An Introductory 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”.

The Technical Lessons

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>**.

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

[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.

[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.

<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 *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".

[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.

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.

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-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.

[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.
[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 AppendixA: Common Tasks (Executing SimplePDP).

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".

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".

3.1.2 The "Attribute Value Spacing" Pitfall

Goals:

1. Understand what the Attribute Value Spacing Pitfall, and why it is problematic[14].

Summary:
In this Lesson, you will compare two policies that have a subtle difference in the value of an **<AttributeValue>** element. You will evaluate both policies against the same request and analyze the different results.

Steps:

3.1.2.1 Inspect Permit-Policy.xml and Request-1.xml

Confirm that this Permit-Policy and Request-1 are the same as the Permit-Policy and Request-1 from Lesson 3.1.1.

3.1.2.2 Evaluate Permit-Policy against Request-1

Recall from Lesson 3.1.1 that the **<Decision>** should be "Permit". Confirm that this is the case.

3.1.2.3 Compare Permit-Policy with Permit-Policy-2

See if you notice the subtle difference. The closing tag of the **<AttributeValue>** element in Permit-Policy-2 on Line 16 does not come immediately after the value "Top Secret". The **MatchId** of the **<SubjectMatch>** (Line 15) is "string-equal"; during evaluation, this function will take into account the extra spaces after the value "Top Secret".

3.1.2.4 Evaluate Permit-Policy-2 against Request-1

Execute SimplePDP with Permit-Policy-2.xml and Request-1.xml, and output the results to Request-1_Permit-Policy-2_Result.xml. Inspect Request-1_Permit-Policy-2_Response.xml. Confirm that the **<Decision>** for "Resource-1" is "NotApplicable". It is "NotApplicable" because the value "Top Secret" in the **<AttributeValue>** in Request-1 on Line 7 is not the same as "Top Secret" with extra spaces as stated in Permit-Policy-2.

[14] You should be very diligent when authoring policies to avoid this problem. Also, it may be possible to construct an XSLT stylesheet to ensure that this condition never occurs.

3.1.3 Multiple Match-Predicates per Instance, Multiple Instances per Class

Goals:

1. Understand the policy evaluation semantics for multiple *match-predicates* in an *instance*.
2. Understand the policy evaluation semantics for multiple *instances* in a *class*.

Summary:

In this Lesson, you will analyze policies that have multiple *match-predicates* in an *instance* and multiple *instances* in a *class*. You will learn the policy evaluation semantics for both scenarios; multiple *match-predicates* in an *instance* are conjunctive while multiple *instances* in a *class* are disjunctive. You will be challenged to author requests that will achieve certain results.

Steps:

3.1.3.1 Inspect Multiple-Predicate-Policy.xml

Open Multiple-Predicate-Policy.xml. Multiple-Predicate-Policy contains a **<Target>** (Lines 12 - 27) with only the **<Subjects>** *class* specified (Lines 13 – 26). It contains a single **<Rule>**, "Rule-1" (Lines 29 – 37), that has an empty **<Target>** (Line 35) and an **Effect** of "Permit".

The **<Subjects>** *class* of the policy **<Target>** contains a single **<Subject>** *instance* (Lines 14 – 25). This *instance* contains two **<SubjectMatch>** *match-predicate* elements; the first is on Lines 15 – 19, and the second is on Lines 20 – 24. The first *match-predicate* can be read: "The GFIPM Security Clearance Level Code of the Subject is 'Top Secret'." The second *match-predicate* can be read: "The Subject is a Sworn Law Enforcement Officer."

Note that the second *match-predicate* uses a **MatchId** of "boolean-equal". This Function compares two Boolean values for equality. When using the SunXACML library, literal Boolean values (i.e., "true" and "false") must be in lower case.

All *match-predicates* need to evaluate to true for the parent *instance* to match a request (see Table 17: Instance Evaluation Table in the GPPTF Guide's Appendix C for more details). In this policy, requests for which the Subject is a Sworn Law Enforcement Officer with a Top Secret Clearance will match the **<Subject>** *instance*. Since this is the only *instance* in the policy, and the policy has a single rule, then this policy should evaluate to "Permit" for Sworn Law Enforcement Officers who have a Top Secret Clearance performing any action to any resource in any environment.

3.1.3.2 Inspect Multiple-Instance-Policy.xml

Open Multiple-Instance-Policy.xml. Multiple-Instance-Policy contains a **<Target>** (Lines 12 - 29) with only the **<Subjects>** *class* specified (Lines 13 – 28). It contains a single **<Rule>**, "Rule-1" (Lines 31 – 39), that has an empty **<Target>** (Line 37) and an **Effect** of "Permit".

The **<Subjects>** *class* of the policy **<Target>** contains two **<Subject>** *instances*. The first is on Lines 14 – 20, and the second is on Lines 21 – 27. Each **<Subject>** *instance* contains a single **<SubjectMatch>** *match-predicate*.

The **<SubjectMatch>** of the first **<Subject>** (Lines 15 – 20) also exists in Multiple-Predicate-Policy. It can be read: "The GFIPM Security Clearance Level Code of the Subject is 'Top Secret'."

The **<SubjectMatch>** *match-predicate* of the second **<Subject>** *instance* (Lines 22 – 26) also exists in Multiple-Predicate-Policy. It can be read: "The Subject is a Sworn Law Enforcement Officer."

For a *class* to match a request, at least one of its *instances* must match the request (see Table 18: Class Evaluation Table in the GPPTF Guide's Appendix C for more details). For this policy, the **<Subjects>** *class* will match requests for which the Subject either has a Top Secret Clearance, or is a Sworn Law Enforcement Officer, or both. Since the **<Subjects>** *class* is the only *class* specified, and there is only one rule, this policy will evaluate to "Permit" for requests that its **<Subjects>** *class* matches.

3.1.3.3 Compare Multiple-Predicate-Policy to Multiple-Instance-Policy

These policies both include the same *match-predicates*. However, since Multiple-Predicate-Policy organizes the *match-predicates* within the same *instance*, and Multiple-Instance-Policy organizes the *match-predicates* in separate *instances*, the semantics of these two policies are different (as described in Steps 0 and 0). Multiple-Predicate-Policy is more restrictive since both *match-predicates* must evaluate to true for that policy to be applicable to a request. Also, Multiple-Instance-Policy will be applicable to every request to which Multiple-Predicate-Policy is applicable.

3.1.3.4 Challenge: Create Request-1

Create a new XML file called "Request-1.xml". In this file, author a request that will be applicable to Multiple-Predicate-Policy. Because of how the two policies are written, this request should also be applicable to Multiple-Instance-Policy. The request should include Subject attributes, and a "resource-id" Resource attribute. For the "resource-id" attribute, use a value of "Resource-1". You can leave the Action and Environment sections empty.

A solution to this Challenge is in Request-1-Solution.xml.

3.1.3.5 Evaluate Multiple-Predicate-Policy against your Request-1

Confirm that the **<Decision>** for “Resource-1” is “Permit”.

3.1.3.6 Evaluate Multiple-Instance-Policy against your Request-1

Confirm that the **<Decision>** for “Resource-1” is “Permit”.

3.1.3.7 Challenge: Create Request-2

Create a new XML file called “Request-2.xml”. In this file, author a request that will not be applicable to Multiple-Predicate-Policy, but will be applicable to Multiple-Instance-Policy. The request should include Subject attributes, and a “resource-id” Resource attribute. For the “resource-id” attribute, use a value of “Resource-1”. You can leave the Action and Environment sections empty.

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

3.1.3.8 Evaluate Multiple-Predicate-Policy against your Request-2

Confirm that the **<Decision>** for “Resource-1” is “NotApplicable”.

3.1.3.9 Evaluate Multiple-Instance-Policy against your Request-2

Confirm that the **<Decision>** for “Resource-1” is “Permit”.

3.1.4 Referencing Resource Content

Goals:

1. Understand how to use the **<AttributeSelector>** element.
2. Understand how a Policy can use the content of resources in the evaluation process.

Summary:

This Lesson introduces the use of the **<AttributeSelector>** element. You will be asked to inspect and analyze policies using this element, and make comparisons with policies that only use *attribute-designators*. You will confirm the analysis by evaluating the policies against requests. Finally, you will be challenged to edit a policy and a request to achieve specific results.

Steps:

3.1.4.1 Inspect Permit-Policy.xml and Request-1.xml

Confirm that this Permit-Policy and Request-1 are the same as the Permit-Policy and Request-1 from Lesson 3.1.1.

3.1.4.2 Evaluate Permit-Policy against Request-1

Recall from Lesson 3.1.1 that the **<Decision>** should be "Permit". Confirm that this is the case.

3.1.4.3 Inspect Selector-Policy.xml

Selector-Policy is semantically the same as Permit-Policy, with two significant syntactical differences. Recall from Lesson 3.1.1 that a *match-predicate* consist of three parts:

- A **MatchId**
- An **<AttributeValue>**
- An *attribute-reference* which can be an *attribute-designator* or an **<AttributeSelector>**
    - The name of *attribute-designator* elements are dependent on which *class* the *attribute-designator* is in. **<Subjects>** contain **<SubjectAttributeDesignator>** elements and so on.

The **<SubjectMatch>** *match-predicate* of Selector-Policy uses an **<AttributeSelector>** *attribute-reference* (Lines 18 – 19), while the **<SubjectMatch>** of Permit-Policy uses an *attribute-designator* (**<SubjectAttributeDesignator>**) *attribute-reference* (Lines 17 – 18). The particular **<AttributeSelector>** in Selector-Policy causes the exact same semantic effect as the **<SubjectAttributeDesignator>** in Permit-Policy: it causes the PDP, when evaluating the policy against a request, to retrieve the values of all **<AttributeValue>** elements (as a **bag** of values) of the GFIPM Security Clearance Level Code Subject attribute of the request.

An **<AttributeSelector>** contains a **DataType** and a **RequestContextPath**. The value of a **RequestContextPath** must be an XPath expression into the request context[15]. The PDP will retrieve the set of nodes[16] referenced by the **RequestContextPath** as a **bag** of values. If no nodes are found, then the PDP returns an empty **bag**[17].

Beware that XACML defines one XML namespace for policies and a separate namespace for the XACML context. Since a **RequestContextPath** is an XPath expressions into the request context, any policy that uses an **<AttributeSelector>** must declare the XACML context namespace. This is done in Selector-Policy on Line 4; a prefix of "ctx" is used to represent the context namespace. On Line 19, the "ctx" prefix is used in the XPath expression.

3.1.4.4 Evaluate Selector-Policy against Request-1

Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.4.5 Inspect ArrestRecord.xsd

This file is in the "$POLICY_GUIDE/arrest_record_simple/" directory. It contains an XML Schema[18] for an **<ArrestRecord>** element. We will use this schema to represent a set of Arrest Records for which we want to protect access.

The schema defines an **<ArrestRecord>** element that contains seven sub-elements:

- **<Id>** - the identifier of the record.
- **<SubjectId>** - the identifier of the individual who was arrested.
- **<Jurisdiction>** - the jurisdiction in which the arrest occurred.
- **<Date>** - the date at which the arrest occurred.
- **<ArrestingOfficerId>** - the identifier of the arresting officer.
- **<ArrestingOfficerAgencyName>** - the name of the agency that employs the arresting officer.
- **<ArrestingOfficerEmailAddress>** - the email address of the arresting officer.

An arrest record articulates that an officer arrested some individual on a particular date within a particular jurisdiction. Valid values for jurisdiction are defined by the GFIPM Jurisdiction Code Set[19].

[15] The XACML "context" is the XML structures of requests and responses.
[16] A "node" is a term used in the context of XPath that means a part of an XML document.
[17] The optional **MustBePresent** property of *attribute-references* changes this behavior.
[18] The IEPD schema used here is technically not a genuine IEPD; however, the schema is NIEM IEPD-conformant and provides a close approximation of a genuine IEPD.
[19] The GFIPM Jurisdiction code set is available at http://gfipm.net/standards/metadata/2.0/codesets/GFIPMJurisdictionCode.html.

3.1.4.6 Inspect Record-1.xml

Confirm that this XML document conforms to ArrestRecord.xsd[20]. Record-1 states that Officer-1 arrested Subject-1 in Georgia on Valentine's Day 2012.

3.1.4.7 Inspect Content-Request-1.xml

This request shows an example of how XML content can be included in a request. The **<ResourceContent>** element (Lines 8 – 16) contains the content of the Record-1 Arrest Record (Lines 9 – 15). Notice the declaration of the Arrest Record namespace on Line 10 and the use of the "ar" prefix throughout the content of the Arrest Record.

Policies must use an **<AttributeSelector>** to retrieve values from a **<ResourceContent>** element in a request.

3.1.4.8 Inspect Content-Policy-1.xml

This policy will evaluate to "Permit" for requests that contain an Arrest Record with a Jurisdiction value of "GA".

The **<Target>** (Lines 16 – 27) contains a single **<ResourceMatch>** *match-predicate* (Lines 19 – 24) that uses an **<AttributeSelector>** (Lines 21 – 23). The **RequestContextPath** (Line 23) expression points to the value of the **<Jurisdiction>** element in an **<ArrestRecord>**.

Notice the use of the "ar" namespace prefix in the XPath expression and the declaration of the Arrest Record namespace on Line 5. When using the SunXACML library, XPath expressions in **RequestContextPath** XML-attributes must be XML namespace qualified.

This policy contains a single "Permit" **<Rule>** that has an empty **<Target>**.

3.1.4.9 Evaluate Content-Policy-1 against Content-Request-1

First, let's manually determine what the result should be. If the single **<ResourceMatch>** of the policy evaluates to true, then the policy should evaluate to "Permit", otherwise the policy should evaluate to "NotApplicable".

The **<ResourceMatch>** will be true for requests that include an **<ArrestRecord>** that has a **<Jurisdiction>** value of "GA". Content-Request-1 has such an **<ArrestRecord>**, therefore Content-Policy-1 should evaluate to "Permit".

[20] This can be done by using an XML Schema validator to validate Record-1.xml against ArrestRecord.xsd.

Now, execute SimplePDP with Content-Request-1.xml and Content-Policy-1.xml, and output the results to Content-Request-1_Content-Policy-1_Response.xml. Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.4.10 Challenge: Add match-predicates to Content-Policy-1

Create a copy of Content-Policy-1 and call the new file: "Content-Policy-2.xml". Open Content-Policy-2.xml. On Line 5, change the end of the **PolicyId** to read "Content-Policy-2".

In the existing **<Resource>** *instance* (Lines 18 – 25), create a new **<ResourceMatch>** that articulates this predicate: "the Subject Id of the Arrest Record equals 'Subject-1'."

Notice that the subject of the Arrest Record is handled in the **<Resources>** *class* because it is a part of the resource content and is not the subject of the request.

Create the **<Subjects>** *class* (currently non-existent) in the **<Target>** of the policy, and include one **<Subject>** *instance*. In this **<Subject>**, create a **<SubjectMatch>** that articulates this predicate: "The request Subject is a Sworn Law Enforcement Officer." Use the GFIPM Sworn Law Enforcement Officer Indicator attribute identifier[21].

A solution to this Challenge is in Content-Policy-2-Solution.xml.

3.1.4.11 Evaluate Content-Policy-2 against Content-Request-1

Evaluate your Content-Policy-2 against Content-Request-1. Confirm that the **<Decision>** for "Resource-1" is "NotApplicable". This should be because the **<SubjectMatch>** you created in Content-Policy-2 should evaluate to false; Content-Request-1 is silent on Subject attributes. Recall that all four *classes* must match a request in order for the parent **<Target>** to match the request.

3.1.4.12 Challenge: Create a request that is applicable to Content-Policy-2

Create a copy of Content-Request-1 and call the new file: "Content-Request-2". Edit Content-Request-2 to make it applicable to Content-Policy-2.

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

3.1.4.13 Evaluate Content-Policy-2 against Content-Request-2

[21] Details on the GFIPM Sworn Law Enforcement Officer Indicator attribute are at http://gfipm.net/standards/metadata/2.0/user/SwornLawEnforcementOfficerIndicator.html.

Evaluate your Content-Policy-2 against Content-Request-2. Confirm that the **<Decision>** for "Resource-1" is "Permit".

3.1.5 Rule Conditions

Goals:

1. Understand the need for rule conditions.
2. Understand how rule conditions affect rule evaluation.
3. Understand how to properly author rule conditions.

Summary:

This Lesson introduces rule conditions. Through inspecting, analyzing, and evaluating sample policies, you are led to understand the need for and semantics of conditions. You will be challenged to author a rule that contains a condition.

Steps:

3.1.5.1 Inspect Condition-Policy-1.xml

This policy has an empty **<Target>** and a single **<Rule>**, “Rule-1”, whose **Effect** is “Deny”. Rule-1 has an empty **<Target>** and a **<Condition>** (Lines 25 – 33).

A **<Condition>** is a type of *predicate* that is more flexible than a *match-predicate*. *Match-predicates* have three main limitations:

1. They use a “hard-coded”, literal value (in an **<AttributeValue>**).
2. They can only involve a single attribute.
3. Only a subset of XACML functions can be used.

Examples of predicates that *match-predicates* cannot articulate are:

- The Jurisdiction of the Resource content does <u>not</u> equal “GA”.
- The Jurisdiction of the Resource content is one of “MD” or “VA”.[22]
- The Subject’s Security Clearance Level Code equals the Resource’s Security Clearance Level code.

A **<Condition>** contains a single top-level **<Apply>** element. An **<Apply>** element, via the **FunctionId** property (see Line 26), is the specification of an invocation of a XACML function. Unlike with *match-predicates*, this function can be any function that returns a Boolean value; there is no restriction on the number or types of input parameters[23]. Therefore, parameters to the function may include:

- Literal values, via an **<AttributeValue>** element
- **<AttributeSelector>** elements
- *Attribute-designator* elements

[22] This predicate could be expressed using multiple *match-predicates*, but not a single one.
[23] Recall that *match-predicates* can only use functions that return a Boolean value and takes in two primitive values as parameters.

- Other function invocations, via other **<Apply>** elements
- “Function pointers”, via **<Function>** elements[24]

If a **<Condition>** exists in a **<Rule>**, then that **<Condition>** must evaluate to true for the **<Rule>** to be applicable to a request (see Table 20: Rule Evaluation Table in the GPPTF Guide’s Appendix C for more details).

The **<Condition>** of Rule-1 in Condition-Policy-1 uses the XACML “not” function as its top-level function call. This function takes in a single Boolean parameter and returns the opposite of that parameter (i.e., true becomes false, and false becomes true). The parameter to the “not” function is another **<Apply>** element (Line 27) specifying a call to the “string-is-in” function.

The “string-is-in” function takes in two parameters. The first must be a primitive string value. Line 28 specifies an **<AttributeValue>** with a literal value of “GA” as the first parameter. The second parameter to “string-is-in” must be a **bag** of string values. Lines 29 – 30 specify an **<AttributeSelector>**, which results in a **bag** of values, as the second parameter. The “string-is-in” function returns true if the first parameter is equal to at least one of the values of the second parameter.

This **<Condition>** *predicate* can be read: “the Arrest Record does not contain a Jurisdiction value of ‘GA’.” When this rule condition is true, the rule will evaluate to its Effect: “Deny”.

3.1.5.2 Inspect Request-1.xml

This is the same as Content-Request-1 of Lesson 3.1.4, except that the Jurisdiction of the Arrest Record is “FL” instead of “GA”.

3.1.5.3 Evaluate Condition-Policy-1 against Request-1

The **<Decision>** for “Resource-1” should be “Deny” since the Jurisdiction of the Arrest Record in the Resource content is not “GA” (it is “FL”). Confirm that this is the case.

3.1.5.4 Inspect Condition-Policy-2.xml

This policy, like Condition-Policy-1, has an empty **<Target>** and a single **<Rule>**, “Rule-1”, with an empty **<Target>** and a **<Condition>**.

The **<Condition>** (Lines 25 – 33) uses the “any-of-any” XACML function at its top-level. This function takes in three parameters. The first parameter must be a **<Function>** element

[24] The difference between the <**Apply**> element and the <**Function**> element should become apparent through the examples provided in this Lesson.

specifying a function that returns a Boolean and takes in two primitive values. The second and third parameters must be **bags** of values, and the **DataType** values of those **bags** must match the expected **DataType** values of the **<Function>** element. The "any-of-any" function applies the function specified by the first parameter between each value of the second parameter and each value of the third parameter. The "any-of-any" function returns true if at least one of the **<Function>** invocations returns true. Otherwise, the "any-of-any" function returns false.

The **<Condition>** of Rule-1 will evaluate to true if the Employment Jurisdiction of the request Subject matches the Jurisdiction of the Arrest Record.

3.1.5.5 Inspect Request-2.xml

The structure of Request-2 is similar to Request-1 of this Lesson, except that Request-2 contains a GFIPM Employment Jurisdiction Subject attribute. Also note that the values of the requested Resource Arrest Record have been changed.

3.1.5.6 Evaluate Condition-Policy-2 against Request-2

The **<Decision>** for "Resource-2" should be "Permit" since the GFIPM Employment Jurisdiction of the Subject is the same as the Jurisdiction of the Arrest Record ("FL"). Confirm that this is the case.

3.1.5.7 Evaluate Request-3

Request-3 is similar to Request-2. The only difference is that Request-3 has a different Subject attribute. Request-3 specifies that the Subject's GFIPM Federation Id is "Officer-3".

3.1.5.8 Challenge: Create a new policy

Create a file called: "Condition-Policy-3.xml". In this file, author a policy that will permit a Subject to read Arrest Records for which the Subject was the arresting officer. In other words, the GFIPM Federation Id of the request Subject must equal the value of the **<ArrestingOfficerId>** element in the Arrest Record, and the request Subject must be performing the "read" Action.

A solution to this Challenge is in Condition-Policy-3-Solution.xml.

3.1.5.9 Evaluate Condition-Policy-3 against Request-3

Confirm that the **<Decision>** for "Resource-3" is "Permit".

3.1.5.10 Inspect Condition-Policy-4.xml

This policy has an empty **<Target>** and a single **<Rule>** with an empty **<Target>** and one **<Condition>**. The **<Condition>** expresses the *predicate*: “the current date is less than the date of the accessed record plus sixty months (five years).” Note that a simpler way to word this *predicate* is: “the accessed record is less than sixty months (five years) old.” However, this simpler wording is not in a form that’s directly implementable in XACML.

The *attribute-expression* “the date on the accessed record plus sixty months” expresses a manipulation on the “record date” attribute; that attribute is manipulated by adding 60 months.

The top-level Function of the **<Condition>** is “date-less-than”, which takes in two parameters of type “date” and returns true if the first parameter is an earlier date than the second parameter. If the first parameter equals the second parameter or if the first parameter is a later date than the second parameter, then “date-less-than” returns false.

The first parameter of “date-less-than” (Lines 28 – 32) is effectively the date at which the request was constructed by the PEP. The XACML “current-date” Environment attribute represents this date[25]. An *attribute-designator* is used (see Lines 29 – 31) which provides a **bag** of values, but the “date-less-than” function requires a single primitive value, not a **bag**. Therefore, the “date-one-and-only” function (Line 28) is used. This function returns the single date primitive value from a **bag** of date values or throws an error if there is more than one value in the **bag**.

The second parameter of “date-less-than” (Lines 33 – 41) expresses the “date on the accessed record plus sixty months” *attribute-expression*. The “date-add-yearMonthDuration” function (Line 34) returns the result of adding a duration of years and months (in this case 60 months; see Lines 39 - 40) to a date value (in this case the date on the accessed record; see Lines 35 – 38).

3.1.5.11 Evaluate Condition-Policy-4 against Request-4 and Request-5

Request-4 is similar to Request-1. The main difference is that Request-4 seeks access to an Arrest Record from Valentine’s Day 2007 and includes a value for the XACML current-date Environment attribute. Recall that this attribute represents the date at which the XACML request was created and is used in the **<Condition>** in Condition-Policy-4. Request-4 expresses a XACML request that was constructed by the PEP on Valentine’s Day 2012.

[25] XACML request construction is covered in Lesson 3.3.3.2.

The value of the current-date Environment attribute is exactly five years later than the date of the record. Therefore, Request-4 should cause Condition-Policy-4 to evaluate to “NotApplicable”. Evaluate Condition-Policy-4 against Request-4 and confirm this result.

Now, inspect Request-5.xml. Request-5 is the same as Request-4 except that the current-date attribute of Request-5 has the value of “2012-02-13” which is just one day less than five years later than the date of the record. Request-5 should therefore cause Condition-Policy-4 to evaluate to “Permit”. Evaluate Condition-Policy-4 against the Request-5 and confirm the result.

3.1.6 Aggregating Multiple Rules

Goals:

1. Understand how to aggregate multiple rules into a single policy.
2. Understand the potential for conflicts.
3. Understand how rule combining algorithms are used.

Summary:

In this Lesson, you will inspect, analyze, manipulate, and evaluate policies that have multiple rules. You will learn about conflicts among rules and how rule-combining algorithms resolve those conflicts. Also, you will be challenged with authoring a policy that expresses a source policy with multiple rules.

Steps:

3.1.6.1 Inspect Policy-1.xml

This policy has an empty **<Target>** and contains two rules. The first rule (Lines 16 – 35), "Rule-1", has an **Effect** of "Permit". The second rule (Lines 37 – 65), "Rule-2", has an **Effect** of "Deny". Rule-1 can be read: "Officers can perform any Action in any Environment on Arrest Records for which they are the arresting officer." Rule-2 can be read: "Arrest Records in the 'MD' Jurisdiction cannot be deleted by any Subject in any Environment."

The two rules have conflicting **Effect** values. During evaluation against a request, if both rules are applicable to the request, the policy will evaluate to "Deny" due to its rule-combining algorithm.

The rule-combining algorithm of the policy is "deny-overrides" (see Line 6). With "deny-overrides", if any rule evaluates to "Deny", then the policy will evaluate to "Deny". If no rule evaluates to "Deny", but at least one rule evaluates to "Permit", then the policy will evaluate to "Permit". Otherwise, the policy will evaluate to "NotApplicable".

Along with "deny-overrides", main rule-combining algorithms available in XACML are "permit-overrides" and "first-applicable"[26]. The "permit-overrides" algorithm can be considered the inverse of "deny-overrides": "Permit" decisions take precedence over "Deny" decisions. With the "first-applicable" algorithm, the rules are evaluated in the order as they appear in the policy; the policy evaluates to the **Effect** of the first rule that is applicable to the request, or "NotApplicable" if no rules are applicable.[27]

[26] The complete list and semantic definitions of all standard rule combining algorithms is in Appendix C of the XACML 2.0 Specification.
[27] These are simplified descriptions of the semantics of "deny-overrides", "permit-overrides", and "first-applicable". These algorithms also handle cases where a rule evaluates to "Indeterminate".

3.1.6.2 Inspect Request-1.xml

Request-1 is the articulation of a request by Officer-1 to delete Resource-1 which is an Arrest Record. Officer-1 is the arresting officer and the arrest Jurisdiction is “VA”.

3.1.6.3 Evaluate Policy-1 against Request-1

First, let’s manually determine what the result should be. Since the policy uses the “deny-overrides” combining algorithm, we should check Rule-2 (the “Deny” rule) first. Rule-2 is not applicable to the request since the Jurisdiction in the request is “VA” and not “MD”.

Now, let’s consider Rule-1. Rule-1 is applicable to the request since Officer-1 is attempting access on a record of which Officer-1 is the arresting officer. Therefore, the policy should evaluate to “Permit”.

Confirm that the **<Decision>** of Resource-1 is “Permit”.

3.1.6.4 Inspect Request-2.xml

Request-2 is the articulation of a request by Officer-2 to delete Resource-2, which is an Arrest Record. Officer-2 is the arresting officer and the Jurisdiction is “MD”.

3.1.6.5 Evaluate Policy-1 against Request-2

First, let’s manually determine what the result should be. We’ll check Rule-2 first. Rule-2 should be applicable to the request since the Jurisdiction in the request is “MD”. Since a “Deny” rule is applicable, and since the rule-combining algorithm is “deny-overrides”, there is no need to check Rule-1. The policy should evaluate to “Deny”.

Confirm that the **<Decision>** of Resource-2 is “Deny”.

3.1.6.6 Inspect Request-3.xml

Request-3 is the articulation of a request by Officer-3 to read Resource-3, which is an Arrest Record. Officer-4 is the arresting officer and the Jurisdiction is “MD”.

3.1.6.7 Evaluate Policy-1 against Request-3

First, let’s manually determine what the result should be. We’ll check Rule-2 first. Rule-2 should not be applicable to Request-3 since Rule-2 applies to the delete Action and Request-3 seeks a read Action.

Now, let’s consider Rule-1. Rule-1 should not be applicable to the request since the request Subject, Officer-3, does not match the arrest record’s OfficerID, Officer-4. Therefore, the policy should evaluate to “NotApplicable”.

Confirm that the **<Decision>** for Resource-3 is “NotApplicable”.

3.1.6.8 Challenge: Create a new policy with multiple rules

The **<Description>** of Policy-1 (Lines 8 – 12) states: “Officers can perform any Action on Arrest Records for which they are the arresting officer. However, under no circumstances can records in the ‘MD’ Jurisdiction be deleted.” Your Challenge is to create a new policy with a slightly different articulation: “Officers can perform any Action on Arrest Records for which they are the arresting officer. However, under no circumstances can records in the ‘MD’ Jurisdiction be deleted, except by holders of a Top Secret Clearance. Holders of a Top Secret Clearance can perform any Action on any Record in any Environment.”

Save your new policy in a file called “Policy-2.xml”. There are several possible solutions to this Challenge. One solution is provided in Policy-2-Solution.xml.

3.1.6.9 Inspect Policy-2-Solution.xml

Let’s compare Policy-2-Solution to Policy-1. The rule-combining algorithm was changed to “first-applicable”. A new Rule-1 provides total access to request Subjects with a Top Secret Security Clearance Level Code. Rule-2 stayed the same. Rule-1 from Policy-1 became Rule-3 in Policy-2-Solution.

The Description of Rule-1 of Policy-2-Solution (Lines 20 – 23) states: “Holders of a Top Secret Clearance can perform any Action on any Record in any Environment.” When evaluating this policy against a request, the PDP will first evaluate Rule-1. If Rule-1 applies to a request, then the “first-applicable” rule-combining algorithm tells the PDP to proceed no further and to apply the Effect of Rule-1: “Permit”. If Rule-1 is not applicable to the request, then the PDP will evaluate Rule-2. If Rule-2 is not applicable to the request, then the PDP will evaluate Rule-3. If Rule-3 is not applicable, then the policy will evaluate to “NotApplicable”.

3.1.6.10 Inspect Request-4.xml

Request-4 can be articulated as follows: “Officer-4, who has a Top Secret Clearance, is attempting to delete Resource-4, which is an Arrest Record. Officer-4 is the arresting officer and the Jurisdiction is ‘MD’.”

3.1.6.11 Evaluate Policy-2 against Request-4

Use Request-4 to test your Policy-2 (and Policy-2-Solution). Confirm that the **<Decision>** for Resource-4 evaluates to “Permit”, since the request matches Rule-1.

3.1.6.12 Inspect Request-5.xml

Request-5 can be articulated as follows: “Officer-5, who has a “Secret” Clearance, is attempting to delete Resource-5, which is an Arrest Record. Officer-5 is the arresting officer and the Jurisdiction is ‘MD’.”

3.1.6.13 Evaluate Policy-2 against Request-5

Use Request-5 to test your Policy-2 (and Policy-2-Solution). Rule-1 should not be applicable to the request since Officer-5 does not have a “Top Secret” Clearance. Rule-2 should be applicable since the request is an attempt to delete an Arrest Record in the “MD” Jurisdiction. Therefore, Policy-2 should evaluate to “Deny”.

Confirm that the **<Decision>** for Resource-5 is “Deny”.

3.1.6.14 Inspect Request-6.xml

Request-6 can be articulated as follows: “Officer-6, who has a Secret Clearance, is attempting to delete Resource-6, which is an Arrest Record. Officer-6 is the arresting officer and the Jurisdiction is ‘VA’.”

3.1.6.15 Evaluate Policy-2 against Request 6

Use Request-6 to test your Policy-2 (and Policy-2-Solution). Rule-1 should not be applicable to the request since Officer-6 does not have a Top Secret Clearance. Rule-2 should not be applicable since the Jurisdiction of the record is not “MD”. Rule-3 should be applicable since Officer-6 is both the Subject of the request and the arresting officer on the record. Therefore, Policy-2 should evaluate to “Permit”.

Confirm that the **<Decision>** for Resource-6 is “Permit”.

3.1.7 Aggregating Multiple Policies

Goals:

1. Understand how to aggregate multiple **<Policy>** elements in a **<PolicySet>** element.

Summary:

In this Lesson, you will inspect, analyze, and evaluate a policy consisting of a top-level **<PolicySet>** element and multiple **<Policy>** sub-elements. We provide the context of a local implementing agency needing to aggregate policies from multiple levels of authority, illustrating how policy-combining algorithms resolve conflicts among policies in a policy set.

Steps:

3.1.7.1 Inspect PolicySet-1.xml

PolicySet-1 is a **<PolicySet>**. It has a **PolicySetId** identifier (Line 5), and a **PolicyCombiningAlgorithm** of "permit-overrides" (Line 6). Policy-combining algorithms work in a similar manner to rule-combining algorithms. PolicySet-1 specifies a **<Target>** (Lines 16 – 26) and two **<Policy>** elements (the first on Lines 28 – 103 and the second on Lines 105 – 160). The first **<Policy>** represents a federation-level policy (of the "ExampleFederation") and the second represents a policy that's local to the agency that is implementing this **<PolicySet>** ("Agency-A").

This **<PolicySet>** is concerned with access to the criminal history records of Agency-A. Arrest Records constitute the entirety of criminal history data of Agency-A. Accordingly, the **<Target>** of the **<PolicySet>** specifies that the GFIPM Criminal History Data Indicator of the Resource must be true.

The first **<Policy>**, Federation-Policy-1, is the federation-level policy. It can be articulated as: "A federated user can read criminal history data (Arrest Records) if that user meets the following criteria: they are a sworn law enforcement officer, they possess the criminal history data agency home search privilege, and they have legal jurisdiction in the jurisdiction of the record."[28] The Subject *match-predicate* on Lines 53 – 57 uses the "string-regexp-match" XACML function to determine if the Subject is a member of ExampleFederation by checking the GFIPM Federation Id attribute[29].

The second **<Policy>**, Local-Policy-1, is the local-level policy. It can be articulated as: "All sworn law enforcement officers of Agency-A who are authorized to search criminal history data are allowed to read any criminal history record."

[28] Note that Federation-Policy-1 duplicates the **<ResourceMatch>** that is in the **<Target>** of the **<PolicySet>** because Federation-Policy-1 needs to be a complete policy in and of itself.
[29] See http://gfipm.net/standards/metadata/2.0/user/FederationId.html.

PolicySet-1 uses the “permit-overrides” policy combining algorithm, therefore “Permit” decisions take precedence over “Deny” decisions. However, since PolicySet-1 does not contain any “Deny” rules, it will never evaluate to “Deny”. It can only evaluate to “Permit” or “NotApplicable”[30].

3.1.7.2 Evaluate PolicySet-1 against Request-1

The **<Target>** of PolicySet-1 will match Request-1 since the request is for criminal history data. Therefore, Federation-Policy-1 will be evaluated.

Federation-Policy-1 will not be applicable to the request since the jurisdiction of the request Subject (“VA”) does not match the jurisdiction of the record (“GA”). Therefore, Local-Policy-1 will be evaluated.

Local-Policy-1 will be applicable to the request since the Subject is a member of Agency-A (see Lines 6 – 8 of Request-1), is a sworn law enforcement officer, and is authorized to search criminal history data records (see Lines 24 – 27 of Request-1). Therefore, PolicySet-1 should evaluate to “Permit”.

Confirm that the **<Decision>** for Resource-1 is “Permit”.

3.1.7.3 Evaluate PolicySet-1 against Request-2

The **<Target>** of PolicySet-1 will match Request-1 since the request is for criminal history data. Therefore, Federation-Policy-1 will be evaluated.

Federation-Policy-1 will not be applicable to the request since the Subject does not have the criminal history data home agency search privilege (see Lines 20 – 23 of Request-2). Therefore, Local-Policy-1 will be evaluated.

Local-Policy-1 will not be applicable to the request since the Subject is not a member of Agency-A (see Lines 6 – 8 of Request-2). Therefore, PolicySet-1 should evaluate to “NotApplicable”.

Confirm that the **<Decision>** for Resource-2 is “NotApplicable”.

3.1.7.4 Evaluate PolicySet-1 against Request-3

[30] Theoretically, PolicySet-1 can also evaluate to “Indeterminate”, however, we have designed the policy and requests to avoid this result.

The **<Target>** of PolicySet-1 will match Request-1 since the request is for criminal history data. Therefore, Federation-Policy-1 will be evaluated.

Federation-Policy-1 will be applicable to the request (you should be able to determine why). Therefore Federation-Policy-1 should evaluate to "Permit". Given the policy-combining algorithm "permit-overrides", there will be no need to evaluate Local-Policy-1, and PolicySet-1 should evaluate to "Permit".

Confirm that the **<Decision>** for Resource-3 is "Permit".

3.1.8 Obligations

Goals:

1. Understand how obligations are expressed in XACML.
2. Understand that obligation semantics are outside of the scope of XACML.

Summary:

This Lesson introduces obligations. You will analyze and evaluate a policy containing multiple obligations. There is a discussion on the design and handling of obligations.

Steps:

3.1.8.1 Inspect Policy-1.xml

Policy-1 is based on the Policy-1 from Lesson 3.1.6, with the addition of obligations using the **<Obligations>** element (Lines 67 – 91).

In the abstract sense, an obligation is an action that must be performed in conjunction with policy enforcement. This policy contains three obligations: the first is on Line 69, the second is on Line 71, and the third is on Lines 73 – 89.

An obligation in XACML (an **<Obligation>** element) has a **FulfillOn** property, an **ObligationId** property, and a set of zero or more **<AttributeAssignment>** elements. The **FulfillOn** property specifies the decision on which the obligation must be fulfilled; the value of this property can be “Permit” or “Deny”. The **ObligationId** is the identifier of the obligation. An **<AttributeAssignment>** is an argument[31] of the obligation. An **<AttributeAssignment>** contains a **DataType**, an identifier as an **AttributeId**, and a literal value.

The first obligation in Policy-1, LogValidAccess, is to be fulfilled on “Permit”; the PDP will include this obligation in the result when the decision is “Permit”. The LogInvalidAccess obligation is to be fulfilled on “Deny”; the PDP will include this obligation in the result when the decision is “Deny”. These obligations instruct the PEP to write data about the access to an audit log. The PEP must recognize and know how to handle these obligations. If a PEP does not understand or cannot fulfill an obligation, then the PEP must not allow access. For these example obligations in particular, we assume that the PEP (or its Obligation Handler components) will know how to retrieve the appropriate data to write to the log.

The third obligation, NotifyDataOwner, instructs the PEP to send a notification to the owner of the accessed record. In our scenario, the owner of an Arrest Record is the arresting officer. This obligation has three **<AttributeAssignment>** elements. The first is DataOwnerId and the value is

[31] An “argument” is data that is needed for the proper processing of the obligation.

actually an **<AttributeSelector>** containing an XPath expression selecting the value of the **<OfficerId>** element of the Arrest Record being accessed. Notice that the angled brackets are URL encoded (i.e., "<" becomes "&lt;" and ">" becomes "&gt;"); the PDP will decode these in the result. We assume that the PEP/Obligation Handler will process this **<AttributeSelector>** to retrieve the value for the DataOwnerId argument. We also assume that the PEP/Obligation Handler will be able to retrieve the appropriate address for the arresting officer.

The second argument is DataRequestorId and the value is the URL encoded **<SubjectAttributeDesignator>** that will retrieve the appropriate value.

The third argument is Message; this is the actual text that should be sent to the arresting officer. We assume that the PEP will replace "[DataRequestorId]" with the result of processing the second argument.

How obligations are designed will affect how the PEP (or its Obligation Handler components) will be designed. Design options include identifier naming conventions, whether to include arguments, and which arguments to include. We developed a particular design style for this tutorial, but there are currently no standard obligation design patterns available. As stated in the GPPTF Guide's Section (Step 6, Requirement C), the Global Federated Identity and Technical Privacy Task Team is currently developing a standardized syntax and processing model for various types of policy obligations.

Since no XACML obligations are returned on the "NotApplicable" decision, care must be taken in designing policies to avoid this decision where appropriate so that all necessary obligations are properly returned to the PEP.

3.1.8.2 Evaluate Policy-1 against Request-1

This is the same Request-1 from Lesson 3.1.6; therefore we know that the decision will be "Permit". Use SimplePDP to evaluate Policy-1 against Request-1, output the result to "Request-1_Policy-1_Response.xml", and open the result. Confirm that the **<Decision>** for Resource-1 is "Permit".

Notice the **<Obligations>** element on Lines 7 – 22. This element contains the two obligations that were specified to be fulfilled on "Permit". The PDP simply copies the appropriate obligations into the result (and decodes any URL-encoded values).

3.1.8.3 Evaluate Policy-1 against Request-2

This is the same Request-2 from Lesson 3.1.6; therefore we know that the decision will be "Deny". Use SimplePDP to evaluate Policy-1 against Request-2, output the result to "Request-

2_Policy-1_Result.xml”, and open the result. Confirm that the **<Decision>** for Resource-2 is “Deny”.

This result includes the obligation that was specified to be fulfilled on “Deny” (see Lines 7 – 10).

3.1.8.4 Evaluate Policy-1 against Request-3

This is the same Request-3 from Lesson 3.1.6; therefore we know that the decision will be “NotApplicable”. Use SimplePDP to evaluate Policy-1 against Request-3, output the result to “Request-3_Policy-1_Result.xml”, and open the result. Confirm that the **<Decision>** for Resource-3 is “NotApplicable”. Since the decision is “NotApplicable”, no obligations were returned in the result.