Tech Lesson 3.1.1

Policy Authoring and Evaluation Basics

Goals:

1. Understand the basic structure of a XACML policy.
2. Understand the basic structure of a XACML request.
3. Understand how to evaluate a policy against a XACML request with the SunXACML library.
4. Understand the basic structure of a XACML response.

Summary:

In this Lesson, you will inspect simple XACML policies and XACML requests to learn the basic syntax of XACML. You will then use the SunXACML library to evaluate a policy against a request and learn how to read the XACML result. Also, you will be challenged with making edits to a request to achieve certain results.

Steps:

3.1.1.1 Inspect Permit-Policy.xml

Line 1 contains the standard XML header tag. Line 2 has an XML comment that contains copyright information (XML comments have no effect on XACML files). The **<Policy>** opening tag is on Lines 3 – 5. A XACML policy has either a top-level **<Policy>** element or a top-level **<PolicySet>** element[1].

Line 3 contains the OASIS XML namespace for the XACML policy language. We strongly recommend always including this namespace. Not including the namespace may make it more difficult, or even impossible, for your policies to be processed by some automated tools[2].

Line 4 contains the policy Identifier (**PolicyId**).

Line 5 contains the rule-combining algorithm (**RuleCombiningAlgId**). Rule combining algorithms govern how multiple rules are aggregated within a single policy and are covered in Lesson 3.1.6. The rule combining algorithm used here is "deny-overrides". No rule combining algorithm will have any effect on this **<Policy>** since this **<Policy>** contains only one **<Rule>**.

[1] The <**PolicySet**> element is covered in Lesson 3.1.7.
[2] There are automated tools available to assist with authoring, testing, and analyzing XACML Policies.

Lines 7 – 10 contain the **<Description>** of the **<Policy>**. **<Policy>**, **<PolicySet>**, and **<Rule>** elements can contain **<Description>** elements. The **<Description>** is for informational purposes only and has no effect on policy semantics.

The **<Target>** of the **<Policy>** is on Lines 12 – 22. A **<Target>** is a collection of attribute *predicates* organized into c*lasses*. In general, a *predicate* is defined as a statement that can be shown to be true or false, and in XACML, *predicates* are statements on attributes.

There are four *classes* of attributes in XACML: **<Subjects>**, **<Resources>**, <**Actions>**, and **<Environments>**. A *class* can contain one or more *instances*. The **<Resources>** *class* can contain one or more **<Resource>** *instances*; and so on. An *instance* can contain one or more *match-predicates*. The *match-predicates* in **<Action>** *instances* are **<ActionMatch>** elements; the *match-predicates* in **<Environment>** *instances* are **<EnvironmentMatch>** elements; and so on. The **<Target>** of this **<Policy>** contains *match-predicates* for only the **<Subjects>** *class*.

Lines 13 – 21 contain the **<Subjects>** *class* which contains a single **<Subject>** *instance* (Lines 14 – 20). Lines 15 – 19 contain the single <**SubjectMatch>** *match-predicate* in the single **<Subject>** *instance*. A *match-predicate* consists of three components: a **MatchId**, an **<AttributeValue>**, and an *attribute-reference*. A **MatchId** is a reference to a XACML function that returns a Boolean value[3]. An **<AttributeValue>** contains a **DataType** and a literal value. *Attribute-references* refer to attributes in XACML Requests. An *attribute-reference* can be one of *attribute-designator* or **<AttributeSelector>**[4]. *Attribute-designators* in the **<Subjects>** *class* are **<SubjectAttributeDesignator>** elements; *attribute-designators* in the **<Resources>** *class* are **<ResourceAttributeDesignator>** elements; and so on.

In the **<SubjectMatch>** on Lines 15 – 19, the **MatchId** (Line 15) is specified to be "string-equal". The **<AttributeValue>** (Line 16) has a **DataType** of "string" and a value of "Top Secret". The **<SubjectAttributeDesignator>** refers to the "SecurityClearanceLevelCode" GFIPM attribute. This *predicate* can be read: "the GFIPM Security Clearance Level Code of the Subject equals 'Top Secret'." A request by a user that has a Top Secret clearance will cause this *predicate* to be true.

Let's look at this *predicate* in more detail. There are three parts: (1) "the GFIPM Security Clearance Level Code of the Subject"; (2) "equals"; and (3) "Top Secret". The first part is an attribute, the second part is an operation that will result in true or false (Boolean operation), and the third part is a literal value. In general, a XACML *predicate* is a Boolean operation on two

[3] Only functions that take two primitive values (as opposed to collections of values, known as bags) as input are able to be used as a **MatchId**. A complete list of standard XACML functions that can be used as a **MatchId** is in Section 7.5 of the XACML 2.0 Specification.
[4] <**AttributeSelector**> elements are covered in Section 3.1.4.

attribute-expressions[5]. We define an *attribute-expression* as being a literal value, an attribute, or a manipulation[6] of an attribute.

In XACML, functions are used to build *predicates*. There are functions for common data operators, such as "equals" and "add", and more. A function will be specific to a certain **DataType** (e.g., "string-equals" and "integer-equals"). There are functions to do operations on strings, numeric values, Boolean values, date-time values, and more. A complete list of standard XACML functions is in Appendix A.3 of the XACML 2.0 Specification.

The **<Target>** of the **<Policy>** will be applicable to requests for which the "SecurityClearanceLevelCode" Subject attribute has a value of "Top Secret". When the **<Target>** of a **<Policy>** is applicable to a Request, then the **<Rule>** elements of that **<Policy>** are evaluated against the request.

A XACML rule, represented by a **<Rule>** element, is the articulation of an authorization. A rule contains an **Effect**, a decision of "Permit" or "Deny", and collection of *match-predicates* in a **<Target>**[7]. The *match-predicates* represent the authorized privileges. The **Effect** determines whether the rule is a positive or negative authorization.

Every **<Policy>**, **<Rule>**, and **<PolicySet>** element is required to have exactly one **<Target>** element[8], however, the **<Target>** may be empty. An empty **<Target>** matches every request. Also, every **<Policy>** element must specify exactly one rule-combining algorithm.

Lines 24 – 32 contain the single <**Rule>** of the <**Policy>**. The **Effect** of this Rule is "Permit", and the Rule Identifier (**RuleId**) is "Rule-1" (Line 24). An **Effect** can either be "Permit" or "Deny". Lines 26 – 28 contain the **<Description>** of the <**Rule>**. The <**Rule>** has an empty **<Target>** (Line 30), which means that this **<Rule>** is applicable to all requests.

When a **<Rule>** is applicable to a request, then the **<Rule>** evaluates to its **Effect**. Therefore, this **<Policy>** will evaluate to "Permit" for requests from Subjects that have a GFIPM Security Clearance Level Code of "Top Secret", regardless of any Resource, Action, and Environment attributes that may exist in the requests.

A XACML policy can evaluate to one of four decisions:

- "Permit" – the requested action is to be allowed.
- "Deny" – the requested action is to be prohibited.
- "NotApplicable" – the policy doesn't apply to the request.
- "Indeterminate" – there was an error during the evaluation.

---

[5] XACML *predicates* do not always have to be in this form, but the authors have never come across a *predicate* that could not be normalized into this form.
[6] Manipulations on attributes are covered in Lesson 3.1.5.
[7] A <**Rule**> can optionally contain a <**Condition**>. <**Condition**> elements are covered in Lesson 3.1.5.
[8] Every <**PolicySet**> element is required to have exactly one <**Target**> element as well.

3.1.1.2 Inspect Request-1.xml

A XACML request is the articulation of one or more Subjects (**<Subject>** elements) seeking to perform a single Action (**<Action>** element) on one or more Resources (**<Resource>** elements) in a single Environment (**<Environment>** element). Each **<Subject>**, **<Action>**, **<Resource>**, and **<Environment>** element contains a set of zero or more **<Attribute>** elements. Each **<Attribute>** contains an **AttributeId** (an attribute Identifier), a **DataType**, and one or more **<AttributeValue>** elements. Each **<AttributeValue>** contains a single, literal value that must match the **DataType**.

This request contains Subject (Lines 5 – 10), Resource (Lines 12 – 17), and Action (Lines 19 – 24) attributes. There are no Environment attributes as shown on Line 26. This request can be read: "A Subject with a GFIPM Security Clearance Level Code of 'Top Secret' is attempting 'write' access to 'Resource-1'."

The **AttributeId** of the single Resource **<Attribute>** is the standard XACML "resource-id" Identifier. Every request must contain at least one **<Attribute>** that has the standard XACML "resource-id" Identifier in at least one **<Resource>**.

3.1.1.3 Evaluate Permit-Policy against Request-1

First, let's manually determine what the result should be. The PDP will first determine the applicability of the policy's **<Target>** to the request. To do this, the PDP will evaluate the *match-predicates* of the **<Target>** using the attributes of the request.

The *match-predicate* in Permit-Policy contains an *attribute-designator*. An *attribute-designator* is a reference to a particular **<Attribute>** in a request. When evaluating an *attribute-designator* against a request, the PDP will attempt to locate the **<Attribute>** in the request that has the following properties[9]:

- The **<Attribute>** must be in the same *class* as the *attribute-designator*.
- The **<Attribute>** must have the same **AttributeId** and **DataType** as the *attribute-designator*.

If a matching **<Attribute>** exists in the request, then the PDP will retrieve the values of all the **<AttributeValue>** elements of the **<Attribute>** (recall that an **<Attribute>** can have multiple **<AttributeValue>** elements) as a **bag**[10] of values. The PDP then invokes the function specified by the **MatchId** of the *match-predicate* one time for each value of the **bag**. For each invocation, the PDP will pass in the literal value of the **<AttributeValue>** of the *match-predicate* as the first parameter, and a value of the **bag** as the second parameter. If at least one invocation returns true, then the *match-predicate* evaluates to true. If all invocations return false, then the *match-*

[9] An *attribute-designator* can also optionally specify an **Issuer**. If an **Issuer** is specified, then a request <**Attribute**> must have the same **Issuer** value in order to match the *attribute-designator*.
[10] A **bag** is a mathematical set in which a value can appear more than once.

predicate evaluates to false[11]. If no matching **<Attribute>** is found in the request, then the PDP will retrieve an empty **bag** and the *match-predicate* will evaluate to false[12].

The Subject attribute of Request-1 (Lines 5 – 10) will cause the *match-predicate* of Permit-Policy on Lines 15 – 19 to be true. The Resource attribute of Request-1 (Lines 12 - 17) will be ignored by Permit-Policy since the Policy is silent on Resource attributes. The Action attribute of Request-1 (Lines 19 – 24) will be ignored by Permit-Policy since the Policy is silent on Action attributes. All the predicates of the Target of Permit-Policy will match the request; therefore the single **<Rule>** of Permit-Policy should be evaluated. Since the **<Rule>** has an empty **<Target>**, it will evaluate to its **Effect** ("Permit"). Since this is the only **<Rule>** in the **<Policy>**, the **<Policy>** should evaluate to "Permit".

To continue this exercise, you must have downloaded a Virtual Machine Player from the Internet and the GFIPM-SP virtual machine per the guidance in Appendix D.

Now, execute SunXACML's SimplePDP[13] with Request-1.xml and Permit-Policy.xml, and output the result to Request-1_Permit-Policy_Response.xml and inspect the result. The command for running the SimplePDP can be found in Appendix A.

Note that SimplePDP does not output the XML declaration tag or XML namespace information in XACML responses.

A XACML response is contained in a **<Response>** element (Lines 1 – 8). There is one **<Result>** element (Lines 2 – 7) that corresponds to the Resource Identifier for which access was requested ("Resource-1").

The **<Decision>** is on Line 3 and is "Permit".

Lines 4 – 6 contain the **<Status>** of the result and Line 5 contains the **<StatusCode>**. Returning a **<Status>** is an optional feature of XACML. If a PDP returns a **<Decision>** of "Permit" or "Deny", then the **<Status>** should have a value of "ok" as it does on Line 5. We will not further investigate the status feature in this Guide.

3.1.1.4 Inspect Deny-Policy.xml

This policy consists of a top-level **<Policy>** element. The **<Target>** of the **<Policy>** is very similar to the **<Target>** of Permit-Policy. The **<Target>** of this policy is applicable to Subjects that have a GFIPM Security Clearance Level Code of "Confidential".

---

[11] See the XACML Reference Tables in Appendix C for complete details.

[12] There is an optional **MustBePresent** property of *attribute-references* that changes this behavior. If the **MustBePresent** property is true and no matching <**Attribute**> is found, then the *match-predicate* will evaluate to "Indeterminate". See the XACML Reference Tables in Appendix C for complete details.

[13] Follow the instructions in Appendix A: Common Tasks (Executing SimplePDP).

The **<Policy>** contains a single **<Rule>**, "Rule-1", which has an **Effect** of "Deny". The **<Target>** of "Rule-1" is applicable to requests to perform the "write" Action. This **<Rule>** (and thus the **<Policy>**), when evaluated against requests that are not performing the "write" Action, will evaluate to "NotApplicable".

3.1.1.5 Evaluate Deny-Policy against Request-1

First, let's manually determine what the result should be. The lone **<SubjectMatch>** of the **<Target>** of Deny-Policy (Lines 15 – 19) should match the lone Subject **<Attribute>** of Request-1 (Lines 6 – 9). However, the **<SubjectMatch>** will evaluate to false because the value of the Subject **<Attribute>** of Request-1 ("Top Secret") does not equal the **<AttributeValue>** of the **<SubjectMatch>** of Deny-Policy ("Confidential"). Therefore, the **<Target>** of Deny-Policy will not match Request-1, "Rule-1" will not be evaluated (even though it would have matched Request-1), and Deny-Policy should evaluate to a decision of "NotApplicable".

Now, execute SimplePDP with Deny-Policy.xml and Request-1.xml, and output the results to Request-1_Deny-Policy_Response.xml and inspect the result. Confirm that the **<Decision>** of the **<Result>** for "Resource-1" states "NotApplicable".

3.1.1.6 Challenge: Create a request that will be applicable to Deny-Policy

Using the Emacs text editor or another text editor provided with the GFIPM-SP, make a copy of the Request-1.xml file and name the copy "Request-2.xml". Open Request-2.xml. The value of the Subject **<AttributeValue>** on Line 8 should read "Top Secret". Change this value to a value that will make Request-2 cause Deny-Policy to evaluate to "Deny".

The solution to this Challenge is in Request-2-Solution.xml.

3.1.1.7 Evaluate Deny-Policy against Request-2

First, let's manually determine what the result should be. The Subject **<Attribute>** of Request-2 (Lines 6 – 9) should match the **<SubjectMatch>** of the **<Target>** of Deny-Policy (Lines 15 – 19). The Resource **<Attribute>** of Request-2 (Lines 13 - 16) should be ignored by the **<Target>** of Deny-Policy since the **<Target>** does not specify the **<Resources>** *class*. The Action **<Attribute>** of Request-2 (Lines 20 - 23) should be ignored by the **<Target>** of Deny-Policy since the **<Target>** does not specify the **<Actions>** *class*. Therefore, Rule-1 of Deny-Policy should be evaluated against Request-2.

The Subject **<Attribute>** of Request-2 (Lines 6 - 9) should be ignored by the **<Target>** of Rule-1 since the **<Target>** does not specify the **<Subjects>** *class*. The Resource **<Attribute>** of Request-2 (Lines 13 - 16) should be ignored by the **<Target>** of Rule-1 since the **<Target>** does not

specify the **<Resources>** *class*. The Action attribute of Request-2 (Lines 20 – 23) should match the **<ActionMatch>** of the **<Target>** of Rule-1 (Lines 37 - 41). Therefore, Rule-1 should evaluate to its **Effect** ("Deny"), and subsequently Deny-Policy should evaluate to "Deny".

Now, execute SimplePDP with Deny-Policy.xml and Request-2.xml, and output the results to Request-2_Deny-Policy_Response.xml. Confirm that the **<Decision>** for "Resource-1" is "Deny".

3.1.1.8 Evaluate Permit-Policy against Request-2

First, let's manually determine what the result should be. The lone **<SubjectMatch>** of the **<Target>** of Permit-Policy (Lines 15 – 19) should match the lone Subject **<Attribute>** of Request-2 (Lines 6 – 9). However, the **<SubjectMatch>** will evaluate to false because the value of the Subject **<Attribute>** of Request-2 ("Confidential") does not equal the **<AttributeValue>** of the **<SubjectMatch>** of Permit-Policy ("Top Secret"). Therefore, the **<Target>** of Permit-Policy will not match Request-2, "Rule-1" will not be evaluated (even though it would have matched Request-2), and Permit-Policy should evaluate to a decision of "NotApplicable".

Now, execute SimplePDP with Permit-Policy.xml and Request-2.xml, and output the results to Request-2_Permit-Policy_Response.xml. Confirm that the **<Decision>** for "Resource-1" states "NotApplicable".

A Note about Notation

XML elements, for XACML and data files, are written as they appear in XML documents, and are indicated in boldface text. For example: **<Policy>**.

XML attributes, for XACML and data files, are written as they appear in XML documents, and are indicated in boldface text. For example: **PolicyId**.

Values of XACML and data elements appear in double quotes. For example: "Permit".

We introduce some terms to serve as labels for certain groups of policy elements; these terms are used to enable discussions about groups of elements as a whole. These terms appear in italics. For example: *class*.

We use labels to refer to files, directories, and data items that exist in the accompanying virtual machine. These labels are used in the style of Linux environment variables – they begin with a dollar sign ($) which is followed by the label in all caps. For example: the label $POLICY_GUIDE refers to the following path on the virtual machine, "/home/guide/policy-guide".