XACML Policy Model

This document defines the ongoing work of the XACML policy model subcommittee. It keeps track of the goal and advancement status of the subcommittee work.

1 Charter

The framework will be flexible and expressive enough to support different access control policies that may need to be applied (and have been proposed or are used in different real-world scenarios).

The framework will comprise of

- a *model*: clearly describing the type of access control rules that can be expressed and their evaluation. The model needs not be formal but the definition must be unambiguous.
- *a language*: for specifying access control rules. The language gives a syntax for expressing the rules whose semantics has been defined in the model. The language will be XML-based, namely a policy will be represented as a valid XML document (each rule corresponding to a valid XML fragment). The XML schema used to validate policies and rules will contain type definitions for all entities composing the rules.

The framework will be flexible and expressive enough to accomodate different protection requirements and policies. It will be extensible, that is it will be possible to define new types of entities by extending existing ones via well defined procedures.

The language can be seen as three-layered:

1. core-layer gives the syntax of the rules
2. type-layer gives the data types allowed as entities inside rules
3. policy-layer defines the overall syntax of policies.

The result of the subcommittee could be used, at the committee level, as a starting point for providing a reference implementation.

The remainder of this document describes the progress on the work towards achieving the goals stated in the charter.

The terminology used in the document refers to the glossary reported in the document "XACML Language".

2 Elements of the access control policy

An access control policy states regulations governing access to resources, and therefore how the system should respond to requests that principals can submit. The access control policy comprises of *access rules* stating which accesses should (or should not) be allowed and, possibly, under which conditions such permissions or denials for the access apply.

We therefore start by identifying the various elements of the access rules. At this stage we characterize the different access components with respect to their format and semantics. Later we will define the precise syntax (a preliminary sketch of the syntax appears in "XACML Language").

An access rule can be seen as comprised of the following elements:

- *principal expression* identifies the (dynamic set of) principals to which the rule applies.
- *resource expression* identifies the (dynamic set of) resources to which the rule applies.
- *action expression* identifies the (dynamic set of) actions to which the rule applies.
- *environment expression* identifies system-dependent and request-dependent conditions to be satisfied for the rule to apply.
- *post-conditions* defines a set of actions that the access control system (PEP) must execute whenever a rule is applied with respect to a given access request.
- IF/ONLY IF conditional statements.....

2.1 Reserved identifiers

The expressiveness of the language will allow us to specify rules whose applicability will depend on conditions that the principal requesting access, or the resource on which access is requested, satisfy. Access rules are therefore not referred to a specific principal or a specific resource but to a set of them which satisfy given conditions. To provide expressiveness needed to make it possible the specification of such generic rules without the need of introducing variables in the language we introduce the following reserved identifiers:

- **principal**: is the principal presenting the request
- **resource**: is the resource on which access is requested
- **action**: is the action requested

2.2 Principal expression

The principal expression defines the principal, or set of principals, to which the access rule applies. It is a boolean expression evaluating SAML assertions (i.e., properties) associated with the principal requesting access.

SAML asserions can refer to any property of the principals, including *groups* to which the user making the request belongs or *roles* (privileged positions) that the user may have activated and present. Groups and role management is outside the scope of the authorization language, we assume information about active roles to be provided through SAML assertions; we assume information about group memberships to be either provided as SAML assertions or to be available at the PIP.

Each given assertion term (i.e., elementary component of a principal expression) evaluates the value of a property associated with the requestor as is of the form

$$\langle\text{SAML-assertion}\rangle\ \langle\text{comparison operator}\rangle\ \langle\text{SAML-assertion}\rangle$$
$$\text{or}$$
$$\langle\text{SAML-assertion}\rangle\ \langle\text{comparison operator}\rangle\ \langle\text{costant-value}\rangle$$

where the comparison operator is a suitable operator (including $<, \leq, >, \geq, =, \subset$) depending on the property type and the XPath language is used to refer to SAML assertions within the specification of the assertion terms.

Some examples of principal expressions are as follows:

- **principal**/login_name='bob'

  user 'bob'

- **principal**/organization='OASIS' AND **principal**/residence='North-Pole'

  principals associated with OASIS and living at the North Pole

- **principal**/organization=**principal**/login_name

  principals working for an organization owned by themselves.

Use of variables as macros (to be completed checking with Ernesto and Simon)

2.3 Resource expression

The resource expression is a boolean expression of conditional terms that evaluate properties of the resource. Properties appearing in these conditional terms can refer to the *resource content* or to *meta properties* associated with the resource. The reserved identifier **resource** can be used in the specification of generic expression applicable to a (dynamic) set of resources. Resource expressions can be also make use of the keyword **principal** for specifying conditions putting in relationships properties of the resources with that of the principals. Each conditional term in a resource expression is therefore of the form

$$\langle\text{resource-assertion}\rangle\ \langle\text{comparison operator}\rangle\ \langle\text{SAML-assertion}\rangle$$
$$\text{or}$$
$$\langle\text{resource-assertion}\rangle\ \langle\text{comparison operator}\rangle\ \langle\text{costant-value}\rangle$$

Reference to the meta-property of the resource or to its content depends on .... and can make use of functions.

Some example of resource expressions are as follows:

- **resource**/creation_date $\leq$ '01/01/01'

  all resources created before January 1, 2001.

- **resource**/owner=**principal**/login_name

  all resources owned by the principal making the requests

- **resource**/label='Top Secret'

  all resources classified as Top-Secret (note: we are not implying support for a multilevel policy here)

2.4 Action

The action component of the access control rule defines the action, or set of actions, to which the rule refers. An action is identified by a *name* and may have associated a set of *parameters*. The parameters can refer to any input/output of the process.

Some examples of action expressions are as follows:

- **action**/withdrawal_amount$\leq$500,000

- **action**/data_recipient$\in$Doctors

2.5 Environment expression

It's a boolean expressions of environmental conditions that can evaluate the system state (e.g., date and time) and request parameters (e.g., location from where the principal is connected).

2.6 If/only if conditions

A policy is composed of a set of access control rules. The usual interpretation for a set of rules specifying permissions is to grant all the accesses from which at least a rule is satisfied. The consideration of permissions only permissions, with this interpretation may result limiting in several cases. Negative access control rules (specifying denials) could be used for specifying accesses that should be denied. Introduction of negative authorizations introduces the problem of the different semantics that denials can carry which should be properly represented with different conflict resolution criteria in the model. For the time being we therefore do not consider negative authorizations. Permissions only, however, result limiting. For instance, suppose we want to say that only UScitizens can access a document. In a permission only scenario we could specify a rule stating the permissions but the semantics of the *only* (meaning no one else can access the document) is not supported, since the insertion of any additional rule can grant the access to some noncitizens. As another example suppose we want to say that access to a given document requires (beside additional conditions to be specified by the security administrator) presentation of a payment certificate (stated as a SAML assertion). In a permission only scenario we should make sure that the payment condition is included in *all* the rules that apply to the access. Beside being difficult to control, this would introduce complicated rules (intuitively the conditions would have to be repeated in AND in every rule instead of being factored out.

Looking at the real world cases, we often find access rules stated in a restrictive form rather than in the inclusive positive form just mentioned. By restrictive form we mean rules that state conditions that *must* be satisfied for an access to be granted and such that, if at least one condition is not satisfied, the access should not be granted. For instance, a rule can state that "access to document-1 can be allowed *only* to citizens". It is easy to see that such a restriction cannot be simply represented as an permission stating that citizens can be authorized. In fact, while the single authorization brings the desidered behavior, its combination with other authorizations may not, leading the *only* constraint to be not satisfied anymore.

A possible approach would be supporting two kinds of rules: *restrictions* and *authorizations.* Intuitively, restrictions are useful to specify requirements of the exclusive *only if* form stated above; while authorizations specify requirements in the traditional positive *if* form.

Restrictions specify requirements that must all be satisfied for an access to be granted. Lack to satisfy any of the requirements that apply to a given request implies the request will be denied.

Syntactically, restrictions have the form

$$\langle \textit{request-description} \rangle \ \langle \textit{conditions} \rangle$$

where:

- *request-description* is the principal, resource, action, and enviroment expressions and
- *conditions* is a boolean expression of conditions that every request to which the restriction applies must satisfy.

Authorizations specify permissions for the access. An access is granted if there is satisfaction of at least one of the permissions that apply to the given request and no restriction is violated.

Syntactically, authorizations have the form

$$\langle \textit{request-description} \rangle \ \langle \textit{conditions} \rangle$$

where *request-description* has the same meaning as before and ⟨*conditions*⟩ is a boolean expression of conditions whose satisfaction authorizes the access.

Unlike for restrictions, lack of satisfaction of a condition in an authorization simply makes the authorization inapplicable but it does not imply the access will be denied. In particular, access can be authorized if there is at least one authorization that applies to it for which the conditions are satisfied.

2.7 Things to discuss

- *purpose of access* discussed in the last concall, still to be inserted
- *dynamic conditions* conditions that cannnot be evaluated but can trigger procedures
- *post-conditions*

- *content-based filtering*. We should decide whether the resource expression can contain conditions evaluating the resource content. The complication arise from the fact that content-querying depends on the specific application/data-model/system.
- *attribute reference* (syntactical matter) in the examples we have used naming based notation to refer to parameters of an action. Should we allow (XPath permits it) positional-based notation as well?
- examples and semplification. Now the language can seem a bit too complicated. For instance, we need to say "**principal**/login_name='bob', in cases where we would have just said 'bob' in traditional systems. This is however consistent with the fact that for us a principal is not a user 'login-name' but it is characterized through assertions. However, we could find a way to simplify expressions in some cases.
- *dealing with unknown attributes* SAML assertions (as well as resource properties) are not predefined and can change. What happens if a rule has a condition on some SAML assertions which cannot be found at runitime?
- description of run-time behavior of access control