An automated model-based test oracle for access control systems

Antonia Bertolino, Said Daoudagh, Francesca Lonetti, Eda Marchetti*
ISTI-CNR
Pisa, Italy
firstname.secondname@isti.cnr.it

ABSTRACT

In the context of XACML-based access control systems, an intensive testing activity is among the most adopted means to assure that sensible information or resources are correctly accessed. Unfortunately, it requires a huge effort for manual inspection of results: thus automated verdict derivation is a key aspect for improving the cost-effectiveness of testing. To this purpose, we introduce XACMET, a novel approach for automated model-based oracle definition. XACMET defines a typed graph, called the XAC-Graph, that models the XACML policy evaluation. The expected verdict of a specific request execution can thus be automatically derived by executing the corresponding path in such graph. Our validation of the XACMET prototype implementation confirms the effectiveness of the proposed approach.

CCS CONCEPTS

• Security and privacy → Access control; **• Theory of computation → Oracles and decision trees**; **• Software and its engineering → Software verification and validation**;

KEYWORDS

XACML, Testing, Oracle derivation

ACM Reference Format:
Antonia Bertolino, Said Daoudagh, Francesca Lonetti, Eda Marchetti. 2018. An automated model-based test oracle for access control systems. In *AST'18: AST'18:IEEE/ACM 13th International Workshop on Automation of Software Test , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3194733.3194743

1 INTRODUCTION

Security is a primary concern in modern pervasive and interconnected distributed systems. An important security aspect is constituted by *access control policies*, which specify which *subjects* can access which *resources* under which *conditions*. They are usually written using the eXtensible Access Control Markup Language (XACML) [15], an XML-based standard language proposed by OASIS, and rely on a specific architecture: incoming access requests are transmitted to the Policy Decision Point (PDP) that grants or denies the access based on the defined XACML policies. The criticality of the PDP component, as explained in [8], imposes an accurate testing activity that mainly consists of probing the PDP with a set of XACML requests and checking its responses against the expected decisions.

In literature, there are different proposals for automating PDP testing, including: mutation [13], coverage [14], random, combinatorial [3, 12] and model-based [20] techniques. However, they share an important drawback: the *lack of oracle*, i.e., for the generated requests the expected PDP decision is not provided. This is an important limitation, especially when test suites are large and manual inspection of results is unfeasible. Recently, Li et al. [11] proposed to implement a PDP automated oracle through voting, i.e. to locally or remotely access more than one PDP engine and collect their responses for the same request. The most frequent decision value is considered the correct one. Although effective, this solution has a high computation and implementation cost and could not be applied in low energy consuming environments. Other proposals, for instance [6], strictly bind the oracle definition to the proposed test generation approach and do not provide generic solutions able to evaluate any kind of requests.

In this paper, we introduce XACMET (XACML Modeling & Testing), a novel model-based approach to support the automation of XACML-based testing. Intuitively, XACMET builds from the XACML specification a typed graph, called the XAC-Graph, representing the XACML policy evaluation. Such graph can be exploited for several purposes, for example it allows for measuring the coverage of test requests in terms of the paths executed on the XAC-Graph. It can also help in deriving an adequate set of test requests such that all paths are executed at least once. However, for lack of space, we focus here on what is, to the best of our knowledge, the most novel feature supported by XACMET, i.e., *the first completely automated model-based oracle for XACML-based PDP testing*. XACMET represents an alternative approach for oracle derivation with respect to the well-known voting mechanism presented in [11]. We refer to [5] for the other features of XACMET.

In summary, the contributions of this paper include: i) the definition of the XAC-Graph for modeling XACML policies; ii) the automatic derivation of an XACML oracle based on the evaluation paths of the XAC-Graph; and iii) a first empirical evaluation of the XACMET oracle against two different oracle specifications.

The rest of this paper is structured as follows. Section 2 briefly introduces XACML. Section 3 overviews the basic idea for the oracle derivation while the XAC-graph and the oracle derivation are formally presented in Sections 4 and 5. Section 6 reports the validation of the proposal. Finally, Section 7 puts our work in context of related work and Section 8 draws conclusions.

*Said Daoudagh is also at University of Pisa, Department of Computer Science, Italy.

ACM ISBN 978-1-4503-5743-2/18/05…$15.00
https://doi.org/10.1145/3194733.3194743

2 XACML

XACML [15][1] is a platform-independent XML-based standard language for the specification of access control policies. Briefly, an XACML policy has a tree structure whose main elements are: PolicySet, Policy, Rule, Target and Condition. The PolicySet includes one or more policies. A Policy contains a Target and one or more rules. The Target specifies a set of constraints on attributes of a given request. The Rule specifies a Target and a Condition containing one or more boolean functions. If a request satisfies the target of the policy, then the set of rules of the policy is checked, else the policy is skipped. If the Condition evaluates to true, then the Rule's Effect (a value of Permit or Deny) is returned, otherwise a NotApplicable decision is formulated (Indeterminate is returned in case of errors). More policies in a policy set and more rules in a policy may be applicable to a given request. The PolicyCombiningAlgorithm and the RuleCombiningAlgorithm define how to combine the results from multiple policies and rules respectively in order to derive a single access result.

For example, the *first-applicable* rule combining algorithm returns the effect of the first applicable rule or *NotApplicable* if no rule is applicable to the request. The *deny-overrides* algorithm specifies that *Deny* takes the precedence regardless of the result of evaluating any of the other rules in the combination, then it returns *Deny* if there is a rule that is evaluated to *Deny*, otherwise it returns *Permit* if there is at least a rule that is evaluated to *Permit* and all other rules are evaluated to *NotApplicable*. Similarly, the *permit-overrides* algorithm returns *Permit* if there is a rule that is evaluated to *Permit*.

```
1  <Policy PolicyId="Pol_Ex" RuleCombiningAlgId="deny-overrides">
2   <Target/>
3   <Rule RuleId="ruleA" Effect="Deny">
4    <Target>
5     <Resources><Resource><ResourceMatch MatchId="string-equal">
6       <AttributeValue DataType="string">book</AttributeValue>
7       <ResourceAttributeDesignator AttributeId="resource-id"
           DataType="string"/>
8      </ResourceMatch></Resource>
9      <Resource><ResourceMatch MatchId="string-equal">
10      <AttributeValue DataType="string">document</AttributeValue>
11      <ResourceAttributeDesignator AttributeId="resource-id"
           DataType="string"/>
12     </ResourceMatch></Resource>
13     <Resource><ResourceMatch MatchId="string-equal">
14      <AttributeValue DataType="string">documententry</
           AttributeValue>
15      <ResourceAttributeDesignator AttributeId="resource-id"
           DataType="string"/>
16     </ResourceMatch></Resource></Resources>
17    <Actions><Action><ActionMatch MatchId="string-equal">
18      <AttributeValue DataType="string">write</AttributeValue>
19      <ActionAttributeDesignator AttributeId="action-id" DataType="
           string"/>
20     </ActionMatch></Action></Actions>
21   </Target>
22   <Condition><Apply FunctionId="string-is-in">
23     <Apply FunctionId="string-one-and-only">
24      <ResourceAttributeDesignator AttributeId="resource-id"
           DataType="string"/>
25     </Apply>
26     <SubjectAttributeDesignator AttributeId="subject-id1" DataType
          ="string"/>
27    </Apply>
28   </Condition>
29  </Rule>
30  <Rule RuleId="ruleB" Effect="Permit">
31   <Target>
32    <Subjects><Subject><SubjectMatch MatchId="string-equal">
33      <AttributeValue DataType="string">Julius</AttributeValue>
34      <SubjectAttributeDesignator AttributeId="subject-id" DataType
           ="string"/>
35     </SubjectMatch></Subject></Subjects>
36    <Resources><Resource><ResourceMatch MatchId="string-equal">
37      <AttributeValue DataType="string">journals</AttributeValue>
38      <ResourceAttributeDesignator AttributeId="resource-id"
           DataType="string"/>
39     </ResourceMatch></Resource></Resources>
40    <Actions><Action><ActionMatch MatchId="string-equal">
41      <AttributeValue DataType="string">read</AttributeValue>
42      <ActionAttributeDesignator AttributeId="action-id" DataType="
           string"/>
43     </ActionMatch></Action></Actions>
44   </Target>
45  </Rule>
46 </Policy>
```

Listing 1: An XACML policy

We show in Listing 1 an example of a simplified XACML policy ruling library access. Its target (line 2) says that this policy applies to any subject, resource and action. This policy has a first rule, *ruleA* (lines 3-29), with a target (lines 4-21) specifying that this rule applies only to the access requests of a "write" action of "book", "document" and "documententry" resources. The rule condition will be evaluated true when the request resource value is contained into the set of request subject values. The effect of the second rule *ruleB* (lines 30-45) is *Permit* when the subject is "Julius", the action is "read", and the resource is "journals". The rule combining algorithm of the policy (line 1) is *deny -overrides*.

3 TEST ORACLE

By referring to the example policy of Listing 1, in this section we explain the underling idea of the approach used for the oracle definition. We refer to Section 4 for formal details.

Given a generic request, the result of the evaluation of an XACML policy with that request strictly depends on: the request values, the policy constraints as well as the combining algorithm that prioritizes the evaluation of the policy rules. Specifically, we define an *evaluation path* as a sequence of policy elements that are exercised by the request during the evaluation of an XACML policy and the verdict associated to that request. Thus, the general idea of the XACMET approach is to derive all possible evaluation paths from the policy specification and order them according to the rule combining algorithm. For instance, let us consider the policy of Listing 1, having as elements, the rules *ruleA* and *ruleB* and *deny-overrides* as the combining algorithm (line 1), the possible evaluation paths are: (1) *ruleA* evaluated to true and *ruleB* evaluated to false: the associated verdict is *Deny*, i.e., the effect of the first rule; (2) *ruleA* evaluated to false and *ruleB* evaluated to true: the associated verdict is *Permit*, i.e., the effect of the second rule; (3) *ruleA* and *ruleB* both evaluated to false: the associated verdict is *NotApplicable*; (4) *ruleA* and *ruleB* both evaluated to true: the associated verdict is *Deny*, because it takes the precedence regardless of the result of the second rule. Note that, for aim of simplicity, we do not explicitly consider the *Indeterminate* value that is returned in case of errors of the policy evaluation, and we assume that in this case the associated verdict is *NotApplicable*. This set of paths is ordered according to

[1]The current implementation of the presented approach is compliant with XACML 2.0 but it can be easily extended to address the functionalities of XACML 3.0.

the semantics of the rule combining algorithm, and then according to the verdict associated to each path. For instance, in case of *deny-overrides* combining algorithm, first the paths having *Deny* are evaluated, then those having *Permit* and finally those having *NotApplicable*. For paths having the same verdict, the evaluation order of the paths is based on their length, namely the shortest path takes the precedence. For the policy of Listing 1, the order of the evaluated paths is (1), (4), (2) and (3).

The ordered set of paths is then used for the requests evaluation and the verdicts association. For each request, the first path for which all the path constraints are satisfied by the request values is identified, the final verdict associated to the request is derived, and the path is considered covered by the request. For instance, considering the ordered set of paths of policy of Listing 1, a request asking to `write` a `documententry` does not match paths (2), (3) and (4), but covers (satisfies) path (1) since it satisfies only *ruleA*, so the associated verdict is *Deny*.

In addition, XACMET also provides the possibility of automatically generating a set of test cases that guarantee the full coverage of the evaluation paths. Indeed, each of the evaluation path represents the set of constraints that should be satisfied by some specific request values so to reach the final verdict. Thus, the set of values satisfying (not satisfying) the identified set of path constraints can be identified using a constraint satisfaction approach [19]. As a side effect, when the various constraints are combined, possible inconsistencies between the selected values can be detected, which hints at the potential presence of unfeasible paths in the policy specification. Moreover, XACMET application gives the possibility of knowing which and how many evaluation paths are covered by a test set. This information can be useful to improve the policy itself and avoid possible security flaws. For space limitation, the test cases generation functionality of XACMET is not further described in this paper. We refer to [5] for more details.

4 XAC-GRAPH

In this section, we provide some formal definitions related to the XAC-graph model.

With reference to the XACML policy, as an XML document it can be represented as a tree, called the XAC-Tree. In particular, the following concepts can be used:

- *Contained:* Element i is contained within element j if i is between the start-tag and the end-tag of j.
- *Parent:* Element i is the parent of element j when j is contained within i and i is exactly one level above j.
- *Sibling*: The siblings in an XML document are the elements that are on a same level of the tree and have the same parent. In particular, given parent i of elements j and k, j is left (right) sibling of k if j is contained just before (after) k within element i.

The XAC-Tree derivation exploits the parent relationship of the XACML policy and uses the following sets of types and values:

- T_V = {Policy, Target Rule, Subjects, Subject, Resources, Resource, Actions, Action, Environments, Environment};
- T_{V_a} = {RuleAlgorithm, Effect, NotApplicable};
- T_{V_v} ={ReturnPermit, ReturnDeny, ReturnNotApplicable};
- `RCA`= {FirstApplicable, DenyOverrides, PermitOverrides};
- `RE` = {Permit, Deny}.

Formally, the parent relationship, called XAC-TreeParent, is defined as:

Definition 1 (XAC-TreeParent). *Given a tree T=(V, E, Root) in which every vertex v in V has an associated type $t_v \in T_V$: Element i ∈ V is* XAC-TreeParent *of j ∈ V if i is parent of j in the XACML policy document.*

From this, the definition of the XAC-Tree as in the following:

Definition 2 (XAC-Tree). *Given an XACML policy, the XAC-Tree is a labeled and typed tree (V, E, Root) where*

- *V is a set of vertices such that every v ∈ V has type $t_v \in T_V$;*
- *Each vertex v in V has parameter i with i = 1,.., n;*
- *E is a set of edges (i,j) such that i,j ⊆ V and i is* XAC-TreeParent *of j;*
- *Root is a vertex v in V with t_{Root} = Policy;*
- *Root has an attribute called* RuleCombAlg ∈ *RCA;*
- *Each vertex v ∈ V with t_v = Rule has an attribute called* EffectRule ∈ *RE.*

Figure 1 shows the XAC-Tree associated to the policy of Listing 1. In particular, *ruleA* becomes the node `Rule_3` into the XAC-Tree. In this case, 3 is the suffix of the node and the EffectRule attribute of the node `Rule_3` is set to Deny as specified in the Effect of *ruleA*. Moreover, there is a XAC-TreeParent relation between `Policy_1` and `Rule_3` nodes because *ruleA* is contained within `policyExample` in Listing 1.

The representation of the XACML policy is then used to derive a model of the XACML evaluation. For this, a parent relationships, called XAC-GraphParent, is defined as in the following:

Definition 3 (XAC-GraphParent). *Given a graph G= (V_g,E_g,Entry) and XT = (V,E,Root) a XAC-Tree, where Entry = Root and $V_g = V \bigcup V_a^\star \bigcup V_v^\star$ with (i) V_a is a set of vertices such that every v ∈ V_a has type $t_v \in T_{V_a}$; (ii) V_v is a set of vertices such that every v ∈ V_v has type $t_v \in T_{V_v}$.*

Element i ∈ V_g is in a XAC-GraphParent *relation with j ∈ V_g if:*

1. *i is left sibling of j in XT, $t_j \neq$ Rule and i has not children in XT;*
2. *i is leaf in XT and ∃ k ∈ V_g such that k is XAC-TreeParent of i and k is left sibling of j in XT and $t_j \neq$ Rule;*
3. *j has no left sibling in XT and i is XAC-TreeParent of j in XT;*
4. *t_j ∈ {Effect, NotApplicable}, i is leaf in XT, $t_i \neq$ Target, ∃ k ∈ V such that t_k=Rule and i is the rightmost leaf of the subtree rooted in k in XT;*
5. *t_i ∈ {Effect, NotApplicable} and t_j = RuleAlgorithm;*
6. *t_i= RuleAlgorithm, t_j = ReturnPermit if ∃ k∈ V, t_k=Rule such that the value of the attribute EffectRule of k is Permit;*
7. *t_i= RuleAlgorithm, t_j = ReturnDeny if ∃ k∈ V, t_k=Rule such that the value of the attribute EffectRule of k is Deny;*
8. *t_i= RuleAlgorithm and t_j = Rule, and if k ∈ V is the left sibling of j in XT than $t_k \neq$ Target.*

Finally, a labelled and typed graph, called the XAC-Graph, is defined as in the following:

Definition 4 (XAC-Graph). *Let XT = (V,E,Root) be a XAC-Tree, a policy graph (XAC-Graph) is a graph (V_g,E_g,Entry) where*

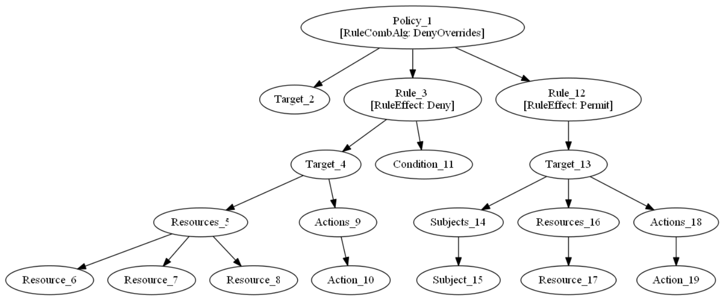

Figure 1: XAC-Tree. Label T_P means node of type T and parameter P. The attributes are within square brackets.

- *Entry = Root;*
- $V_g = V \bigcup V_a^\star \bigcup V_v^\star$
- *V_a is a set of vertices such that every $v \in V_a$ has type $t_v \in T_{V_a}$;*
- *V_v is a set of vertices such that every $v \in V_v$ has type $t_v \in T_{V_v}$;*
- *Each vertex v in V_a with t_v = Effect has an attribute called* EffectValue ∈ *RE;*
- *Each vertex v in V_a has parameter i with i = 1,.., n;*
- *The vertex v in V_a with t_v = RuleAlgorithm is unique and has an attribute called* Algorithm *of value equal to the value of the attribute RuleCombAlg of the Root in XT;*
- *E is a set of edges (i,j) such that i,j ⊑ V_g and i is XAC-GraphParent of j;*
- *Each vertex j, j, k ∈ V_g with t_j = Effect, i XAC-GraphParent for the point 4 of the definition 3 and t_k=Rule the value of the attribute EffectValue of j is equal to value of the attribute EffectRule of k.*

Figure 2 shows the XAC-Graph of the XAC-Tree of Figure 1. In particular, considering the conditions of Definition 3, in XAC-Graph there exists a XAC-GraphParent relation between:

- `Target_2` and `Rule_3` due to Condition 1;
- `Resource_7` and `Actions_9` due to Condition 2;
- `Policy_1` and `Target_2` due to Condition 3;
- `Condition_11` and `NotApplicable_3` due to Condition 4;
- `Condition_11` and `Effect_3` due to Condition 4;
- `Effect_3` and `RuleAlgorithm` due to Condition 5. Note that `Policy1` and `RuleAlgorithm` have the same algorithm value;
- `RuleAlgorithm` and `ReturnPermit` due to Condition 6;
- `RuleAlgorithm` and `ReturnDeny` due to Condition 7;
- `RuleAlgorithm` and `Rule_12` due to Condition 8.

The XAC-graph could basically be derived applying a depth-first search approach to the XAC-tree. For the sake of simplicity, we show the XACMET approach applied to a policy rooted in the Policy element. Definitions 1, 2, 3, 4, can be easily extended to also consider the *PolicySet* and the XACMET approach can be used also for deriving the XAC-Graph considering a policy rooted in the *PolicySet* node.

5 ORACLE DERIVATION

In this section, we detail the approach used for the oracle derivation that is based on the XAC-Graph paths identification. Specifically, the process adopted is divided into two main steps: *coloring, unfolding*.

The coloring process exploits the concepts of *Forward Node* defined as: given the XAC-Graph G = (V_g , E_g, Entry), for each node i ∈ V_g it is possible to identity the Forward Node $FN(i) \in V_g$ as the set of nodes j such that i is a XAC-GraphParent of j. Thus, given a XAC-Graph for each node b and c ∈ V_g, with t_b ∈ {Subject, Resource, Action, Environment}, the cardinality of $FN(b)$ = 2, and t_c ∈ {Subject, Resource, Action, Environment, NotApplicable}, the coloring process marks each edge (b,c)∈ $FN(b)$: with red dashed line if $t_b = t_c$ or t_c = NotApplicable; with blue dotted line otherwise. In practice the red dashed edges represent a successful evaluation of the node b.

During the unfolding process the paths are obtained by visiting the XAC-Graph from the Entry node to each node in V_v. The cycles are due to the presence of the node typed `Rule_Algorithm`. In XACMET, the order of the paths strictly depends on the order in which the rules are evaluated, which in turn is guided by the FirstApplicable, DenyOverrides, PermitOverrides algorithms.

Thus, let P a path of k nodes on the XAC-Graph and h the last included one, with t_h =Rule_Algorithm. If the value of Algorithm attribute of h is equal to:

FirstApplicable and the node k-1 has type:

- Effect, and EffectValue = Deny (Permit), then the next node has type ReturnDeny (ReturnPermit);
- NotApplicable, then the next node v has type Rule iff v is not already included in P (ReturnNotApplicable otherwise).

DenyOverrides and the node k-1 has type

- Effect and EffectValue = Deny, then the next node has type ReturnDeny;
- Effect and EffectValue = Permit, then the next node v has type Rule iff v is not already included in P (ReturnPermit otherwise);
- NotApplicable, then the next node v has type Rule iff v is not already included in P;
- NotApplicable and each node v with Rule ∈ P the next node has type ReturnPermit, if ∃ a node p ∈ P with t_p=Effect (ReturnNotApplicable otherwise).

Permit_Overrided and the node k-1 has type

- Effect and EffectValue = Permit, then the next node has type ReturnPermit;
- Effect and EffectValue = Deny, then the next node v has type Rule iff v is not already included in P (ReturnDeny otherwise);

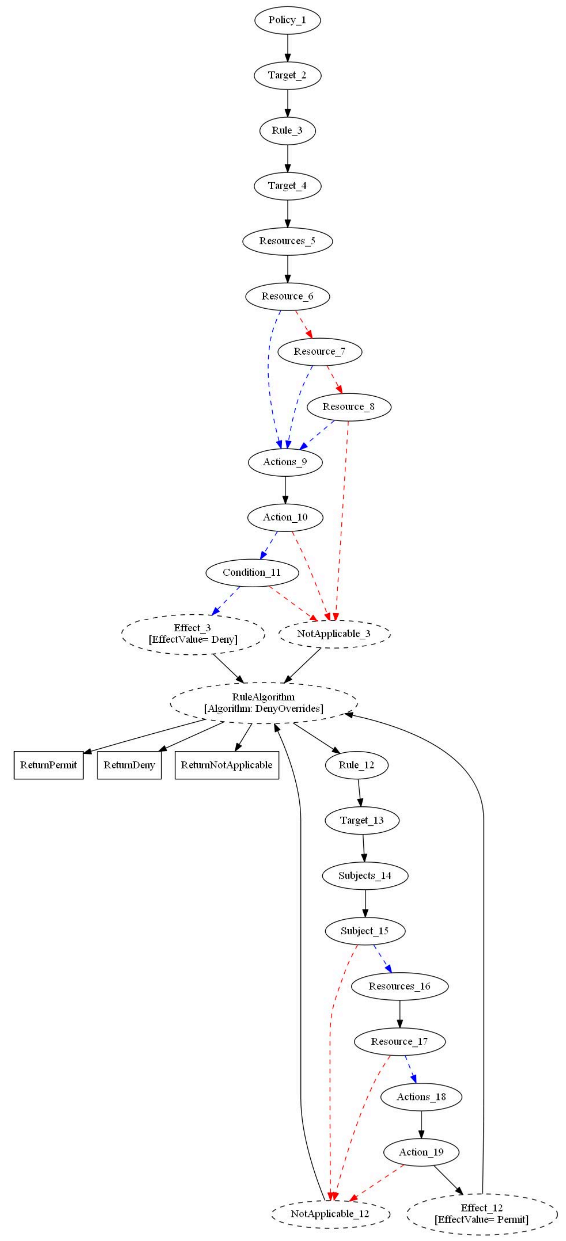

Figure 2: XAC-Graph. Label T_P means node of type T and parameter P. The attributes are within square brackets.

- NotApplicable, then the next node v has type Rule iff v is not already included in P;
- NotApplicable and each node v with Rule $\in P$ the next node has type ReturnDeny, if $\exists$ a node p $\in P$ with t_p=Effect (ReturnNotApplicable otherwise).

In Figure 3, we show a path of XAC-Graph.

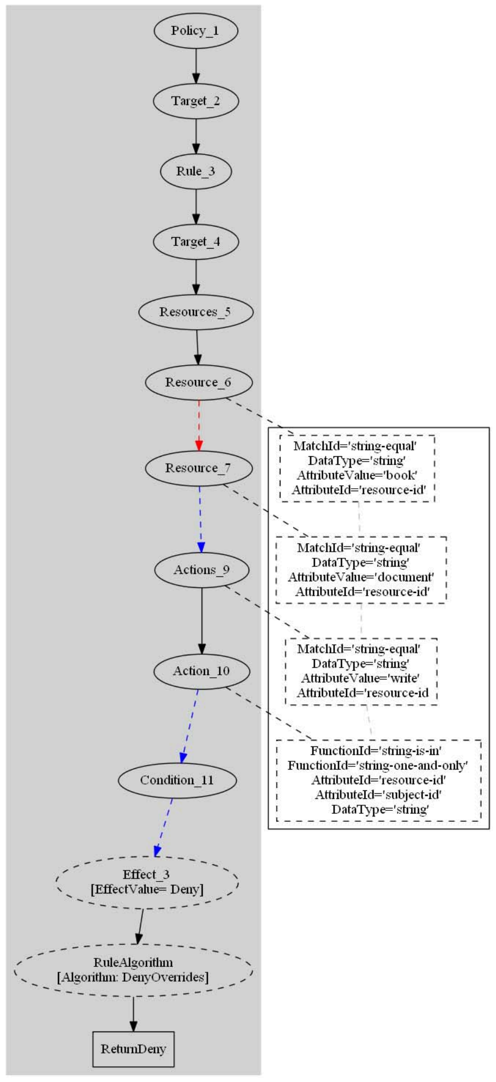
Policy_1
Target_2
Rule_3
Target_4
Resources_5
Resource_6
Resource_7
Actions_9
Action_10
Condition_11
Effect_3 [EffectValue= Deny]
RuleAlgorithm [Algorithm: DenyOverrides]
ReturnDeny
MatchId='string-equal' DataType='string' AttributeValue='book' AttributeId='resource-id'
MatchId='string-equal' DataType='string' AttributeValue='document' AttributeId='resource-id'
MatchId='string-equal' DataType='string' AttributeValue='write' AttributeId='resource-id
FunctionId='string-is-in' FunctionId='string-one-and-only' AttributeId='resource-id' AttributeId='subject-id' DataType='string'

Figure 3: A path of XAC-Graph. The boxes connected to the nodes contain the functions and values.

6 EMPIRICAL EVALUATION

We conducted an empirical evaluation of the XACMET oracle. The evaluation includes two studies: a first study assessed the compliance of the XACMET oracle with the XACML specification given by the conformance test suite; the focus of the second study was to assess the compliance of the XACMET oracle with respect to a voting PDP mechanism, considering both the policies of conformance test suite and some real policies.

6.1 First Study

In this first study, we considered the tests of the XACML 2.0 Conformance Tests V0.4 [16]. Each test consists of three elements: an XACML policy, an XACML request, and an XACML response representing the expected access decision associated to that request. We focused on the subset of tests implementing the mandatory functionalities and specifically on the following groups of tests: i) IIA: tests exercising attribute referencing; ii) IIB: tests exercising target matching; iii) IIC: test exercising function evaluation; and finally iv) IID: tests exercising combining algorithms. For each group, we selected a subset of tests specifying only the functionalities implemented in XACMET. In particular, we excluded by IIA group the tests referring to *Indeterminate* values, in IIC group we considered only the most used arithmetic and equality functions (for instance *type-equal*) while for IID group we considered the combining algorithms presented in Section 5. In Table 1 (column 1), we show for each group the percentage of considered tests. Moreover, in rows from 4 to 7, the structure of the considered XACML policies belonging to the four groups is described in terms of cardinality of policies, rules, conditions, subjects, resources, actions and functions (total number of functions and distinct functions) respectively. Table 1 (column 9) shows the number of XACML requests for the four groups of tests. To validate the XACMET oracle, we applied the XACMET approach to the policies of the conformance tests. Specifically, for each test case we derived starting from the XACML policy, the associated XAC-Graph and an ordered set of paths as described in Section 4 and 5. Then, we evaluated the XACML request belonging to the test case, against the obtained set of paths, we identified the first covered path and derived the verdict associated to that path. Finally, we compared this verdict with the decision value specified in the response belonging to the test case. For all the tests of the conformance, we obtained that the XACMET verdict coincides with the expected access decision. This improves the confidence in the effectiveness of the XACMET approach for the XACML functionalities specified in the conformance tests.

6.2 Second Study

The goal of this second study was to evaluate the XACMET oracle with respect to other competing automated oracles. We referred to the already mentioned black box approach of multiple implementations testing presented in [11]. Hence we derived an automated majority oracle by running the request on three implementations of an XACML based PDP and then using a majority voting to derive the expected XACML response associated to that request. Specifically, we used a pool of three XACML PDPs, namely Sun PDP [2], Herasaf PDP [3] and Balana PDP [4].

In this second study, we used XACMET to derive starting from a XACML policy a set of XACML requests and the associated verdict as described in Section 3. The considered policies were: i) those of the conformance tests described before; ii) the Fedora (Flexible Extensible Digital Object Repository Architecture) XACML policies (demo-5, demo-11 and demo-26) [5] and other six policies released in the context of the TAS3 European project [6]. For each policy, Table 1 (column 10) shows the cardinality of the derived XACML requests. Each policy and the derived set of requests were executed on the PDPs pool, we observed whether the different PDPs produced the same responses and took as oracle value the majority output. For each request we compared the XACMET oracle with the majority output of the PDPs pool. We observed that for all requests the XACMET oracle coincided with the automated majority oracle. This enhances the confidence of the effectiveness of the XACMET oracle considering also the functionalities of real policies. Moreover, in this second study we also applied the test cases generation functionalities of XACMET. It is out of scope of this paper to validate them and we refer to [5] for an experimental evaluation of XACMET test suites. However, as side effect of this experiment we obtained an enhancement of the requests of the conformance tests. As showed in Table 1 (column 10), this enhancement is evident mostly for IIB policies group (201 additional requests). This is due to the more complex structure of the policies of this group as evidenced also by the higher number of elements (mainly actions and functions) of these policies.

[2]Sun PDP is available at: http://sunxacml.sourceforge.net.
[3]Herasaf PDP is available at: https://bitbucket.org/herasaf/herasaf-xacml-core.
[4]Balana PDP is available at: https://github.com/wso2/balana.
[5]Fedora Commons Repository Software. http://fedora-commons.org.
[6]Trusted Architecture for Securely Shared Services. http://www.tas3.eu.

7 RELATED WORK

The work presented in this paper spans over the following research directions:

Analysis and modeling of policy specification. Available proposals include different verification techniques [21], such as model-checking [22] or SAT solvers [19]. Well-known analysis and verification tools for access control policies are: i) Margrave [7], which represents policies as Multi-Terminal Binary Decision Diagrams (MTBDDs) and can answer queries about policy properties; and ii) ACPT (Access Control Policy Testing) tool [9] that transform policies into finite state machines and represent static, dynamic and historic constraints into Computational Tree Logic. The capabilities and performances of such tools are analytically evaluated in [10].

Differently from the above approaches, XACMET models the expected behaviour of the evaluation of a given XACML policy as a labeled graph and guarantees the full path coverage of such graph. Moreover, our proposed XAC-Graph model is richer since it also represents the rule combining algorithm, the functions, and the associated conditions. The authors of [17] provide an optimized approach for XACML policies modeling based on tree structures aimed at fast searching and evaluation of applicable rules. Differently from our proposal, the main focus of this work is on performance optimization more than on oracle derivation.

Test cases and oracle derivation. Considering the automated test cases generation, solutions have been proposed for testing either the XACML policy or the PDP implementation [2, 3]. Among them, the most referred ones such as X-CREATE and the Targen tools [3, 12] use combinatorial approaches for test cases generation. However, combinatorial approaches are shallow with respect to policy semantics.

Model-based testing has already been widely investigated for policy testing, e.g. [18, 20]. Such approaches provide methodologies or tools for automatically generating access control test models from functional models and access control rules. The key original aspect of our approach is in the XAC-Graph model which we derive, which is richer in expressiveness than other proposed models, and can provide directly the evaluation paths including a verdict associated to a request. About the automated oracle, notwithstanding the huge interest devoted to this topic, reducing the human

Table 1: Experimental values

XACML Policy	Functionality							# XACML Request	
	# Policy	# Rule	# Cond	# Sub	# Res	# Act	# Funct (distinct)	I Study	II Study
Conformance Test Suite XACML Policies									
IIA (90%)	18	18	12	18	8	16	112 (12)	18	68
IIB (100%)	53	53	6	51	50	98	410 (7)	53	254
IIC (10%)	22	22	22	18	3	1	102 (19)	22	31
IID (17%)	5	13	7	13	-	-	60 (5)	5	19
Real-world XACML Policies									
2_73020419964_2	1	6	5	3	3	0	4	-	8
create-document-policy	1	3	2	1	2	1	3	-	5
demo-5	1	3	2	2	3	2	4	-	13
demo-11	1	3	2	2	3	1	5	-	8
demo-26	1	2	1	1	3	1	4	-	16
read-document-policy	1	4	3	2	4	1	3	-	6
read-informationunit-policy	1	2	1	0	2	1	2	-	4
read-patient-policy	1	4	3	2	4	1	3	-	6
Xacml-Nottingham-Policy-1	1	3	0	24	3	3	2	-	18

activity in the evaluation of the testing results is still an issue [1]. The automated oracle derivation is a key aspect in the context of XACML systems and testers need usually to manually verify the XACML responses. The few available solutions mainly deal with model based approaches. Specifically, the authors of [6] provide an integrated toolchain including test case generation as well as policy and oracle specification for the PDP testing. Other proposals such as [4] address the use of monitoring facilities for the assessment of the run-time execution of XACML policies. Differently from the above approaches, the main benefits of XACMET deal with the derivation of an XACML verdict for each XACML request.

8 CONCLUSIONS

We have introduced a novel model-based approach to automatic generation of XACML oracle for testing policy evaluation engines. The XACMET approach fully automatically derives a verdict for each XACML request by considering the set of evaluation paths derived from the obtained graph. We have illustrated the approach on an example policy and provided experimental results evidencing the effectiveness of our proposal with respect to the oracle provided in the XACML conformance tests and an automated oracle implemented as a voting mechanism.

In the future, we plan to extend our automated oracle in order to consider more functionalities of the XACML conformance policies, such as the different combining algorithms and the *PolicySet* element. The XACMET approach will also be extended to be compliant with the last XACML 3.0 standard version. Future work will also include further experimentation of XACMET, and its comparison with other model-based approaches.

ACKNOWLEDGMENTS

This work has been partially supported by the GAUSS national research project (MIUR, PRIN 2015, Contract 2015KWREMX).

REFERENCES

[1] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The oracle problem in software testing: A survey. *IEEE transactions on software engineering* 41, 5 (2015), 507–525.

[2] Antonia Bertolino, Said Daoudagh, Francesca Lonetti, Eda Marchetti, Fabio Martinelli, and Paolo Mori. 2014. Testing of PolPA-based usage control systems. *Software Quality Journal* 22, 2 (2014), 241–271.

[3] Antonia Bertolino, Said Daoudagh, Francesca Lonetti, Eda Marchetti, and Louis Schilders. 2013. Automated testing of eXtensible Access Control Markup Language-based access control systems. *IET Software* 7, 4 (2013), 203–212.

[4] Antonello Calabrò, Francesca Lonetti, and Eda Marchetti. 2017. Access Control Policy Coverage Assessment Through Monitoring. In *Proc. of TELERISE*. 373–383.

[5] Said Daoudagh. 2017. *A Data Warehouse and a Framework for the Validation and Testing of Access Control Systems.* Master's thesis. University of Pisa, Italy.

[6] Said Daoudagh, Donia El Kateb, Francesca Lonetti, Eda Marchetti, and Tejeddine Mouelhi. 2015. A toolchain for model-based design and testing of access control systems. In *Proc.of MODELSWARD*. IEEE, 411–418.

[7] K. Fisler, S. Krishnamurthi, L.A. Meyerovich, and M.C. Tschantz. 2005. Verification and change-impact analysis of access-control policies. In *Proc. of ICSE*. 196–205.

[8] JeeHyun Hwang, Evan Martin, Tao Xie, and Vincent C. Hu. 2011. Policy-Based Testing. In *Encyclopedia of Software Engineering*. Taylor & Francis, 673–683.

[9] JeeHyun Hwang, Tao Xie, Vincent Hu, and Mine Altunay. 2010. ACPT: A tool for modeling and verifying access control policies. In *Proc. of International Symposium on Policies for Distributed Systems and Networks (POLICY)*. 40–43.

[10] Ang Li, Qinghua Li, Vincent C Hu, and Jia Di. 2015. Evaluating the capability and performance of access control policy verification tools. In *Proc. of MILCOM*. 366–371.

[11] Nuo Li, JeeHyun Hwang, and Tao Xie. 2008. Multiple-implementation testing for XACML implementations. In *Proc. of TAV-WEB*. 27–33.

[12] Evan Martin and Tao Xie. 2006. Automated Test Generation for Access Control Policies. In *Supplemental Proc. of ISSRE*.

[13] Evan Martin and Tao Xie. 2007. A Fault Model and Mutation Testing of Access Control Policies. In *Proc. of WWW*. 667–676.

[14] Evan Martin, Tao Xie, and Ting Yu. 2006. Defining and Measuring Policy Coverage in Testing Access Control Policies. In *Proc. of ICICS*. 139–158.

[15] OASIS. 2005. eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf. (February 2005).

[16] OASIS. 2005. XACML 2.0 Conformance Tests v0.4. https://www.oasis-open.org/committees/document.php?document_id=14846. (February 2005).

[17] Santiago Pina Ros, Mario Lischka, and Félix Gómez Mármol. 2012. Graph-based XACML evaluation. In *Proc. of the 17th ACM symposium on Access Control Models and Technologies*. 83–92.

[18] A. Pretschner, T. Mouelhi, and Y. Le Traon. 2008. Model-based tests for access control policies. In *Proc. of ICST*. 338–347.

[19] Fatih Turkmen, Jerry den Hartog, Silvio Ranise, and Nicola Zannone. 2015. Analysis of XACML policies with SMT. In *Proc. of International Conference on Principles of Security and Trust*. Springer, 115–134.

[20] Dianxiang Xu, Michael Kent, Lijo Thomas, Tejeddine Mouelhi, and Yves Le Traon. 2015. Automated model-based testing of role-based access control using predicate/transition nets. *IEEE Trans. Comput.* 64, 9 (2015), 2490–2505.

[21] Dianxiang Xu and Yunpeng Zhang. 2014. Specification and analysis of attribute-based access control policies: An overview. In *Proc. of Eighth International Conference on Software Security and Reliability-Companion (SERE-C)*. IEEE, 41–49.

[22] Nan Zhang, Mark Ryan, and Dimitar Guelev. 2005. Evaluating Access Control Policies Through Model Checking. In *Information Security*. Lecture Notes in Computer Science, Vol. 3650. 446–460.