Technical Lesson 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”.[1]
- 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[2]. Therefore, parameters to the function may include:

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

- Literal values, via an **<AttributeValue>** element
- **<AttributeSelector>** elements
- *Attribute-designator* elements
- Other function invocations, via other **<Apply>** elements
- “Function pointers”, via **<Function>** elements[3]

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>**.

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

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 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[4]. 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

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

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.

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.

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".