic within itself when compared with 2.0.

In XACML 2.0, the Target element contains the OR and AND relationship between the same category. However, in XACML 3.0, AllOf and AnyOf elements have been introduced to Target element. That clearly helps to define the OR and AND relationship between different categories.

As an example, let look at a Target element. In XACML 2.0, we have an AND relationship between foo1 and foo2 resources and an OR relationship between bar1 and bar2 actions. However, we cannot create an OR relationship between a foo1 resource and bar1 action. so we cannot define something such as “Target would be matched when Bob can access the foo resource or do a bar action” by using the Target element.
XACML 3.0 has an AND relationship between “foo” resource and “bar1” role and an OR relationship between “bar2” action. So we cannot define something as “Target would be matched, when Bob can access foo resource and do bar1 action or do bar2 action.”
<Target>
  <AnyOf>
    <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-regexp-match">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">foo</AttributeValue>
        <AttributeDesignator MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Match>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">bar1</AttributeValue>
        <AttributeDesignator MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Match>
    </AllOf>
    <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">bar2</AttributeValue>
        <AttributeDesignator MustBePresent="false"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Match>
    </AllOf>
  </AnyOf>
</Target>

More Functions and Algorithms

XACML3 has introduced new String functions such as:

- urn:oasis:names:tc:xacml:3.0:function:string-starts-with
- urn:oasis:names:tc:xacml:3.0:function:string-ends-with
- urn:oasis:names:tc:xacml:3.0:function:string-contains
- urn:oasis:names:tc:xacml:3.0:function:string-substring

Some improvements to other functions such as
Also improvements to existing combine algorithms (deny-overrides, permit-overrides, ordered-deny-overrides and ordered-permit-overrides) and new two combine algorithms for policy and rule combining.

- `urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny`

**Improvements in XPath**

New XPath data type is introduced with XACML 3.0. In XACML 2.0, XPath is defined as a String and cannot define the context that the namespace prefix is going to resolve. Also XPath based multiple decisions scheme is introduced with XACML 3.0.

**Improvement in XACML Request and Response**

As it is possible to define custom attribute categories, many types of attribute categories can be in the XACML 3.0 request. XACML 2.0 request can contain only subject, resource, environment or action categories.

The XACML Response can contain additional data such as:

- Request attributes that are defined in the XACML response.
- Applicable policy ids for a given XACML request are defined in the XACML response.

**Multiple Decision Profile**

This is new profile which comes with XACML 3.0 and is not included in the XACML3 core specification. However, this is very useful for performance improvement between PEP and PDP. This is especially the case since PEP can request decisions for multiple requests in one XACML request.

**Administrative Delegation Profile**

This is also a new profile which comes with XACML 3.0. This allows you to define policies about who can write policies about what. For example, “Bob may issue a policy but only about resources in department X”.