Towards a Set-Based Approach for Detecting Flaws in XACML Policies

Hussein Jebbaoui and Azzam Mourad

Department of Computer Science and Mathematics,
Lebanese American University, Beirut, Lebanon
{hussein.jebbaoui, azzam.mourad}@lau.edu.lb

***Abstract*—XACML is a widely used standard for defining and controlling access between Web services and distributed systems. It is the default access control language for Web applications and services. XACML policies are becoming more complex to handle the advancements of web services and applications. However, the size and complexity of XACML policies raise many concerns related to the correctness of complex policies in terms of flaws presence. This paper addresses the aforementioned problem through introducing a set-based schema that provides accurate and efficient analysis of XACML policies. Our approach consists of elaborating an intermediate set-based representation that covers the elements of XACML and a policy analysis module that allows the detection of access flaws in XACML policies. Experiments have been carried out on synthetic and real-file XACML policies for exploring the relevance of our proposition.**

***Index Terms*—Web Services Security; Access Control; Policy Analysis; XACML.**

I. Introduction

The heavy reliance on Web services as the one of the primary methods for data exchange between partners and distributed systems still face the risk of exploitation as a result of infinite accessibility of these services over the Internet [12]. In addition, services with critical data such as banking and other financial businesses are emerging, which increase many security challenges. In this regard, policy-based computing is taking an increasing role in governing the systematic interaction among distributed services. Particularly, access control is the most challenging aspect of Web service security to determine which partner can access which service. Currently, an increasing trend is to declare policies in a standardized specification language such as XACML, the OASIS standard eXtensible Access Control Markup Language [1]. Many vendors are utilizing XACML for access control. It has become the alternative solution to the traditional way of embedding policy verification as part of the application features.

XACML is a mechanism for defining and enforcing access control policies between partners [1]. It has a complex policy structure. It is divided into three layers. The top layer consists of a policy set, the middle layer consists of policies and the lower layer consists of rules. Each of the layers contains a target element which is used to define the subjects, resources and actions. The policy set contains a set of policies, a set of obligations and a policy combining algorithm which is used to break a tie between its policies. Each policy has a set of rules, a set of obligations and a rule combining algorithm which is used to break a tie between its rules. A rule consists of a set of conditions and a rule effect. The obligations at the policy set and policy level are carried out when the final decision is reached to either "permit" or "deny". The policy set example illustrated in Listing 1 demonstrates the policy structure.

The policy set example in Listing 1 contains a policy set PS with one policy P1 and two rules R1 and R2. PS1 has no target defined (lin 3), therefore it is applicable to any request. P1 has a rule combining algorithm "permit-overrides". P1 has no target defined (line 5), therefore it is applicable to any request. Rule R1 (line 6-13) has no target defined and it has a rule condition which permits access to requests with Resource equal to "deposit". Rule R2 (14-27) has no target defined and it has a rule condition which permits access to requests with Resource equal to "deposit" and subject equal to "Joe".

Listing 1: Sample XACML Policy

```
[1]. <?xml version="1.0" encoding="UTF-8"?>
[2]. <PolicySet PolicySetId="PS1" PolicyCombiningAlgId="permit-overrides">
[3].   <Target />
[4].  <Policy PolicyId="P1" RuleCombiningAlgId="permit-overrides">
[5].    <Target />
[6].    <Rule Effect="Permit" RuleId="R1">
[7].      <Condition>
[8].        <Apply FunctionId="function:string-equal">
[9].          <ResourceAttributeDesignator AttributeId="resource:resource-id"
      DataType="string" />
[10].          <AttributeValue DataType="string">deposit</AttributeValue>
[11].        </Apply>
[12].      </Condition>
[13].    </Rule>
[14].    <Rule Effect="Permit" RuleId="R2">
[15].      <Condition>
[16].        <Apply FunctionId="function:and">
[17].          <Apply FunctionId="function:string-equal">
[18].            <ResourceAttributeDesignator AttributeId="resource:resource-id"
      DataType="string" />
[19].            <AttributeValue DataType="string">deposit</AttributeValue>
[20].          </Apply>
[21].          <Apply FunctionId="function:string-equal">
[22].            <SubjectAttributeDesignator AttributeId="subject:subject-id"
      DataType="string" />
[23].            <AttributeValue DataType="string">Joe</AttributeValue>
[24].          </Apply>
[25].        </Apply>
[26].      </Condition>
[27].    </Rule>
[28].  </Policy>
[29].</PolicySet>
```

The complex structure of distributed systems and the need for access control leads to larger and more complex XACML policies. In today's world, mid-size and large companies with complex systems may have policies with thousands of rules. As a result of this complexity, policies used as means of protection can be a source of weaknesses due the presence of flaws and conflicts between rules. In addition, the complexity of XACML structure allows for insertion of possible access flaws among rules and policies. For instance, considering the example in Figure 1, Rules R1 and R2 cause an access flaw because both rules have no targets and R2 is more restricted than R1. Both rules have the same effect ”Permit”. In addition, the generic rule R1 will always take precedence and be evaluated before the restricted rule R2. Therefore the response will be always given by R1 that grants access to any Subject, while R2 that limits the access to Subject ”Joe” will be disregarded. The true objective of access control is to give higher priority to more restricted rules. In this context, the current XACML tools give major role to security administrators for resolving some tie/conflict decisions through policies/rules modifications and/or combining algorithms (e.g. Permit-overrides and First-Applicable). However, manual correction seems practical for small sized policies but it is doubtful for large polices ones. The problem grows when integrating and composing different policies, where contractions between combining algorithms are also risen. In this regard, few approaches have been proposed addressing XACML policy composition and analysis [6], [10], [11], [3], [2], [4], [5]. However, these propositions did not address the access flaw problem.

In this paper, we address the access flaw problem by elaborating a set-based scheme with a formal specification of policies and rules that allows to efficiently perform analysis tasks. The proposed scheme is composed of an initial formal representation, an automatic converter and a policy analysis module embedding a set of algorithms. The set-based syntax maintains the same XACML policy structure respecting its elements and their sub elements. The corresponding converter offers automatic conversion from XACML to set-based constructs. The elaborated algorithms for XACML policy analysis enable to detect access flaws at both policy and rule levels. All the approach components have been implemented in one development framework that accepts XACML policies as inputs, converts them automatically to set based constructs, and produces a list of access flaws that may exist in the policy set. The provided experiments conducted on real-life and synthetic XACML policies explore the relevance of our approach.

The rest of the paper is organized as follows. The related work is summarized in Section II. Section III is devoted for the approach overview and architecture. Section IV illustrates the analysis algorithms. A case study of policy analysis is depicted in Section V. Finally, the conclusion is presented in Section VI.

II. Related Work

In this section, we provide an overview of the related work in the literature addressing XACML policy analysis. Several approaches have been proposed in this regard. Our approach differs from them by providing a set-based algebra representation that accounts for the XACML elements and a policy analysis framework for detecting access flaws in XACML policies, which are not yet addressed by the current propositions.

Kolovski et al. [6] proposed a formalization of XACML using description logics (DL), which are a decidable fragment of First-Order logic. They perform policy verification by using the existing DL verifiers. Their analysis service can discover redundancies at the rule level. A rule is redundant if its decision is always overridden by other rules higher up. This approach may also speed up the evaluation process by removing rules that do not affect the final decision. However, they do not address access flaws and do not support multi-subject requests, complex attribute functions, rule Conditions and Only-One-Applicable combining algorithm.

Fisler et al. [10] proposed a suite called Margrave. It verifies whether an access control policy satisfies a given property and computes the semantic difference of two XACML policies. Margrave can perform a change-impact analysis on the policy to determine the impact of changing one or more rules on the whole policy. However, their proposal does not address policy analysis with respect to access flaws, and does not work on all types of XACML policies.

Tschantz et al. [11] present a set of properties for examining the reasonability of access control policies under enlarged requests, policy growth, and policy decomposition. Their approach focuses on the request and corresponding response behavior under different circumstances and policy reasoning for scalability. However, they do not address policy analysis with respect to access flaws.

Mazzoleni et al. [3] proposed an authorization technique for distributed systems with policies from different parties. Their approach is based first on finding similarities between policies from different parties based on requests. Then, it provides an XACML extension by which a party can provide the integration preferences. This approach focuses on policy integration from different parties and do not address policy analysis for flaws.

Bertino et al. [2] introduced an algebra for fine-grained integration that supports specification of a large variety of integration constraints. They introduced a notion of completeness and prove that their algebra is complete with respect to this notion. Then, they proposed a framework that uses the algebra of fine-grained integration of policies expressed in XACML. Their approach, however, does not cover rule conditions and obligations and focuses on integration between different parties, unlike ours which focuses on analyzing policy sets individually and after integration. Moreover, they mention that there are no guarantees to know if the algebraic expression will hold as expected.

Wijesekera et al. [4] have proposed algebra for manipulating access control policies at a higher level, where the operations of the algebra are abstracted from their specification details. This algebra is motivated by discretionary and role based access control. However, they do not address XACML and do not provide implementation for their algebra.

Bonatti et al. [5] introduced the concept of policy composition under constraints, which aims at combining authorization specifications originating from different independent parties. They proposed algebra for composing access control policies using a variable free authorization terms which are subject, object and action. They suggest logic programming for implementation. However, this approach focuses on policy composition from distributed parties and do not target XACML.

III. Approach Overview

In this section, we present the overall approach illustrated in Figure 1. Our proposition includes the Set-Based, Compiler, and Analysis modules. All the approach components have been implemented in one development framework that accepts XACML policies as inputs, convert them to set-based structure, and perform systematically and automatically all the analysis steps. Using the framework, the user can analyse the policies for security flaws and get the corresponding analysis report using the module embedding the analysis algorithms.

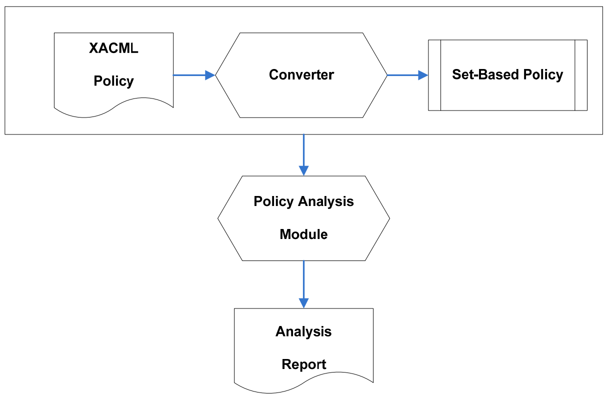

Fig. 1: Approach Architecture

The set-based representation is a formal structure based on algebra sets and composed of the elements and constructs needed for the specification of XACML based policy. A set-based algebra is an accumulation of distinct mathematical elements that describes the fundamental characteristics and includes the regulations of sets and other operations such as union, intersections, complementation, equality and inclusion. It additionally provides systematic procedures for evaluating expressions and performing calculations, involving these operations and relations.

The developed converter includes XACML parser that takes XACML policy set as input, parses its XACML elements and generates set-based constructs. The converter is implemented using PHP (PHP Hypertext Preprocessor). It is embedded in our framework with the policy analysis modules to perform conversion at run time.

The policy analysis module allows to verify policies for detecting access control flaws. It is composed of policy-level and rule-level analysis algorithms. The policy-level algorithm is responsible for analyzing policies and triggers the rule-level one in order to analyze the rules in each policy. The analysis module works effectively if scheduled as a trigger on the repository to run whenever any modification is performed on policies. It is implemented in PHP and accepts a policy set as input.

In this following, we present the constructs, operators and structure of the set-based representation: (PS; P; R; *op* ($\subseteq$; $\subset$; $\wedge$; $>$; $\vee$; $\cap$; $\cup$)), where

- PS : represents a policy set (or based policy) which is composed of one or more policies.
  - P : represents a policy which is composed of one or more rules.
    - R : represents a rule.
- *op* : represents an operator.
  - $\subseteq$: represents a subset or equal.
  - $\subset$: represents a subset.
  - $\wedge$: represents logical operator ”and”.
  - $\vee$: represents logical operator ”or”.
  - $>$: represents the precedence order between operations.
  - $\cap$: represents the intersection between two sets.
  - $\cup$: represents the union of two sets.

An XACML based policy, which they also refer to as a policy set PS, is ordered into 3 levels: $PolicySet$, $Policy$, and $Rule$. Every element can contain a $Target$. $PolicySet$ element contains other $PolicySet(s)$ and/or $Policie(s)$. $Policy$ contains $Rule(s)$. In the sequel, we present the set definitions and syntax of all the elements.

A target is a common element that is used at the policy set, policy and rule levels. A target TR is an objective and is mapped to sets within the context of rule, policy and policy set according to the following syntax:

$$TR = \{S, R, A\}$$

where S is a set of subjects, R is a set of resources and A is a set of actions.

A PolicySet PS is a container of policies. PS may contain other policy sets, policies or both. It can also be referenced by other policy sets. It is mapped to sets according to the following syntax:

$$PS ::=< ID, SP, PR, PCA, IPS, TR >$$

where ID is the policy set id, SP is the set of policies that belongs to policy set PS, PR is the precedence order of policies that belongs to PS, PCA is the policy combining algorithm, IPS is the policies or policy set that are referenced by PS and TR is the target.

a) Example 1: Consider a policy set $PS1$ with two policies $P1$ and $P2$. $PS1$ has a $PCA = deny - overrides$. $PS1$ has a target subject $= Bob$, resource $= FileA$ and action $= Read$. It has no reference to other policies. The policy set $PS1$ is mapped to Set as follows::

$$PS ::=< PS1, \{P1, P2\}, \{P1 > P2\}, \{deny - overrides\}, \{\}, \{\}, \{\{Bob\}, \{FileA\}, \{Read\}\} >$$

A policy P is a single access control policy. It is expressed through a set of rules. A policy contains a set of rules, rule combining algorithm and target. It is mapped to Set according to the following syntax:

$$P ::=< ID, SR, PR, RCA, TR >$$

where ID is the policy id, SR is the set of rules that belongs to policy P, PR is the precedence order of rules that belongs to P, RCA is the rule combining algorithm and TR is the target.

b) Example 2: Consider a policy $P1$ with two rules $R1$ and $R2$. $P1$ has a $rulecombiningalgorithm = permit - overrides$. It has a target $subject = Bob$, $resource = FileA$ and $action = write$. The policy $P1$ is mapped to Set as follows::

$$P ::=< P1, \{R1, R2\}, \{R1 > R2\}, \{permit - overrides\}, \{\}, \{\{Bob\}, \{FileA\}, \{write\}\} >$$

A rule R is the most elementary element of a policy. A rule contains rule conditions, target and rule effect. It is mapped to Set according to the following syntax:

$$R ::=< ID, RC, TR, RE >$$

where ID is the rule id, RC is the set of rule conditions, TR is the target and RE is the rule effect.

A rule condition RC is a boolean function over subjects, resources, actions or functions of attributes. It is mapped to Set within the context of a rule according to the following syntax:

$$RC = \{Apply_function, \{parameters\}\}$$

where $Apply_function$ is the function used in evaluating the elements in the apply and $parameters$ are the inputs to the function being applied.

c) Example 3: Consider a rule $R1$ with $ruleeffect = permit$. $R1$ has no target defined. Its only condition is that anyone accessing $File1$ is allowed at any time. The rule is mapped to Set as follows::

$$R ::=< R1, \{\{string - equal, \{ResourceAttributeDesignator, string, File1\}\}\}, \{\{\}, \{\}, \{\}\}, \{Permit\} >$$

IV. Policy Analysis Algorithms

In this section, we present the algorithms realizing the set-based policy analysis approach. The analysis module is divided into three algorithms: (1) The Rule Analysis algorithm is presented in Algorithm 1, (2) the Policy Analysis algorithm in Algorithm 2 and (3) PolicySet Analysis algorithm in Algorithm 3.

A. Rule Analysis

In this subsection, we present the Rule Analysis algorithm in Algorithm 1. It takes rules *R1* and *R2* as input. It checks for access flaws between *R1* and *R2*. The output is either Flaw or null.

The Rule analysis algorithm in Algorithm 1 takes two rules *R1* and *R2* as input and compares their targets, rule conditions and rule effects to determine if there exists any access flaws between the two rules. It returns the proper response to the Policy Analysis algorithm in Algorithm 2. If the target of rule *R2* is a subset of the target of rule *R1* (line 3), the rule condition set of *R2* is a subset of the rule condition set of

Algorithm 1 Rule_Analysis($R1$, $R2$)

```
Input  : Two Rules R1 with Target TR1 = {S1,R1,A1}, rule condition
         RC1, rule
         effect RE1 and R2 with Target TR2 = {S2,R2,A2}, rule condition
         RC2, rule effect RE2
Output : Rule analysis ∈ {Flaw, Null}
 1: Flaw check if a rule R2 is a subset of R1
 2: // Is R2 target is a subset or equal to R1 target
 3: if ($S2 \subseteq S1$) ∧ ($R2 \subseteq R1$) ∧ ($A2 \subseteq A1$) then
 4:     //Check rule conditions for R1 and R2
 5:     if ($RC2 \subseteq RC1$) then
 6:         // Check if R1 and R2 have the same effect
 7:         if (RE1 = RE2) then
 8:             // R2 is a subset of R1
 9:             return "Flaw";
10:         end if
11:     end if
12: end if
13: return ;
```

R1 (line 5), *R1* and *R2* have the same effect (line 7) and *R1* takes a precedent order over *R2*, then the rule *R1* is considered as access flaw and it should be removed. Otherwise Null is returned, which means R1 and R2 do not cause any access flaw.

B. Policy Analysis

In this subsection, we present the policy analysis algorithm in Algorithm 2. It takes two policies *P1* and *P2* as input. The output is a Flaw set *FS* of all flaws at rule and policy levels.

Algorithm 2 Policy_Analysis($P1$, $P2$)

```
Input  : Policy P1 with Target TR1 = {S1,R1,A1} and P2 with Target
         TR2 = {S2,R2,A2}
Output : Flaw Set FS
 1: //Check rules in each policy
 2: for $l := 1$ to $2$ do
 3:     for $i := 1$ to $P_{lNumberofRules-1}$ do
 4:         for $j := 2$ to $P_{lNumberofRules}$ do
 5:             RA = RULE_ANALYSIS($R_{li}, R_{lj}$);
 6:             // RA response
 7:             if (RA = "Flaw") then
 8:                 FS = FS ∪ $Flaw_{R_{li},R_{lj}}$;
 9:             end if
10:         end for
11:     end for
12: end for
13: //Compare Rule Combining of P1 and P2
14: if ($RCA_{P1} = RCA_{P2}$) then
15:     //Common subjects, Resources and Actions from P1 and P2
16:     if ((($S1 \cap S2$) ≠ ∅) ∧ (($R1 \cap R2$) ≠ ∅) ∧ (($A1 \cap A2$) ≠ ∅)) then
17:         // Check rules for Flaws
18:         for $l := 1$ to $P1_{\_NumberofRules}$ do
19:             for $m := 1$ to $P2_{\_NumberofRules}$ do
20:                 RA = RULE_ANALYSIS($R_l, R_m$);
21:                 // RA response
22:                 if (RA = "Flaw") then
23:                     FS = FS ∪ $Flaw_{R_l,R_m}$;
24:                     FS = FS ∪ $Flaw_{P_1,P_2}$;
25:                 end if
26:             end for
27:         end for
28:     end if
29: end if
30: return ;
```

The first part of the algorithm checks for access flaws within each policy (lines 2-12). It calls the Rule Analysis algorithm in Algorithm 1 (line 5) to check every two rules for flaws. The returned response from the Rule Analysis algorithm is appended to the flaw set FS if "Flaw" is returned from the Rule Analysis (lines 7-9). The second part of the algorithm checks for flaws between rules from different policies if the rule combining of both policies *P1* and *P2* is the same and the targets of both policies *P1* and *P2* intersect (i.e. the subjects of *P1* share common subjects with *P2* subjects, resources of *P1* share common resources with *P2* resources, and actions of *P1* share common actions with *P2* actions (lines 14-29)). It calls the rules analysis on (line 20) to check every two rules *R1* from *P1* and *R2* from *P2* for flaws. The returned response from the Rule Analysis is appended to the flaw set.

C. PolicySet Analysis

In this subsection, we present the PolicySet analysis algorithm in Algorithm 3. It takes a policy set *PS* as input. It calls the Policy Analysis algorithm presented in Algorithm 2 to analyze the policies at the middle layer in the based policy. The output is the analysis report, which contains all access flaws at the policy and rule levels saved in *FS*.

Algorithm 3 PolicySet_Analysis(PS)

```
Input  : A Policy Set PS
Output : Analysis report of all access flaws
1: // Initialize set for Flaws
2: Global FS = $\emptyset$;
3: //Loop through policies in PS
4: for $i := 1\ to\ PS_{NumberofPolices-1}$ do
5:     for $j := i+1\ to\ PS_{NumberofPolices}$ do
6:         // Call Policy Analysis
7:         PA = POLICY_ANALYSIS($P_i, P_j$);
8:     end for
9: end for
```

The PolicySet Analysis algorithm in Algorithm 3 takes a policy set *PS* as input and produces a report of all access flaws between policies and rules. It initializes global set *FS* (line 2) for appending flaws found at both policy and rule levels. It calls the Policy Analysis algorithm in Algorithm 2 (line 7) to check for flaws between every two policies.

V. Case Study & Discussion

In this section, we present a case study illustrating the usability of set-based policy analysis process through experiments and tracing of algorithms. Listing 2 contains the generated set-based policy corresponding to the XACML one in Listing 1 according to the aforementioned set syntax.

Listing 2: Sample Set-XACML Policy

```
[1].PS::=<PS1,{P1},{},{permit-overrides},{},{},{{},{},{}}>
[2].P::=<P1,{R1,R2},{R1>R2},{permit-overrides
    },{},{},{{},{},{}}>
[3].R::=<R1,{string-equal,{ResourceAttributeDesignator,
    string,deposit}},{{},{},{}},{Permit}>
[4].R::=<R2,{{and,{string-equal,{
    ResourceAttributeDesignator,string,deposit}},{string-
    equal,{SubjectAttributeDesignator,subject-id,string,Joe
    }}}},{{},{},{}},{Permit}>
```

Line 1 is the policy set PS. The policy set ID is $PS1$. It has one policy $P1$. The policy combining algorithm is $Permit-Overrides$. $PS1$ has no reference to other policies. Its target subjects, resources and actions are any. Line 2 is the policy $P1$. The policy ID is $P1$. It has two rules $R1$ and $R2$. $R1$ is ordered before $R2$. The rule combining algorithm is $permit-overrides$. $P1$ has no target. Line 3 is the rule $R1$. The rule ID is $R1$. $R1$ has one condition. The condition expresses that the resource must be equal to $deposit$. The target subjects, resources and actions are not specified. $R1$ has a $Permit$ effect. Line 4 is the rule $R2$. The rule ID is $R2$. $R2$ has a set of conditions. The conditions expresses that the subject must be equal to Joe and the resource must be equal to $deposit$. The target subjects, resources and actions are not specified. $R2$ has a $Permit$ effect.

Based on the set-based policy analysis algorithms in Section IV, the elaborated framework analyzes the based policy *PS1* presented in Listings 1 for access flaws. We describe in the following the analysis steps in order based on set operations and reflecting the presented algorithms of policy sets, policies and rules.

PolicySet *PS1* flaw analysis starts here

Step 1. (PS1 Evaluation)
This step requires the analysis of every policy individually to determine the results. *PS1* has one policy *P1*, therefore the analysis starts with *P1*.

Step 1.1. (P1-Analysis)
P1 Analysis starts here.

Step 1.1.1. (P1 Evaluation)
This step requires analysis of rules to determine the results. *P1* has two rules *R1*, and *R2*. The results of P1 depends on evaluating both.

Step 1.1.1.1 (R1,R2-Analysis)
Rules *R1* and *R2* Analysis starts here.

Step 1.1.1.1.1 (R1,R2-Step1 Evaluation)
Both *R1* and *R2* have no targets defined which means TR1 = {{Any},{Any},{Any}} and TR2 = {{Any},{Any},{Any}}. By applying subset set operation, Targets *TR1* and *TR2* are the same therefore *TR2* is subset or equal to *TR1*.

Step 1.1.1.1.2 (R1,R2-Step2 Evaluation)
Rule *R1* has one conditions defined, RC1 = {string-equal,{RAD,string,deposit}}, which means the resource must be equal to deposit and rule *R2* has two conditions, RC2 = {and,{string-equal,{RAD,string,deposit}},{string-equal,{SAD,subject-id,string,Joe}}}, which means that the resource must be equal to deposit and subject must be equal to Joe. *RC2* is a subset of *RC1* because both require the resource to be deposit and *RC2* limits the subject while *RC1* accepts any subject.

Step 1.1.1.1.3 (R1,R2-Step3 Evaluation)
Both *R1* and *R2* have the same effect. All steps are satisfied, hence the returned response for *R1* and *R2*

is $Flaw_{R1,R2}$.

Rules *R1* and *R2* Analysis ends here.

Flaw Set FS = $\{Flaw_{R1,R2}\}$.

Policy *P1* Analysis ends here.

Step 2. (PS1-Step2 Evaluation)

This step requires the analysis of every pair of policies to determine the results. *PS1* has one policy *P1* therefore the analysis stops.

PS1 analysis ends here.

Flaw Set FS = $\{Flaw_{R1,R2}\}$. The result from the set-based analysis above shows that *R1* and *R2* cause an access flaw in the policy set *PS1*.

To explore the relevance of our approach, we performed several experiments on different sizes of real-life and synthetic polices, while inserting several flaws in single and composed policies. The size of the experimented policies reached 400 rules, while the size of the synthetic ones attained 4000 rules. Our results illustrate that our proposed approach and developed modules are able to detect all the inserted flaws.

VI. Conclusion

The significant growth of size and complexity of XACML policies through their composition may lead to insertion of flaws. This paper is targeted this issue through introducing a set-based scheme that allows to perform formal analysis on XACML policies. The proposed approach is composed of an initial formal representation, an automatic converter and a policy analysis module embedding a set of algorithms. The set mathematical representation of policies maintains the same XACML structure and accounts for the needed elements and their sub elements. The corresponding converter offers automatic conversion from XACML to set-based constructs. Finally, the policy analysis module, which embeds the elaborated algorithms, allows detecting access flaws at both policy and rule levels.

Acknowledgment

This work is supported by the Lebanese American University(LAU) and CNRS Lebanon.

References

[1] T. Moses, OASIS eXtensible Access Control Markup Language(XACML), OASIS Standard 2.0. http://www.oasis-open.org/committees/xacml/.

[2] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. An algebra for fine-grained integration of XACML policies. In Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT), pages 6369, 2009.

[3] P. Mazzoleni, E. Bertino, and B. Crispo. Xacml policy integration algorithms. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT), pages 223-232, 2006.

[4] D. Wijesekera and S. Jajodia. A propositional policy algebra for access control. ACM Transactions on Information and System Security (TISS), 6(2):286-325, 2003.

[5] P. Bonatti, S. D. C. D. Vimercati, and P. Samarati. An algebra for composing access control policies. ACM Transactions on Information and System Security (TISS), 5(1):1-35, 2002.

[6] V. Kolovski, J. Hendler, and B. Parsia, Analyzing Web Access Control Policies, Proc. 16th Intl Conf. World Wide Web (WWW 07),pp. 677-686, 2007.

[7] Frank G. Pagan. Formal Specification of Programming Languages. Prentice-Hall, Inc., 1981.

[8] G.D. Plotkin. A structural approach to operational semantics. Logic and Algebraic Programming, 60-61:17-139, 2004.

[9] Kenneth Slonneger and Barry L. Kurtz. Formal Syntax and Semantics of Programming Language: A Laboratory Based Approach. Addison-Wesley Publishing Company, Inc., 1995.

[10] K. Fisler, S. Krishnamurthi, L. Meyerovich, and M. Tschantz. Verification and change impact analysis of access-control policies. In Proc. ICSE, pages 196205, 2005.

[11] M. C. Tschantz and S. Krishnamurthi. Towards reasonability properties for access-control policy languages. In Proc. SACMAT, 2006.

[12] N. Bhalla and S. Kazerooni. (2007, Feb.). Web services vulnerabilities. http://www.blackhat.com/presentations/bh-europe-07/Bhalla-Kazerooni/Whitepaper/bh-eu-07-bhalla-WP.pdf