Technical Lesson 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[1] 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.

[1] An "argument" is data that is needed for the proper processing of the obligation.

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 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 "<" and ">" becomes ">"); 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 2.3.1 (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.

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".