Conformance Testing of Balana: An Open Source Implementation of the XACML3.0 Standard

Sung-Ju Fan Chiang
Department of Computer Science
Boise State University
Boise, ID 83725, USA
sungjufanchiang@u.boisestate.edu

Daniel W. Chen
Department of Economics
University of Chicago
Chicago, IL 60637, USA
danielwchen@uchicago.edu

Dianxiang Xu
Department of Computer Science
Boise State University
Boise, ID 83725, USA
dianxiangxu@boisestate.edu

***Abstract*—As a new generation access control method, Attribute-Based Access Control (ABAC) has gained increasing attention. Currently, Balana is the only open-source implementations of XACML 3.0, which is an OASIS standard for specifying ABAC. Considering that XACML is much more complex than traditional access control models, conformance testing of any XACML implementation is an important problem. Using a non-conformance implementation may lead to misunderstanding of access decisions or even security violations. This paper presents an approach to conformance testing of Balana, focusing on the main elements of the XACML3.0 language, such as targets, rules, policies, and policy sets. In particular, we have thoroughly tested the key rule combining algorithms in policies and policy combining algorithms in policy sets. This has revealed several conformance issues.**

Keywords—attribute-based access control; Balana; conformance testing; decision tables; XACML.

I. INTRODUCTION

Access control is a fundamental mechanism for preventing malicious or accidental security violation. An access control policy specifies the conditions under which access to resources can be granted and to whom [12]. With the increasing system complexity, access control methods have evolved from Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role-Based Access Control (RBAC) to Attribute-Based Access Control (ABAC). ABAC combines various attributes of authorization elements into access control decisions [6]. These attributes are predefined characteristics of subjects (e.g., job title and age), resources (e.g., data, programs, and networks), actions, and environments (e.g., current time and IP address.

XACML (eXtensible Access Control Markup Language) [11] is an OASIS standard for specifying ABAC policies in the XML format. It can be used within a large enterprise or across multiple organizations. Currently Balana [1] is the only open source implementations of XACML 3.0. It is worth pointing out that the original open source implementation of XACML from Sun Microsystems, Inc. only supports 1.0 and 2.0. While it is believed to be upgraded to 3.0 in Oracle's Identity Server, the upgraded version is no longer open source. XACML3.0 is much more complex than traditional access control methods such as RBAC. For example, XACML 3.0 provides various combining algorithms to support rule and policy composition. A combining algorithm aims at rendering a single access decision by combining the decisions of individual access control rules or policies. The standard specification of XACML3.0 lacks a rigorous representation of the semantics of the combining algorithms. Our prior work on the formalization of the semantic differences between various combining algorithms in XACML 3.0 shows that, for any pair of rule (or policy) combining algorithms studied, they can be functionally equivalent with respect to certain rules (or policies) [13]. The similarities and differences among the combining algorithms are subtle. This increases the likelihood of having errors. Thus, conformance testing of any XACML3.0 implementation is an important issue. Using a non-conformance XACML implementation may lead to misunderstanding of access decisions or even security violations.

In this paper, we present our work on the systematic testing of conformance between Balana and XACML3.0. It focuses on the main language elements of XACML3.0, such as targets, rules, policies (including rule combining algorithms), and policy sets (including policy combining algorithms). As XACML is a logic-based language, we use decision tables as the main technique for formulating the semantics of these language elements and their test requirements. Although decision tables are a traditional technique, the particular decision tables resulted from this research provide an accurate understanding of the meanings of the main XACML 3.0 elements. They offer important guidelines for XACML3.0 practitioners. In addition, they are useful for testing other implementations of XACML3.0 in order to verify functional conformance. Based on the decision tables, the conformance tests for targets, rules, and policies are created manually. For policy sets and policy combining algorithms, however, all conformance tests are generated automatically from the existing conformance tests. In our experiment, all conformance tests are executed automatically. The results have revealed several conformance issues in Balana.

The remainder of this paper is organized as follows. Section II gives a brief introduction to the main XACML language elements. Section III describes our conformance testing method. Section IV presents the results of our conformance testing experiment. Section V reviews related work. Section VI concludes this paper.

II. XACML LANGUAGE ELEMENTS

The first-class entities in XACML are policy and policy set. A policy is composed of an optional policy target, one or more rules, and a rule combining algorithm. A policy set consists of

DOI reference number: 10.18293/SEKE2016-082

an optional policy set target, one or more sub-policy sets or policies, and a policy combining algorithm. Figure 1 shows the main elements of XACML 3.0 and their relationships.

A rule consists of a target, a condition, and an effect. The target is a logical expression that specifies the set of requests to which the rule is intended to apply. The logical operators are *AnyOf* and *AllOf*. Specifically, a target consists of zero or more *AnyOf* clauses, and each *AllOf* clause is made up of one or more match predicate. The condition is a Boolean expression that refines the applicability of the rule established by the target. Predicates in target and condition are defined over attributes and attribute values (e.g., gender is male).

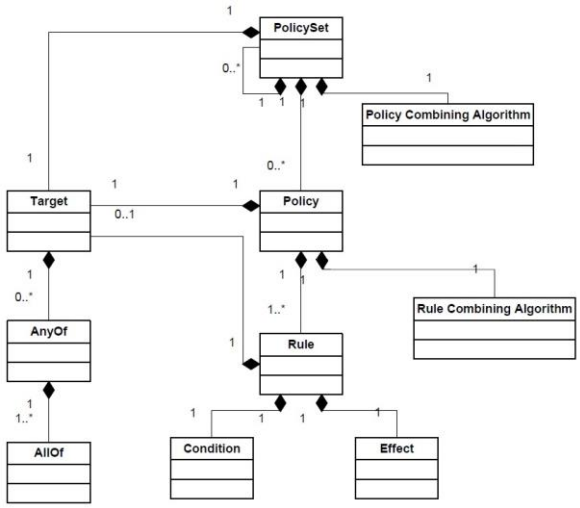

Figure 1. Main language elements of XACML 3.0 [11]

An access request consists of attribute and value pairs. Given a request, the decision of a policy depends on the policy target, the decisions of individual rules in the policy, and the rule combining algorithm. Each rule may yield one of the following decisions if the policy target evaluates to true:

- *Permit*: access is granted when the rule effect is *Permit* and the rule target evaluates to *Match* and the rule condition evaluates to true.
- *Deny*: access is denied when the rule effect is *Deny*, the rule target evaluates to *Match*, and the rule condition evaluates to true.
- *Not-Applicable*, denoted as *N/A* in this paper: either the rule target evaluates to *No Match* or the rule condition evaluates to false.
- *Indeterminate Deny*, denoted as *I(D)*: An error occurred when the rule target or the rule condition was evaluated. The decision could have evaluated to *Deny* if no error had occurred.
- *Indeterminate Permit*, denoted as *I(P)*: An error occurred when the rule target or the rule condition was evaluated. The decision could have evaluated to *Permit* if no error had occurred.

The rule combining algorithm combines the decisions of individual rules into a single policy-level decision. In addition to the above decisions, a policy decision can be *Indeterminate Deny Permit*, denoted as *I(DP)*. *I(D)*, *I(P)* and *I(DP)* will be a plain *Indeterminate* if it is the final decision returned by the XACML engine.

XACML3.0 provides 11 rule combining algorithms. Four of them are for compatibility support of old versions - *Legacy Ordered-deny-overrides*, *Legacy Permit-overrides*, *Legacy Ordered-permit-overrides*, and *Legacy Ordered-permit-overrides*. In Balana, the implementations of *Ordered-deny-overrides* and *Ordered-permit-overrides* are the same as *Deny-overrides* and *Permit-overrides*. Thus this paper focuses on the following five rule combining algorithms:

- *Deny-overrides*: Intended for those cases where a *Deny* decision should have priority over a *Permit* decision;
- *Permit-overrides*: Intended for the cases where a *Permit* decision should have priority over a *Deny* decision.
- *Deny-unless-permit*: Intended for those cases where a *Permit* decision should have priority over a *Deny* decision, and an *Indeterminate* or *N/A* must never be the result.
- *Permit-unless-deny*: Intended for those cases where a *Deny* decision should have priority over a *Permit* decision, and an *Indeterminate* or *N/A* must never be the result.
- *First-applicable*: Rules are evaluated in the order in which they are listed. If a rule's target matches and condition evaluates to *True*, then return the rule's effect (*Permit* or *Deny*). If the target or condition evaluates to *False*, the next rule is evaluated. If no further rule exists, then return *N/A*. If an error occurs, then return *Indeterminate*, with the appropriate error status.

Given a request, a policy set yields one of the six decisions: *Permit*, *Deny*, *N/A*, *I(D)*, *I(P)*, and *I(DP)*. It depends on the policy set target, the decisions of individual policy sets and policies in the policy set, and the policy combining algorithm. XACML 3.0 specifies 12 policy-combing algorithms. Similar to the reasons for the selected rule combining algorithms, this paper focuses on the six policy combining algorithms: *Deny-overrides*, *Deny-unless-permit*, *Permit-overrides*, *Permit-unless-deny*, *First-applicable*, and *Only-one-applicable*.

III. The Conformance Testing Method

Although XACML has a number of language elements, policy and policy set are the first-class executable units. Thus, an executable conformance test case must include a policy (or policy set), an access request, and expected response. The policy (or policy set) and access request are called test input, whereas the expected response is called test oracle. Test oracles of all conformance tests are defined per the XACML3.0 standard specification. When creating conformance tests for other language elements such as policy targets, rule targets, rule conditions, and rules, we also need to create policy files. When a conformance test is executed with Balana, we verify whether the actual response produced by Balana is the same as the test oracle. If not, Balana does not conform to the standard specification and the test is called a non-conformance test.

The main technique used for the conformance testing is decision tables because XACML is essentially a logic-based language. Unlike the traditional mathematical logic that has two truth values (true and false), XACML is more like multi-valued logic due to the consideration of error conditions. For example, a match predicate could evaluate to *True*, *False*, or *Indeterminate* (which means an error has occurred during the evaluation).

We build decision tables for the main XACML language elements based on their semantics described in the standard specification. The decision tables capture the requirements of conformance testing. Concrete conformance tests are then created or generated to cover every entry. Specifically, the tests for targets, rules, and policies are created manually, whereas the tests for policy sets (i.e., sample policy sets and access requests) are generated from the existing tests for policies (i.e., sample policies and access requests).

In the following, we present the test requirements of the main XACML language elements in the form of decision tables. The decision tables are not only useful for the conformance testing, but can help XACML users get an accurate understanding of XACML policies. We start with the basic elements (i.e., targets and rules) and then focus on policies and policy sets.

A. Conformance Testing of Targets

The target in a rule, policy, or policy set consists of zero or more *AnyOf* clauses. An *AnyOf* clause consists of a sequence of *AllOf* clauses. An *AllOf* clause consists of a sequence of match predicates, which are the basic element of targets. A match predicate matches an attribute name with an attribute value. A match predicate evaluates to *True*, *False*, or *Indeterminate* (i.e., an error has occurred during evaluation). A target can evaluate to *Match*, *No Match*, or *Indeterminate*. Table 1 shows the decision table of target evaluation per the XACML3.0 standard specification. *AllOf* is similar to the logical operator "and" – an *AllOf* clause evaluates to *Match* if and only if all match predicates in the *AllOf* clause evaluate to true. It evaluates to indeterminate if one of the match predicate evaluates to *Indeterminate*. *AnyOf* is similar to the logical operator "or". An *AnyOf* clause evaluates to true if one of the *AllOf* clauses evaluate to true.

Table 1 essentially specifies the minimum test requirements for the target element in XACML 3.0. Our test design ensures that each entry in the decision table is covered by at least one conformance test.

TABLE 1. DECISION TABLE FOR TARGET EVALUATION

AnyOf1				AnyOf2				Decision
AllOf1		AllOf2		AllOf3		AllOf4		
Match ch1	Match ch2	Match ch3	Match ch4	Match ch5	Match ch6	Match ch7	Match ch8	
T	T	T	T	T	T	T	T	M
T	T	T	T	T	T	T	I	M
T	T	T	T	T	T	T	F	M
T	T	T	T	T	I	T	I	I
T	T	T	T	T	I	T	F	I
T	T	T	T	T	F	T	F	N
T	T	T	I	T	T	T	T	M
T	T	T	I	T	T	T	I	M
T	T	T	I	T	T	T	F	M
T	T	T	I	T	I	T	I	I
T	T	T	I	T	I	T	F	I
T	T	T	I	T	F	T	F	N
T	T	T	F	T	T	T	T	M
T	T	T	F	T	T	T	F	M
T	T	T	F	T	I	T	I	I
T	T	T	F	T	I	T	F	I
T	T	T	F	T	F	T	F	N
T	I	T	I	T	I	T	I	I
T	I	T	I	T	I	T	F	I
T	I	T	I	T	F	T	F	N
T	I	T	F	T	I	T	F	I
T	I	T	F	T	F	T	F	N
T	F	T	F	T	F	T	F	N

T stand for "True", I stand for "Indeterminate", F stand for "False", M stand for "Match", N stand for "No match".

B. Conformance Testing of Rules

A rule consists of rule target, rule condition, and rule effect (either *Permit* or *Deny*). Rule target (or rule condition) is optional and evaluates to *Match* (or *True*) when it is absent. Rule condition evaluates to *True*, *False*, or *Indeterminate*. Table 2 shows the decision table of rules, where *D/C* refers to "don't care". In the case of rule condition, *D/C* means that the evaluation result of the rule condition is either *True*, *False*, or *Indeterminate*. In the case of rule effect, *D/C* means either *Permit* or *Deny*. For example, when the rule target evaluates to Indeterminate and the rule effect is permit, the rule's decision is *I(P)*, regardless of the evaluation result of the rule condition. When the rule target evaluates to *No Match*, the rule's decision is *N/A* regardless of the rule condition and rule effect. Our test design ensures that each entry is covered by at least one test.

TABLE 2. DECISION TABLE FOR RULE EVALUATION

Evaluation of Rule Target	**Evaluation of Rule Condition**	**Rule Effect**	**Rule Decision**
Match	True	Permit	Permit
Match	True	Deny	Deny
Match	False	D/C	N/A
Match	Indeterminate	Permit	I(P)
Match	Indeterminate	Deny	I(D)
No Match	D/C	D/C	N/A
Indeterminate	D/C	Permit	I(P)
Indeterminate	D/C	Deny	I(D)

C. Conformance Testing of Policies

The main elements of a policy include a policy target, one or more rules, and a rule combining algorithm. Table 3 shows a general decision table about how a policy is evaluated per the XACML3.0 specification. Given a request, if it matches the policy target, then the policy decision depends on the decisions of individual rules and the rule combining algorithm. If the request does not match the policy target, the policy decision is *N/A* regardless of the decisions of individual rules. If the policy target evaluates to *Indeterminate* (i.e., an error has occurred), the policy decision depends on the rule decisions.

The decision tables for rule combining algorithms are created according to the descriptions and pseudo code in the XACML3.0 specification. They are applied when a given

request matches the policy target. Table 4 shows the decision table for the *Deny-overrides* rule combining algorithm. The decision of an individual rule can be *Permit*, *Deny*, *N/A*, *I(D)*, or *I(P)*. Table 4 shows all of the 25 combinations of two rules. In particular, if one rule evaluates to *Deny*, the combined decision is *Deny* - this reflects the meaning of *Deny-overrides*. When there are more than two rules, the combined decision of *n-1* rules can be combined with the *n*-th rule to obtain the policy-level decision.

TABLE 3. DECISION TABLE FOR POLICY EVALUATION

Evaluation Result of Policy Target	**Rule Decisions**	**Policy Decision**
Match	Deny	Specified by the *rule-combining algorithm*
Match	Permit	
Match	N/A	
Match	I(D)	
Match	I(P)	
No Match	D/C	N/A
Indeterminate	Deny	I(D)
Indeterminate	Permit	I(P)
Indeterminate	N/A	N/A
Indeterminate	I(D)	I(D)
Indeterminate	I(P)	I(P)

TABLE 4. DECISION TABLE FOR THE DENY-OVERRIDES RULE-COMBINING ALGORITHM

Deny-overrides		*Decision of the first rule*				
		Permit	**Deny**	**N/A**	**I(D)**	**I(P)**
Decision of the second rule	**Permit**	Permit	Deny	Permit	I(D)	Permit
	Deny	Deny	Deny	Deny	Deny	Deny
	N/A	Permit	Deny	N/A	I(D)	I(P)
	I(D)	I(D)	Deny	I(D)	I(D)	I(DP)
	I(P)	Permit	Deny	I(P)	I(DP)	I(P)

When the name of a rule combining algorithm is also used as a policy combining algorithm, the decision table for the policy combining algorithm (e.g., Table 6) is more general than the decision table of the corresponding rule combining algorithms (e.g., Table 5). Thus, this paper does not present the decision tables of other rule combining algorithms.

Based on the decision tables for the policy evaluation and the rule combining algorithms, we create policy files and request files to cover all entries of each decision table. A policy file and a request file form the input of a conformance test. The test oracle is the corresponding policy decision in the decision table. For an entry of *D/C*, we create a conformance test to cover each possible value of that entry.

To execute the tests automatically, we specify all the conformance tests (policy file, request file, and oracle value) in a spreadsheet. For each entry of the spreadsheet, our test execution framework will invoke Balana with the corresponding policy file and request file, compare the actual response from Balana with the oracle value, and report the verdict (pass/fail).

D. Automated Conformance Testing of Policy Sets

The main elements for a policy set include a policy set target, sub-policies or policy sets, and a policy combining algorithm. The evaluation of a policy set is similar to that of policy evaluation. Table 5 shows the general decision table for policy set evaluation. Given a request, if it matches the policy set target, then the policy set decision depends on the decisions of individual sub-policies/policy sets, and the policy combining algorithm. If the request does not match the policy set target, the policy set decision is *N/A* regardless of the decisions of individual sub-policies/policy sets. If the policy set target evaluates to Indeterminate (i.e., an error has occurred), the policy set decision depends on the decisions of individual sub-policies/policy sets.

TABLE 5. DECISION TABLE FOR POLICY SET EVALUATION

Policy Set Target	**Decisions of Sub-Policies or Policy Sets**	**Policy Set Decision**
Match	Deny	Specified by the *policy-combining algorithm*
Match	Permit	
Match	N/A	
Match	I(D)	
Match	I(P)	
No Match	D/C	N/A
Indeterminate	Deny	I(D)
Indeterminate	Permit	I(P)
Indeterminate	N/A	N/A
Indeterminate	I(D)	I(D)
Indeterminate	I(P)	I(P)
Indeterminate	I(DP)	I(DP)

Tables 6-11 are the decision tables for the six policy combining algorithms. An important feature of this work is that we automatically generate conformance tests for policy sets from the decision tables of the policy combining algorithms and the existing conformance tests for policies.

Let us use the *Deny-overrides* policy combining algorithm as an example. Table 6 is its decision table. We need to create a conformance test for each entry in Table 6 (i.e., a total of 36 tests for Table 6). Consider the entry where the decision of the first policy in the policy set is *Permit* and the decision of the second policy in the policy set is *Deny*. Our goal is to create a policy set file and a request file such that (a) the policy set has two policies, (b) the policy combining algorithm is *Deny-overrides*, (c) the first policy evaluates to *Permit* with respect to the request, and (d) the second policy evaluates to *Deny* with respect to the request. We generate such a policy set file and a request file as follows:

(1) Find an existing policy test including a policy file and a request file such that the policy decision should be *Permit*. Let us denote the policy file as *P1* and the request file as *R1*.

(2) Find an existing policy test including a policy file and a request file such that the policy decision should be *Deny*. Let us denote the policy as *P2* and the request as *R2*.

(3) Find the attribute names in both policy tests. For any attribute that appears in both policy tests, rename the attribute in the second policy test (*P2* and *R2*). Let *P2'* and *R2'* be the revised policy and request.

(4) Generate a policy set file from *P1* and *P2'* using the *Deny-overrides* as the policy combining algorithm. The policy set target is set to empty (which always evaluates to *Match*) or move the target of *P1* or *P2'* to the policy set target.

(5) Generate a request file by composing the attributes and their values in *R1* and *R2'*.

(6) The oracle value of the policy set conformance test is *Deny*, according to the decision table.

Note that the renaming in step (3) is critical. It resolves the naming conflicts – the same attribute from different tests may have different meanings. Without this step, (4) and (5) would not guarantee that the first policy in the policy set evaluates to *Permit* or the second policy in the policy set to *Deny*.

For each policy combining algorithm, the generated conformance tests (including policy set file, request file, and oracle value) are specified in a spreadsheet. For each entry of the spreadsheet, the test execution framework will call Balana with the corresponding policy set file and request file, compare the actual response from Balana with the oracle value, and report the verdict (pass/fail).

TABLE 6. DECISION TABLE FOR THE DENY-OVERRIDE POLICY-COMBINING ALGORITHM

Deny-overrides		*Decision of the first policy or policy set*					
		Permit	**Deny**	**N/A**	**I (D)**	**I (P)**	**I (DP)**
Decision of the second policy or policy set	**Permit**	Permit	Deny	Permit	I (DP)	Permit	I (DP)
	Deny	Deny	Deny	Deny	Deny	Deny	Deny
	N/A	Permit	Deny	N/A	I (D)	I (P)	I (DP)
	I(D)	I (DP)	Deny	I (D)	I (D)	I (DP)	I (DP)
	I(P)	Permit	Deny	I (P)	I (DP)	I (P)	I (DP)
	I(DP)	I (DP)	Deny	I (DP)	I (DP)	I (DP)	I (DP)

TABLE 7. DECISION TABLE FOR THE PERMIT-OVERRIDE POLICY- COMBINING ALGORITHMS

Permit-overrides		*Decision of the first policy or policy set*					
		Permit	**Deny**	**N/A**	**I (D)**	**I (P)**	**I (DP)**
Decision of the second policy or policy set	**Permit**	Permit	Permit	Permit	Permit	Permit	Permit
	Deny	Permit	Deny	Deny	Deny	I (P)	I (DP)
	N/A	Permit	Deny	N/A	I (D)	I (P)	I (DP)
	I(D)	Permit	Deny	I (D)	I (D)	I (DP)	I (DP)
	I(P)	Permit	I (P)	I (P)	I (DP)	I (P)	I (DP)
	I(DP)	Permit	I (DP)	I (DP)	I (DP)	I (DP)	I (DP)

TABLE 8. DECISION TABLE FOR THE DENY-UNLESS-PERMIT POLICY-COMBINING ALGORITHMS

Deny-unless-permit		*Decision of the first policy or policy set*					
		Permit	**Deny**	**N/A**	**I (D)**	**I (P)**	**I (DP)**
Decision of the second policy or policy set	**Permit**	Permit	Permit	Permit	Permit	Permit	Permit
	Deny	Permit	Deny	Deny	Deny	Deny	Deny
	N/A	Permit	Deny	Deny	Deny	Deny	Deny
	I(D)	Permit	Deny	Deny	Deny	Deny	Deny
	I(P)	Permit	Deny	Deny	Deny	Deny	Deny
	I(DP)	Permit	Deny	Deny	Deny	Deny	Deny

TABLE 9. DECISION TABLE FOR THE PERMIT-UNLESS-DENY POLICY-COMBINING ALGORITHMS

Permit-unless-deny	*Decision of the first policy or policy set*					
	Permit	**Deny**	**N/A**	**I (D)**	**I (P)**	**I (DP)**
Decision of the second policy or policy set						
Permit	Permit	Deny	Permit	Permit	Permit	Permit
Deny	Deny	Deny	Deny	Deny	Deny	Deny
N/A	Permit	Deny	Permit	Permit	Permit	Permit
I(D)	Permit	Deny	Permit	Permit	Permit	Permit
I(P)	Permit	Deny	Permit	Permit	Permit	Permit
I(DP)	Permit	Deny	Permit	Permit	Permit	Permit

TABLE 10. DECISION TABLE FOR THE FIRST-APPLICABLE POLICY- COMBINING ALGORITHMS

First-applicable		*Decision of the first policy or policy set*					
		Permit	**Deny**	**N/A**	**I (D)**	**I (P)**	**I (DP)**
Decision of the second policy or policy set	**Permit**	Permit	Deny	Permit	I (D)	I (P)	I (DP)
	Deny	Permit	Deny	Deny	I (D)	I (P)	I (DP)
	N/A	Permit	Deny	NA	I (D)	I (P)	I (DP)
	I(D)	Permit	Deny	I (D)	I (D)	I (P)	I (DP)
	I(P)	Permit	Deny	I (P)	I (D)	I (P)	I (DP)
	I(DP)	Permit	Deny	I (DP)	I (D)	I (P)	I (DP)

TABLE 11. DECISION TABLE FOR THE ONLY-ONE-APPLICABLE POLICY-COMBINING ALGORITHM

Only-one-applicable		*Decision of the first policy or policy set*			
		Permit	**Deny**	**N/A**	**I (Indeterminate)**
Decision of the second policy or policy set	**Permit**	I	I	Permit	I
	Deny	I	I	Deny	I
	N/A	Permit	Deny	N/A	I
	I	I	I	I	I

IV. RESULTS OF CONFORMANCE TESTING

Our conformance testing has revealed several non-conformance cases as summarized below. It is worth pointing out that these cases do not necessarily lead to security violations in an XACML application. It depends on how the responses are handled by the application's policy enforcement point. However, understanding the differences between Balana and the XACML standard specification is important for the users of Balana to correctly enforce access control policies.

The current implementation of the *Permit-overrides* rule and policy combining algorithm does not conform to the XACML3.0 specification with respect to the error conditions. Table 12 shows the three non-conformance tests for which Balana's responses are different from the test oracles per the XACML standard specification (refer to the decision table for *Permit-overrides* in Table 7). When the decisions of two policies in a policy set are *N/A* and *Indeterminate Deny*, or both *Indeterminate Deny*, the decision of the policy set should be *Indeterminate Deny*. However, the actual response of Balana is *N/A*.

TABLE 12. NON-CONFORMANCE TESTS OF THE PERMIT-OVERRIDES POLICY-COMBINING ALGORITHM

Non-Conformance Test	**Test Input**		**Test Oracle per XACML**	**Actual Result by Balana**
	Decision of Policy 1	***Decision of Policy 2***		
1	N/A	I(D)	I(D)	N/A
2	I(D)	N/A	I(D)	N/A
3	I(D)	I(D)	I(D)	N/A

The current implementation of the *Deny-overrides* combining algorithm also has four non-conformance tests as

shown in Table 13. For example, when the decisions of two policies in a policy set are *I(D)* and *Permit*, the decision of the policy set should be *I(DP)*. However, the actual response of Balana is *I(D)*. When the decision of this policy set is the final response to the user, there will be no difference because both *I(DP)* and *I(D)* will result in a plain *Indeterminate*. However, when such a policy set is used by other policy sets, the non-conformance results may lead to different final decisions for access requests.

TABLE 13. NON-CONFORMANCE TESTS OF THE DENY-OVERRIDES POLICY-COMBINING ALGORITHM

Non-Conformance Test	Test Input		Test Oracle per XACML	Actual Result by Balana
	Decision of Policy 1	***Decision of Policy 2***		
1	I(D)	Permit	I(DP)	I(D)
2	I(D)	I(P)	I(DP)	I(D)
3	Permit	I(D)	I(DP)	I(D)
4	I(P)	I(D)	I(DP)	I(D)

In addition, the initial version of Balana used in our project failed the conformance tests of the *Permit-unless-deny* policy-combining algorithm. Examination of the source code revealed that the bugs resulted from the copy-paste of the *Deny-unless-permit* policy-combining algorithm. This has been fixed in the current version of Balana, though.

V. RELATED WORK

Existing work on XACML-related testing has focused on the testing of XACML policies, not the implementation of the XACML standard. Thus, no literature is directly comparable to this paper. In Cirg [8], tests are generated from counterexamples produced by the change-impact analysis of two synthesized versions of an XACML policy. The difference of the two versions of a policy targets a test coverage goal (e.g., rule, or condition). Targen [9] is a test generator for XACML policies that derives access requests to satisfy all the possible combinations of truth values of the attribute id-value pairs found in a given policy. Access requests generated by Cirg and Targen typically use a limited number of subject, resource, action, and environment attributes. A real request, however, could use any combination of attributes. Because requests are encoded in XML, they must conform to the XML Context Schema. To address this issue, Bertolino et al., have developed several test generation algorithms [2][3][4][5]. These algorithms can generate requests that use more than one subject, resource, action, or environment attribute. They can also produce robustness tests, where invalid attribute values are generated randomly. Li et al. have applied symbolic execution technique to generation of access requests for testing XACML policies [7]. They convert the policy under test into semantically equivalent C Code Representation (CCR) and symbolically execute CCR to create test inputs and translate the test inputs to access control requests. Xu et al., have proposed a fault-based testing approach for determining existence or absence of incorrect combining algorithms in XACML 3.0 policies [12].

VI. CONCLUSIONS

We have presented an approach to the conformation testing of Balana, which is currently the only open source implementation of the XACML3.0 standard. Our experiment has revealed subtle conformance issues. The decision tables used to define the conformance test requirements are not only useful for testing XACML3.0 implementations, but also provide important guidelines for understanding the meanings of XACML3.0 language elements. In particular, the various rule combining algorithms and policy combining algorithms have subtle differences and similarities.

ACKNOWLEDGMENT

This work was supported in part by US National Science Foundation (NSF) under grants CNS 1359590 and 1461133. Dr. Yunpeng Zhang and Mr. Ning Shen participated in the initial conformance testing of XACML combining algorithms.

REFERENCES

[1] Balana, "Open source XACML 3.0 implementation," http://xacmlinfo.org/2012/08/16/balana-the-open-source-xacml-3-0-implementation/, 2012.

[2] A. Bertolino, S. Daoudagh, F. Lonetti, and E.marchetti. "Automatic XACML requests generation for policy testing." Fifth IEEE International Conference on Software Testing, Verification and Validation (ICST), 2012, pp.842-849.

[3] A. Bertolino, S. Daoudagh, F. Lonetti, and E.marchetti. "The X-CREATE Framework-A Comparison of XACML Policy Testing Strategies." *Proc. of the 8th International Conference on Web Information Systems and Technologies (WEBIST)*. pp.155-160.

[4] A. Bertolino, S. Daoudagh, F. Lonetti, and E. Marchetti. "Xacmut: Xacml 2.0 mutants generator." Sixth IEEE Int'l Conf. on Software Testing, Verification & Validation Workshops (ICSTW). 2013, pp.28-33.

[5] A. Bertolino, S. Daoudagh, F. Lonetti, E. Marchetti and L. Schilders. "Automated testing of extensible access control markup language-based access control systems." Software, IET 7.4 (2013), pp.203-212.

[6] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnizer, K. Sandlin, R. Miller and K. Scarfone. "Guide to Attribute Based Access Control (ABAC) Definition and Considerations." NIST Special Pub 800 (2014): 162.

[7] Y. C. Li, Y. Li, L. Z. Wang, and G.Chen. "Automatic XACML Requests Generation for Testing Access Control Policies." Proc. of the 26th International Conf. on Software Engineering and Knowledge Engineering (SEKE'14), Vancouver. July 2014.

[8] E. Martin and T. Xie. "Automated test generation for access control policies," in *Supplemental Proc. of ISSRE*, November 2006.

[9] E. Martin, and T. Xie. "Automated test generation for access control policies via change-impact analysis." Proceedings of the Third International Workshop on Software Engineering for Secure Systems. IEEE Computer Society, 2007, pp.5-11.

[10] E. Martin, and T. Xie. "A fault model and mutation testing of access control policies." Proceedings of the 16th International Conference on World Wide Web. ACM, 2007, pp.667-676.

[11] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," http://www.oasisopen.org/committees/xacml/, 2013.

[12] D. Xu, N. Shen, Y. Zhang, "Fault-based testing of combining algorithms in XACML 3.0 policies," Proc. of the 27th Int'l Conf. on Software Engineering and Knowledge Engineering (SEKE'15), 2015.

[13] D. Xu, Y. Zhang, N. Shen, "Formalizing semantic differences between combining algorithms in XACML 3.0 policies," Proc. of the 2015 International Conference on Software Quality, Reliability and Security (QRS'15), pp. 163-172. Vancouver, Canada. August 2015.