DEVELOPMENT OF A NEW POLICY EVALUATION PROCEDURE FOR XACML

Jorian van Oostenbrugge
Supervisor: Fatih Turkmen

August 19, 2016

System and Network Engineering
University of Amsterdam
WHY

- Customer data more and more valuable
- Data stored in cloud
- Access control becomes critical
XACML

- eXtensible Access Control Markup Language
- XML-based language
- Also an architecture
- OASIS standard for the expression of security policies
<PolicySet>
  <Policy RuleCombiningAlg="..." >
    <Target/>
    <Rule RuleId="..." Effect="Permit">
      <Target/>
      <Condition/>
    </Rule>
  </Policy>
  <Policy RuleCombiningAlgId="...">
    ...
  </Policy>
  <Policy RuleCombiningAlg="..." >
    ...
  </Policy>
</PolicySet>
EXAMPLE XACML POLICY

```xml
<Policy
    RuleCombiningAlgId="identifier:rule-combining-algorithm:permit-overrides">
    <Target/>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule1" Effect="Permit">
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="string-equal">
                        <AttributeValue DataType="string">admin</AttributeValue>
                        <AttributeDesignator AttributeId="role" DataType="string"/>
                    </Match>
                </AllOf>
            </AnyOf>
        </Target>
        <Condition>
            ...
        </Condition>
    </Rule>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule2" Effect="Deny">
        ...
    </Rule>
</Policy>
```
XACML IN ACTION

1. Request intercepted by PEP
2. Request converted to XACML
3. PDP evaluates policy
4. If needed retrieve additional attributes
5. PDP reaches decision and forwards this to PEP
6. Request arrives at resource

(Adaptive) reordering
- Based on statistics and categorization

Decision Diagrams
- XEngine
- Matching Tree (MT) and Combining Tree (CT)
- SNE-XACML with MIDD

Source: XEngine: A Fast and Scalable XACML Policy Evaluation Engine
RESEARCH QUESTION

• Propositional encoding

• PDP
SAT & CNF

- Boolean function: $f(x_1, x_2, ..., x_n)$
- Variables, operators and parentheses: $x_1, \land, \lor, \neg, ()$
- SAT solvers
- CNF: $(p_1 \lor p_2) \land (p_3 \lor p_4) \land (p_5 \lor p_6)$
  - Conjunction of clauses
  - Disjunction of literals
ALGORITHM

- Constructing attribute domains
- Policy flattening
- SAT encoding
CONSTRUCTING ATTRIBUTE DOMAINS (1)

- Attributes
  - AttributeValue
  - AttributeDesignator
  - AttributeSelector

```
<rule Effect="Permit">
  ...
  <AttributeValue DataType="String">admin</AttributeValue>
  <AttributeDesignator AttributeId="role" DataType="String"/>
  ...
</rule>
```
$D_{\text{role}} \in \{\text{admin, manager, hr, user}\}$

$\text{admin} \in \{\text{admin, manager, hr, user}\}$
The access decisions are: Permit, Deny, and NotApplicable, respectively. The access requests are determined by the target decision space, which is created in the same way, using both the rule's constraints. The decision space is created in the same way, using both the rule's constraints. For all policy elements, the first step is to find the first available applicability condition. As these applicability constraints work in a bottom-up manner, the combined decision space is created in the same way, using both the rule's constraints. The combined decision space is created in the same way, using both the rule's constraints. In our case the first constraints we find are the ones for the rule's constraints. In that we use only boolean predicates, whilst they use formulas and are hence not bound by using only boolean predicates. We only implemented the encoding of a single policy containing policies = add result to policies. The algorithm as shown in Algorithm 2 is based on the encoding of a single policy containing policies = add result to policies. The access request to which the rule's constraints apply is Permit. The access request to which the rule's constraints apply is Permit. The access request to which the rule's constraints apply is Permit.

Algorithm 1 EnumerateVariables

Input: A map m containing the DataTypes as keys and (empty) arrays as values and a policy p

1: procedure EnumerateVars(p,m)
2: for all target elements do
3: update m with values found in the policy target
4: end for
5: for all variable definitions do
6: update m with values found in the variable definitions
7: end for
8: for all policy elements do
9: if element is a policy then
10: enumerateVars(element,m)
11: else if element is a rule then
12: update m with values found in the rule targets
13: update m with values found in the rule condition
14: end if
15: end for
16: end procedure
ALGORITHM

• Constructing attribute domains

• Policy flattening

• SAT encoding
Policy Flattening (1)

- Applicability space $<AS_A, AS_{IN}, AS_{NA}>$
- Decision space $<DS_P, DS_{D}, DS_{IN}, DS_{NA}>$
Algorithm 2 FlattenPolicy

**Input:** A policy \( p \)

**Output:** Decision space

\(< DS_P, DS_D, DS_{IN(P)}, DS_{IN(D)}, DS_{IN(NA)}, DS_{NA} >\)

1: procedure FLATTENPOLICY\((p)\)

2: \hspace{1em}if \( p \) is a rule then

3: \hspace{2em}\( AS_P^A = AS_T^A \cap AS_A^C \)

4: \hspace{2em}\( AS_{IN}^P = AS_{IN}^C \cup AS_{IN}^T \)

5: \hspace{2em}if effect of \( p \) is Permit then

6: \hspace{3em}\( DS_P = AS_P^P \)

7: \hspace{3em}\( DS_D = \emptyset \)

8: \hspace{3em}\( DS_{IN(P)} = AS_P^P \)

9: \hspace{3em}\( DS_{IN(D)} = \emptyset \)

10: \hspace{2em}else if effect of \( p \) is Deny then

11: \hspace{3em}\( DS_P = \emptyset \)

12: \hspace{3em}\( DS_D = AS_P^A \)

13: \hspace{3em}\( DS_{IN(P)} = \emptyset \)

14: \hspace{3em}\( DS_{IN(D)} = AS_P^P \)

15: \hspace{2em}end if

16: \hspace{2em}\( DS_{IN(PD)} = \emptyset \)

17: \hspace{2em}\( DS_{IN(NA)} = (DS_P \cup DS_D \cup DS_{IN(P)} \cup DS_{IN(D)} \cup DS_{IN(PD)}) \)

18: return \((DS_P, DS_D, DS_{IN(P)}, DS_{IN(D)}, DS_{IN(PD)}, DS_{IN(NA)})\)

19: \hspace{2em}else if \( p \) is a policy (set) then

20: \hspace{3em}policies = \emptyset

21: \hspace{3em}for all elements \( e \) of \( p \) do

22: \hspace{4em}result = flattenPolicy\((e)\)

23: \hspace{4em}add result to policies

24: \hspace{3em}end for

25: combiningAlg = combining algorithm of \( p \)

26: return applyCA\((policies, combiningAlg)\)

27: end if

28: end procedure
ALGORITHM

• Constructing attribute domains

• Policy flattening

• SAT encoding
SAT ENCODING

\[ DS_P \cup DS_D \cup DS_{IN(P)} \cup DS_{IN(D)} \cup DS_{IN(PD)} \cup DS_{NA} \]
FIGURE 1. A graphical representation of the framework as created during this research. Step (1) shows the XACML policy being parsed by the preprocessor creating the cnf formula. Step (2) is feeding the formula combined with the parsed request (in step (3) and (4)) to the SMT solver which returns either sat or unsat. Depending on if the formula was satisfiable or not the output is fed to the post processor which returns the final access decision (step (6)).

REFERENCES


Conclusion

- Creating SAT formula
- SAT solvers
- No trees
- Experimental validation