Core XACML and Term Rewriting*

Daniel J. Dougherty

March 2007

Abstract

We define a notion of "core" XACML and show how these can be represented as ground associativee-commutative term rewriting systems with strategies.

1 Core XACML

The first few sections are an introduction to XACML and a precise specification of the fragment that is the source of our translation.

Roughly speaking our notion of "core XACML" is the fragment of XACML in which the applicability of rules is determined by matching *attributes* of the subject, action, and resource in the access request with corresponding attributes referenced in the policy. What we exclude is XACML's use of arbitrary boolean conditions (evaluated in the context of the dynamic environment) in rules. Of course it would be fruitless to attempt to model this use of arbitrary constraints as pure term rewriting.

For convenience (only) below we do not treat XACML *Policy Sets*.

2 Attributes

([XACML] section 6.7) An Attribute is a pair consisting of an *AttributeId* and a multi-set of *Attribute Values*. Under the XACML semantics ([XACML] sec 7.5) an Attribute can be treated as a set of (attId, attVal) pairs. See discussion of SubjectMatch below. So for simplicity

$$\mathsf{Attribute} \quad ::= \quad (\mathsf{attId}, \ \mathsf{attVal})$$

3 Requests (contexts)

Requests ([XACML] sec 6.1) are the "input" to the policies. XACML calls these *contexts*, and takes the point of view that a request is really a context for evaluating a Target.

*WPI Computer Science Technical Report WPI-CS-TR-07-07

A request collects a bunch of attributes, into sorts we call SubjectQ, ResourceQ, ActionQ, and EnvironmentQ.

Request	::=	SubjectQ ResourceQ ActionQ EnvironmentQ
SubjectQ	::=	(attId, attVal)*
ResourceQ	::=	(attId, attVal)*
ActionQ	::=	(attId, attVal)*
EnvironmentQ	::=	(attId, attVal)*

For SubjectQ, e.g., typically have at least the Attribute which is (subject-id, name).

4 Policies, Rules, Targets

4.1 Policies

Policy	::=	Target Rule* CombiningAlgorithm
Rule	::=	RuleId Effect Target
CombiningAlgorithm	::=	"permit override" | "first-applicable" | etc
Effect	::=	"permit" | "deny"

Here's how a Policy handles a Request ([XACML] sec 7.10). Let Q be a request, presented to a Policy as above. As defined below, the given Request matches (or not) any given Target. So:

1. if Q does not match the Target at the top level of the Policy, the result of the request is not-applicable.
2. if Q does match the top level Target, each Rule is applied...
3. for each such Rule, see if the request matches the Target of the Rule. If it matches the result of the Rule is its Effect; if it does not match the result of the Rule is not-applicable.
4. The results of applying each Rule are combined using the CombiningAlgorithm of the Policy

Note that relationship between Policy target and Rule targets is subtle. The XACML specification states ([XACML] section 5.22):

> The Target element may be declared by the creator of the Policy element, or it may be computed from the Target elements of the referenced Rule elements either as an intersection or as a union.

Also, if a policy target is explicitly given then a Rule need not give one if it is to have the same target.

4.2 Targets and evaluation

Target ([XACML] section 5.5)

Target	::=	Subjects $\wedge$ Resources $\wedge$ Actions $\wedge$ Environments.
Subjects	::=	Subject $\vee$ Subject $\vee$ Subject ....
Subject	::=	SubjectMatch $\wedge$ SubjectMatch $\wedge$ SubjectMatch ...
SubjectMatch	::=	attId matchId attVal

and similarly for Resources, Actions, Environments...

We have inserted the $\wedge$ above as a mnemonic to suggest that in the matching semantics, *all* need to match, and $\vee$ is to suggest that *at least one* needs to match.

Typically matchId is an operator like =, less-than, etc, and the selector tells what attribute name from the context to focus on; the attVal is the value.

How does a Subject match a request? The XACML spec [7.5, p82] says:

> Given SubjectMatch *a* and a Request, the Request has a multiset of values for the given id. The match succeeds if the assertion made by the SubjectMatch is true of one of the values in the Request values multiset. (Note that if the Request gives the empty multiset for this id, match fails.)

So if a Subject has several SubjectMatches, matching succeeds if *each* of the attributes in the subject part matches one of the attributes in the request. Since, in our simplification matchId is just equality, this reduces to saying that the Attributes in the Subject are a subset of those in the Request.

5 XACML policies as term-rewriting systems

In the term-rewriting setting we treat a Policy as a set of rewrite-rules. In [DKKS07] we show how the effect of an XACML Combining Algorithm can be obtained by using strategies in term-rewriting systems. So it will suffice to show how term rewriting can simulate the effect of applying a single Rule to a Request.

5.1 Assumptions

1. We will assume that Policies do *not* have top-level Targets, and that the implicit target is the union of the Rule Targets (this has the effect of letting the Rules operate independently, modulo the strategy embodied in the CombiningAlgorithm).
2. We assume that each Target has only a single Subjects node, a single Resources node, a single Actions node, and a single Environments node. This assumption is without loss of generality, by the following argument.

   By distributing the (implicit) disjunction in the Subjects, Resources, etc over the (implicit) conjunction in the Subject, Resource, etc we can transform any Rule into a collection of Rules without these disjunctive aspects. Any Policy written in the standard style can be simulated in this style, and vice-versa. This is easy to see. Note that there is no need to be careful about how Rules are combined here. The actual claim is that for any Target (old style) there is a "non-disjunctive" Target with the same semantics. It follows that the semantics of Rules, and hence Polices, are unchanged with this assumption.
3. Finally, for simplicity let us assume that the matchId in a SubjectMatch is simply equality. In this case we may re-formulate the definition of SubjectMatch, ResourceMatch, etc as simply

   SubjectMatch ::= Attribute

   and similarly for ResourceMatch, ActionMatch, EnvironmentMatch.

   with the understanding that an Attribute matches a SubjectMatch just when they are identical.

5.2 The construction

We will compile XACML policies into *ground AC rewriting systems*, with strategies.

It is easiest to describe the rewrite system as a many-sorted system. Sorts include RuleId, Subject, Resource, Action, Environment, and Effect.

The signature of the rewrite system includes

- a unary constant for each Attribute a; for ease of notation we use the name a for this constant as well
- a binary operator $\wedge$, (which will be treated as associative-commutative operator)
- an operator req, of sort (Subject $\times$ Resource $\times$ Action $\times$ Environment) $\rightarrow$ Effect
- constants permit and deny of sort Effect
- for each constant d in RuleId, constants $subd, resd, actd$, and $envd$, of sorts Subject, Resource, Action, and Environment, respectively.

Translating Requests

- If a is an Attribute its translation is a
- If S is an element of SubjectQ with set of Attributes $b_1, \ldots, b_n$ then its translation is

$$(b_1 \wedge \cdots \wedge b_n)$$

- Similarly for ResourceQ, ActionQ, and EnvironmentQ.
- If Q is a Request, its translation is $req(sub, res, act, env)$ where sub etc are the translations of the Subject etc parts of Q.

Translating Rules Let R be an XACML Rule with RuleId d, Effect $e \in \{\text{permit, deny}\}$ and target t. Recall that we are assuming that the Target of the rule has a single Subjects node, a single Resources node, etc.

Suppose the Subject of the target has k SubjectMatch elements, each of which is an attribute; let $\{a_1, \ldots, a_k\}$ be the set of these attributes. Then we have the rule

$$(a_1 \wedge \cdots \wedge a_k \wedge x) \rightarrow subd$$

Note the use of the constant $subd$ which is associated to the given Rule; the right-hand side of the rule is a code for "matches the Subject part of Rule d. "

We add analogous rules for the Resource, Action, and Environment parts of the Target. Finally we add the rule

$$req(subd, resd, actd, envd) \rightarrow e$$

where e is the Effect of the rule (permit or deny). [1]

[1] this could all be coded as one rule but it seems easier to understand this way.

Justification The claim is that if Q is a request and R is a rule then Q matches the Target of R if and only if the translation of Q rewrites, under the rewrite rules derived from R, to the (constant encoding the) Effect of R.

It suffices, without loss of generality, to see that if SQ is the Subject part of a request Q and SM is the Subject part of a Target of Rule d then SM evaluates to "Match" (in the XACML sense) if and only if the translation of SQ rewrites to $subd$ under the rewrite rule capturing SM.

The translation of SQ is of the form

$$(b_1 \wedge \cdots \wedge b_n),$$

while the translation of SM is a rule

$$(a_1 \wedge \cdots \wedge a_k \wedge x) \rightarrow subd$$

Repeating the semantics of a SubjectMatch from above, we need to check that the Attributes in the Subject are a subset of those in the Request.

It is clear that AC-rewriting implements this semantics.

References

[XACML] eXtensible Access Control Markup Language (XACML) Version 2.0 Committee draft 04, 6 Dec 2004. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`

[DKKS07] D. J. Dougherty, C. Kirchner, H. Kirchner, A. S. de Oliveira. Modular Access Control via Strategic Rewriting. 12th European Symposium On Research In Computer Security (ESORICS). Lecture Notes in Computer Science 4734, pp 578–593, 2007.