Computers and Electrical Engineering 44 (2015) 91–103

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Semantics-based approach for detecting flaws, conflicts and redundancies in XACML policies ☆

Hussein Jebbaoui [a], Azzam Mourad [a,*], Hadi Otrok [b], Ramzi Haraty [a]

[a] Department of Computer Science and Mathematics, Lebanese American University, Lebanon
[b] Department of Computer Engineering, Khalifa University of Science, Technology & Research, United Arab Emirates

A R T I C L E I N F O

Article history:
Received 15 May 2014
Received in revised form 4 October 2014
Accepted 16 December 2014
Available online 16 February 2015

Keywords:
Web services security
Access control
Policy analysis
Set theory
Semantics
XACML

A B S T R A C T

XACML (eXtensible Access Control Markup Language) policies, which are widely adopted for defining and controlling dynamic access among Web/cloud services, are becoming more complex in order to handle the significant growth in communication and cooperation between individuals and composed services. However, the large size and complexity of these policies raise many concerns related to their correctness in terms of flaws, conflicts and redundancies presence. This paper addresses this problem through introducing a novel set and semantics based scheme that provides accurate and efficient analysis of XACML policies. First, our approach resolves the complexity of policies by elaborating an intermediate set-based representation to which the elements of XACML are automatically converted. Second, it allows to detect flaws, conflicts and redundancies between rules by offering new mechanisms to analyze the meaning of policy rules through semantics verification by inference rule structure and deductive logic. All the approach components and algorithms realizing the proposed analysis semantics have been implemented in one development framework. Experiments carried out on synthetic and real-life XACML policies explore the relevance of our analysis algorithms with acceptable overhead. Please visit http://www.azzammourad.org/#projects to download the framework.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

The heavy reliance on Web services as one of the primary methods for data exchange between partners and distributed systems still faces the risk of exploitation as a result of their infinite accessibility over the Internet [1,2]. In addition, services with critical data such as banking and other financial businesses are emerging, which increase security challenges [3]. In this regard, policy-based computing [4–6] 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 [7]. Currently, an increasing trend is to declare policies in a standardized specification language such as XACML, the OASIS standard eXtensible Access Control Markup Language [8]. Many vendors are adopting XACML for controlling access to their services.

Before stating the addressed problems and contributions of our work, we depict in the sequel a brief introduction about XACML [8], which has a policy structure 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 layer contains a target element which is used to define the

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. Srinivasan Rajavelu.
* Corresponding author.

http://dx.doi.org/10.1016/j.compeleceng.2014.12.012
0045-7906/© 2014 Elsevier Ltd. All rights reserved.

subjects, resources and actions. The policy set contains a set of policies, a set of obligations and a policy combining algorithm used to break the tie between its policies. Each policy has a set of rules, a set of obligations and a rule combining algorithm used to break the 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 illustrative policy set example in Fig. 1, which will be used and explained in the case study (Section 5), depicts the policy structure.

Nowadays, mid and large size online systems may embed several distributed services heavily interacting and composed together to provide features satisfying the clients' needs. This may require policies with hundreds and even thousands of rules to control access and enforce business behaviors. As a result, policies used as means of protection can be a source of weaknesses due to the presence of flaws and conflicts between their rules. For instance, considering the example in Fig. 1, rules $R3$ and $R4$ lead to an access flaw because both rules have no targets, both rules have the same effect *Permit*, $R3$ precedes in order $R4$ and $R4$ is more restricted than $R3$. With the current XACML decision mechanism, the generic rule $R3$ will always take precedence and be evaluated before the restricted rule $R4$. Therefore the response will always be given by $R3$ that grants access to any subject, while $R4$ that limits the access to subject *Joe* will be disregarded. In this context, the true objective of access control is to give higher priority to more restricted rules. Current XACML tools give major role to security administrators to resolve some tie/conflict decisions through policies/rules modifications and/or combining algorithms (e.g. $Permit - overrides$ and $First - Applicable$). Although manual corrections may seem practical for small size policies, it is doubtful if not impossible for large ones within the complex structure of XACML. The problem grows more when integrating and composing different policies [9,2,7,10,6,5], where contradictions between combining algorithms are apparent. In this regard, some approaches have been proposed addressing XACML policy composition and analysis [11–18]. However, these propositions did not address the presence of access flaws, conflicts and redundancies between policies, and did not consider the logical meaning of rules that reflect the objectives of a policy.

In this paper, we tackle the aforementioned problems by elaborating a set-based scheme that provides formal specification of policies and semantics-based detection built on top of it to efficiently perform analysis tasks. The main contributions of this paper are two folds: (1) Addressing the complex constructs of XACML through an abstract set-based syntax (SBA-XACML), while maintaining a similar policy structure that covers all its elements and sub elements and (2) offering novel detection mechanisms that analyze the meaning of policy rules through semantics verification by inference rule structure and deductive logic. All the approach components and algorithms have been implemented in one development framework that accepts XACML policies as inputs, converts them automatically to SBA-XACML constructs, and produces a list of access flaws, conflicts and redundancies between rules. The provided experiments conducted on real-life and synthetic XACML policies explore the relevance and efficiency of our analysis approach with acceptable overhead.

The rest of the paper is organized as follows. Section 2 covers for the approach overview and architecture. Section 3 presents the semantics rules for policy and rule analysis. Section 4 illustrates the analysis algorithms. Section 5 depicts the case study and semantics-based detection. Section 6 focuses on the experiments and performance analysis. Section 7 summarizes the related work. Finally, Section 8 presents the conclusion.

2. Approach Overview

The overall architecture of our approach is illustrated in Fig. 2 with all its components, i.e. SBA-XACML Language, Compiler and Analysis Module. Using the framework, the user can analyse the policies for access flaws, conflicts and redundancies and get the corresponding analysis report using the module embedding the analysis algorithms.

2.1. SBA-XACML Language and Compiler

SBA-XACML is a set-based language composed of all the elements and constructs needed for the specification of XACML based policy. Please refer to [19] for the complete definition and syntax of SBA-XACML elements and attributes. Its compiler includes XACML parser and converter to SBA-XACML. It takes XACML policy set as inputs, parses their XACML elements and generates SBA-XACML constructs according to the language syntax and structure. In the sequel, we present a brief summary about its constructs that are needed in this paper. SBA-XACML based policy, referred 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 $Rule(s)$.

A target *TR* is an objective and is mapped to SBA-XACML 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.

PS may contain other policy sets, policies or both. It can also be referenced by other policy sets. It is mapped to SBA-XACML according to the following syntax:

$$PS ::= \langle ID, SP, PR, PCA, IPS, OBLs, TR\rangle$$

```
[1] <?xml version="1.0" encoding="UTF-8"?>
[2] <PolicySet  PolicySetId="PS1" PolicyCombiningAlgId="policy-combining-
algorithm:permit-overrides" >
[3]  <Policy PolicyId="P1" RuleCombiningAlgId="rule-combining-algorithm:deny-
overrides">
[4]    <Rule Effect="Permit" RuleId="R1">
[5]      <Condition>
[6]        <Apply FunctionId="function:and">
[7]          <Apply FunctionId="function:string-equal">
[8]            <ResourceAttributeDesignator AttributeId="resource:resource-id"
DataType="xml:string" />
[9]            <AttributeValue DataType="xml:string">
BankService/withdraw</AttributeValue>
[10]           </Apply>
[11]           <Apply FunctionId="function:string-equal">
[12]             <SubjectAttributeDesignator AttributeId="subject:subject-id"
DataType="xml:string" />
[13]             <AttributeValue DataType="xml:string">Bob</AttributeValue>
[14]           </Apply>
[15]         </Apply>
[16]       </Condition>
[17]    </Rule>
[18]    <Rule Effect="Deny" RuleId="R2" />
[19]  </Policy>
[20]  <Policy PolicyId="P2" RuleCombiningAlgId="rule-combining-
algorithm:permit-overrides">
[21]    <Rule Effect="Permit" RuleId="R3" />
[22]    <Condition>
[23]      <Apply FunctionId="function:string-equal">
[24]        <ResourceAttributeDesignator AttributeId="resource:resource-id"
DataType="xml:string" />
[25]        <AttributeValue DataType="xml:string">
BankService/deposit</AttributeValue></Apply>
[26]    </Condition>
[27]    <Rule Effect="Permit" RuleId="R4" />
[28]    <Condition>
[29]      <Apply FunctionId="function:and">
[30]        <Apply FunctionId="function:string-equal">
[31]          <ResourceAttributeDesignator AttributeId="resource:resource-id"
DataType="xml:string" />
[32]          <AttributeValue DataType="xml:string">
BankService/deposit</AttributeValue></Apply>
[33]        <Apply FunctionId="function:string-equal">
[34]          <SubjectAttributeDesignator AttributeId="subject:subject-id"
DataType="xml:string" />
[35]          <AttributeValue DataType="xml:string">Joe</AttributeValue>
[36]        </Apply>
[37]      </Apply>
[38]    </Condition>
[39]    <Rule Effect="Deny" RuleId="R5" />
[40]    <Condition>
[41]    <Apply FunctionId="function:and">
[42]        <Apply FunctionId="function:string-equal">
[43]          <ResourceAttributeDesignator AttributeId="resource:resource-id"
DataType="xml:string" />
[44]          <AttributeValue DataType="xml:string">
BankService/deposit</AttributeValue></Apply>
[45]        <Apply FunctionId="function:string-equal">
[46]          <SubjectAttributeDesignator AttributeId="subject:subject-id"
DataType="xml:string" />
[47]          <AttributeValue DataType="xml:string">Joe</AttributeValue>
[48]       </Apply>
[49]      </Apply>
[50]    </Condition>
[51] </Policy>
[52] </PolicySet>
```

Fig. 1. XACML policy structure.

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, $OBLs$ is the set of obligations and TR is the target.

A policy P contains a set of rules, rule combining algorithm, target and obligations. It is mapped to SBA-XACML according to the following syntax:

$$P ::= \langle ID, SR, PR, RCA, OBLs, TR \rangle$$

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, $OBLs$ is the set of obligations and TR is the target.

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

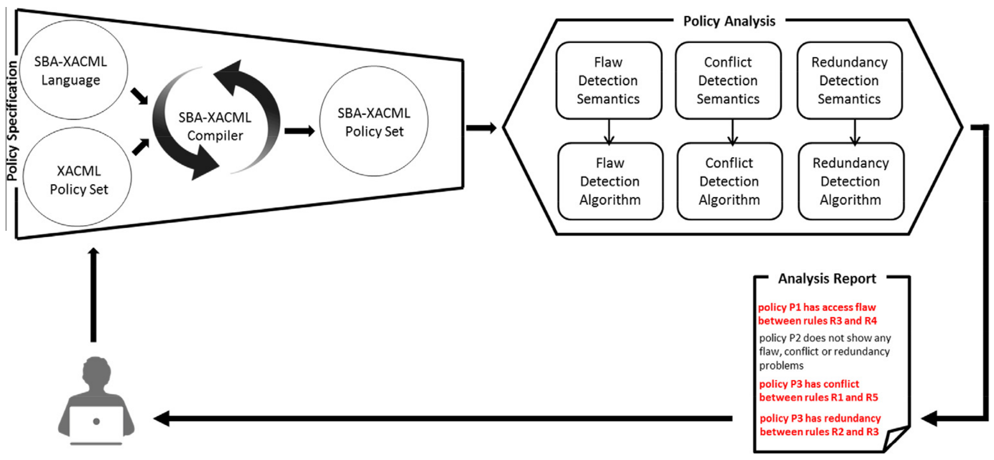

Policy Specification
SBA-XACML Language
XACML Policy Set
SBA-XACML Compiler
SBA-XACML Policy Set
Policy Analysis
Flaw Detection Semantics
Conflict Detection Semantics
Redundancy Detection Semantics
Flaw Detection Algorithm
Conflict Detection Algorithm
Redundancy Detection Algorithm
Analysis Report
policy P1 has access flaw between rules R3 and R4
policy P2 does not show any flaw, conflict or redundancy problems
policy P3 has conflict between rules R1 and R5
policy P3 has redundancy between rules R2 and R3

Fig. 2. Approach architecture.

$$R ::= \langle ID, RC, TR, RE \rangle$$

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 SBA-XACML within the context of a rule according to the following syntax:

$$RC = \{Apply_f unction, \{parameters\}\}$$

where $Apply_f unction$ is the function used in evaluating the elements and *parameters* are the input to the function being applied.

2.2. Policy analysis module

This module allows to analyse policies for detecting access control flaws, conflicts and redundancies between rules. It is composed of policy-level and rule-level analysis algorithms that realize the elaborated analysis semantics presented in Section 3. The policy-level algorithm is responsible for analysing 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 can be scheduled to run in parallel with policy evaluation as well. It accepts a policy set as input and generates an analysis report.

3. Semantics-based analysis

The structural operational semantics used in this paper is an approach proposed to give logical means in defining operational semantics [20,21]. It defines the behavior of a process in terms of the behavior of its parts. Computation is represented by means of deductive logic that turn the abstract machine into a system of logical inferences. This allows to apply formal analysis on the behavior of processes. The behavior of a process is defined in terms of a set of transition relations. Such specifications take the form of inference rules. Definitions are given by inference rules, which consist of a conclusion that follows from a set of premises, possibly under control of some conditions. An inference rule has a general form consisting of the premises listed above a horizontal line, the conclusion below, and the condition, if present, to the right [21].

In this section, we present the formal semantics of SBA-XACML policy analysis following the inference rule structure and deductive logic. Given a policy set *PS*, the analysis report *R* is derived from the evaluation $\underset{PA}{\rightarrow}$ of all premises combined between each other using designated operators *op* as follows:

$$\frac{(premise_1) \quad op \quad (premise_2) \quad op \quad \ldots \quad op \quad (premise_n)}{\langle PS \rangle \underset{PA}{\rightarrow} R}$$

Throughout the rest of the paper, please note the difference between a semantic rule that expresses the analysis at a particular level, and a policy rule which is a construct in SBA-XACML. All the semantics rules follow the bottom up structure, where all the common ones are presented first, then followed by the rule level, policy level and policy set level ones.

3.1. Subset and intersection function

Rule *R*1 has subject set *S*1, resource set *R*1 and action set *A*1. Rule *R*2 has subject set *S*2, resource set *R*2 and action set *A*2.

Rules 1 and 2 in Table 1 describe the different cases of subset rules. In Rule 1, a target *TR*1 is a subset of target *TR*2 if subject set *S*1 is a subset of subject set *S*2, resource set *R*1 is a subset of resource set *R*2, and action set *A*1 is a subset of action set *A*2. In Rule 2, a target *TR*1 is not a subset of target *TR*2 if subject set *S*1 is not a subset of subject set *S*2, or resource set *R*1 is not a subset of resource set *R*2, or action set *A*1 is a not a subset of action set *A*2.

Rules 3 and 4 in Table 2 describe the different cases of intersection rules. In Rule 3, two targets *TR*1 and *TR*2 intersect if subject set *S*1 and subject set *S*2 share common elements, resource set *R*1 and resource set *R*2 share common elements, and action set *A*1 and action set *A*2 share common elements. In Rule 4, two targets *TR*1 and *TR*2 do not intersect if subject set *S*1 and subject set *S*2 share no common elements, resource set *R*1 and target resource set *R*2 share no common elements, or action set *A*1 and action set *A*2 share no common elements.

3.2. Flaw detection

Table 3 presents the analysis semantics rules for detecting access flaws in SBA-XACML policy set. Two rules *R*1 and *R*2 cause access flaw if: (1) Both share the same effect (i.e. access decision) over the same set of subjects, resources and actions, (2) *R*2 is more restricted than *R*1 and (3) *R*1 takes a precedence order over *R*2, i.e. the decision is based on the evaluation of *R*1 while *R*2 is ignored. In other words, the access decision is taken based on the more general rule *R*1 evaluated first, while the more restricted *R*2 is ignored. Accordingly, semantics Rules 5–8 realize the aforementioned logic and describe the different access flaw analysis cases for a policy set *PS*. It is worth noting that such flaws cannot be resolved with the current combining algorithms of XACML.

In Rule 5, two rules *R*1 and *R*2 return flaw if *R*2 target *TR*2 is a subset of rule *R*1 target *TR*1, rule condition *RC*2 is a subset of *R*1 rule condition *RC*1, and both *R*1 and *R*2 have the same rule effect, i.e. *RE*1 is equal to *RE*2. In Rule 6, for every pair of rules *R*1 and *R*2 in policy *P* such that *R*1 and *R*2 are appended to the Flaw Set *FS* if *R*1 and *R*2 evaluate to flaw. In Rule 7, given a pair of policies *P*1, *P*2 in policy set *PS*, *P*1 and *P*2 are appended to the Flaw Set *FS* if the rule combining of *P*1 is equal to the rule combining of *P*2, the targets of *P*1 and *P*2 intersect, and there exists *R*1 in *P*1 and *R*2 in *P*2 such that *R*1 and *R*2 evaluate to flaw and appended to the Flaw Set *FS*. In Rule 8, given policy set *PS*, the Flaw Set *FS* is the union of all flaws between any two flawed rules *R*1 and *R*2 in one policy and between two flawed rules *R*1 and *R*2 from every two flawed policies *P*1 and *P*2.

3.3. Redundancy detection

Table 4 presents the analysis semantics rules for detecting redundancies in SBA-XACML policy set. Two rules *R*1 and *R*2 are considered redundant if both have the same effects (i.e. access decision) over the same set of subjects, resources and actions. Accordingly, semantics Rules 9–12 realize the aforementioned logic and describe the different conflict analysis cases for a policy set *PS*.

In Rule 9, two rules *R*1 and *R*2 are redundant if *R*1 target *TR*1 and *R*2 target *TR*2 intersect, rule condition *RC*1 and rule *R*2 rule condition *RC*2 intersects, and both rules *R*1 and *R*2 have the same rule effect, i.e. *RE*1 is equal to *RE*2. In Rule 10, for every pair of rules *R*1 and *R*2 in policy *P* such that *R*1 and *R*2 are appended to the Redundant Set *RS* if *R*1 and *R*2 are redundant. In Rule 12, given a pair of policies *P*1, *P*2 in policy set *PS*, *P*1 and *P*2 are appended to the Redundant Set *RS* if the rule combining of *P*1 is equal to the rule combining of *P*2, the targets of *P*1 and *P*2 intersect, and there exists *R*1 in *P*1 and *R*2 in *P*2 such that *R*1 and *R*2 are redundant and appended to the Redundant Set *RS*. In Rule 12, given policy set *PS*, the Redundant Set *RS* is the union of all redundancies between any two redundant rules *R*1 and *R*2 in one policy and between two redundant rules *R*1 and *R*2 from every two redundant policies *P*1 and *P*2.

Table 1
Subset function semantics rules.

$\dfrac{(S1 \subseteq S2) \wedge (R1 \subseteq R2) \wedge (A1 \subseteq A2)}{\langle (TR1, TR2) \rangle \underset{subset}{\vdash} True}$	(Rule 1)
$\dfrac{(S1 \not\subset S2) \vee (R1 \not\subset R2) \vee (A1 \not\subset A2)}{\langle (TR1, TR2) \rangle \underset{subset}{\vdash} False}$	(Rule 2)

Table 2
Intersection function semantics rules.

$\dfrac{((S1 \cap S2 \neq \emptyset)) \wedge ((R1 \cap R2 \neq \emptyset)) \wedge ((A1 \cap A2 \neq \emptyset))}{\langle TR1, TR2 \rangle \underset{intersect}{\vdash} True}$	(Rule 3)
$\dfrac{((S1 \cap S2 = \emptyset)) \vee ((R1 \cap R2 = \emptyset)) \vee ((A1 \cap A2 = \emptyset))}{\langle TR1, TR2 \rangle \underset{intersect}{\vdash} False}$	(Rule 4)

Table 3
Rules of access flaw detection semantics.

$\dfrac{\left(\langle TR2, TR1\rangle \underset{subset}{\vdash} True\right) \wedge (RC2 \subseteq RC1) \wedge (RE1 = RE2)}{\langle R1, R2\rangle \underset{R.FA}{\rightarrow} Flaw_{R1,R2}}$	(Rule 5)
$\dfrac{\left(\forall R1, R2 \in SR; FS \leftarrow FS \cup \left(\langle R1, R2\rangle \underset{R.FA}{\rightarrow} Flaw_{R1,R2}\right)\right)}{\langle P\rangle \underset{P.FA}{\rightarrow} FS}$	(Rule 6)
$\dfrac{(RCA.P1 = RCA.P2) \wedge \left(\langle TR1, TR2\rangle \underset{Intersect}{\rightarrow} true\right) \wedge \left(\forall R1 \in SR1, R2 \in SR2; FS \leftarrow FS \cup \left(\langle R1, R2\rangle \underset{R.FA}{\rightarrow} Flaw_{R1,R2}\right)\right)}{\langle P1, P2\rangle \underset{P.F.A}{\rightarrow} FS}$	(Rule 7)
$\dfrac{\left(\forall P \in SP; FS \leftarrow FS \cup \left(\langle P\rangle \underset{P.FA}{\rightarrow} FS\right)\right) \cup \left(\forall P1, P2 \in SP; FS \leftarrow FS \cup \left(\langle P1, P2\rangle \underset{P.FA}{\rightarrow} FS\right)\right)}{\langle PS\rangle \underset{PS.FA}{\rightarrow} FS}$	(Rule 8)

Table 4
Rules of redundancy detection semantics.

$\dfrac{\left(\langle TR2, TR1\rangle \underset{intersect}{\vdash} True\right) \wedge ((RC2 \cap RC1) \neq \emptyset) \wedge (RE1 = RE2)}{\langle R1, R2\rangle \underset{R.RA}{\rightarrow} Redundant_{R1,R2}}$	(Rule 9)
$\dfrac{\left(\forall R1, R2 \in SR; RS \leftarrow RS \cup \left(\langle R1, R2\rangle \underset{R.RA}{\rightarrow} Redundant_{R1,R2}\right)\right)}{\langle P\rangle \underset{P.RA}{\rightarrow} RS}$	(Rule 10)
$\dfrac{(RCA.P1 = RCA.P2) \wedge \left(\langle TR1, TR2\rangle \underset{intersect}{\rightarrow} true\right) \wedge \left(\forall R1 \in SR1, R2 \in SR2;\ RS \leftarrow RS \cup \left(\langle R1, R2\rangle \underset{R.RA}{\rightarrow} Redundant_{R1,R2}\right)\right)}{\langle P1, P2\rangle \underset{P.RA}{\rightarrow} RS}$	(Rule 11)
$\dfrac{\left(\forall P \in SP;\ RS \leftarrow RS \cup \left(\langle P\rangle \underset{P.R.A}{\rightarrow} RS\right) \cup \left(\forall P1, P2 \in SP;\ RS \leftarrow RS \cup \langle P1, P2\rangle \underset{P.RA}{\rightarrow} RS\right)\right)}{\langle PS\rangle \underset{PS.RA}{\rightarrow} RS}$	(Rule 12)

Table 5
Rules of conflict detection semantics.

$\dfrac{\left(\langle TR1, TR2\rangle \underset{intersect}{\vdash} True\right) \wedge ((RC1 \cap RC2) \neq \emptyset) \wedge (RE1 \neq RE2)}{\langle R1, R2\rangle \underset{R.CA}{\rightarrow} Conflict_{R1,R2}}$	(Rule 13)
$\dfrac{\left(\forall R1, R2 \in SR;\ RS \leftarrow CS \cup \left(\langle R1, R2\rangle \underset{R.CA}{\rightarrow} Conflict_{R1,R2}\right)\right)}{\langle P\rangle \underset{P.CA}{\rightarrow} CS}$	(Rule 14)
$\dfrac{(RCA.P1 = RCA.P2) \wedge \left(\langle TR1, TR2\rangle \underset{intersect}{\rightarrow} true\right) \wedge \left(\forall R1 \in SR1, R2 \in SR2;\ CS \leftarrow CS \cup \left(\langle R1, R2\rangle \underset{R.CA}{\rightarrow} Conflict_{R1,R2}\right)\right)}{\langle P1, P2\rangle \underset{P.CA}{\rightarrow} CS}$	(Rule 15)
$\dfrac{\left(\forall P1, P2 \in SP;\ CS \leftarrow CS \cup \left(\langle P1\rangle \underset{P.CA}{\rightarrow} CS\right) \cup \left(\langle P2\rangle \underset{P.CA}{\rightarrow} CS\right) \cup \left(\langle P1, P2\rangle \underset{P.CA}{\rightarrow} CS\right)\right)}{\langle PS\rangle \underset{PS.CA}{\rightarrow} CS}$	(Rule 16)

3.4. Conflict detection

Table 5 presents the analysis semantics rules for detecting conflicts in SBA-XACML policy set. Two rules $R1$ and $R2$ cause conflict if both have opposite effects (i.e. access decision) over the same set of subjects, resources and actions. Accordingly, semantics Rules 13–16 realize the aforementioned logic and describe the different conflict analysis cases for a policy set PS. Once such conflict is detected, the current combining algorithms of XACML can be used to resolve it.

In Rule 14, two rules $R1$ and $R2$ conflict if $R1$ target $TR1$ and $R2$ target $TR2$ intersect, rule condition $RC1$ and $R2$ rule condition $RC2$ intersects and $R1$ with effect $RE1$ is the opposite of $R2$ with effect $RE2$. In Rule *14*, for every pair of rules $R1$ and $R2$ in policy P such that $R1$ and $R2$ are appended to the Conflict Set CS if $R1$ and $R2$ conflict. In Rule 15, given a pair of policies $P1$, $P2$ in policy set PS, $P1$ and $P2$ are appended to the Conflict Set CS if the rule combining of $P1$ is equal to the rule combining of $P2$, the targets of $P1$ and $P2$ intersect, and there exists $R1$ in $P1$ and $R2$ in $P2$ such that $R1$ and $R2$ conflicted and appended to the Conflict Set CS. In Rule 16, given policy set PS, the Conflict Set CS is the union of all conflicts between any two conflicting rules $R1$ and $R2$ in one policy and between two conflicting rules $R1$ and $R2$ from every two conflicting policies $P1$ and $P2$.

4. Policy analysis algorithms

In this section, we present the algorithms realizing the SBA-XACML policy analysis semantics. 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) policy set analysis algorithm in Algorithm 3.

4.1. Rule analysis algorithm

The Rule Analysis Algorithm is presented in Algorithm 1. It takes two rules $R1$ and $R2$ as input and compares their targets, rule conditions and rule effects to determine if there exists any flaw, conflict or redundancy 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 $R1$, the rule condition set of $R2$ is a subset of the rule condition set of $R1$, $R1$ and $R2$ have the same effect and $R1$ takes a precedent order over $R2$, then the rule $R1$ is considered as access control flaw. If the subject set of $R1$ intersects with subject set of $R2$, resource set of $R1$ intersects with resource set of $R2$ and action set of $R1$ intersects with action set of $R2$ and $R1$ and $R2$ have opposite effect, then $R1$ conflicts with $R2$. Otherwise, if $R1$ and $R2$ have the same effect then $R1$ and $R2$ are redundant. Empty set is returned if no issues were found between the two rules.

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, Conflict, Redundant or Null}
 1: if $(S2 \subseteq S1) \wedge (R2 \subseteq R1) \wedge (A2 \subseteq A1)$ then
 2:    if $(RC2 \subseteq RC1)$ then
 3:       if (RE1 = RE2) then
 4:          // R2 is a subset of $R1$
 5:          return "Flaw";
 6:       end if
 7:    end if
 8: end if
 9: if $((S1 \cap S2) \neq \emptyset) \wedge ((R1 \cap R2) \neq \emptyset) \wedge ((A1 \cap A2) \neq \emptyset)$ then
10:    if $((RC1 \cap RC2) \neq \emptyset)$ then
11:       if (RE1 $\neq$ RE2) then
12:          return "Conflict";
13:       else
14:          return "Redundant";
15:       end if
16:    end if
17: end if
18: return;
```

4.2. Policy analysis algorithm

The policy analysis algorithm is presented in Algorithm 2. It takes two policies $P1$ and $P2$ as input and produces a set of all access flaws FS, conflicts CS and redundancies RS. The algorithm is composed of two parts. The first part of the algorithm checks for flaws, conflicts and redundancies within each policy. The second part of the algorithm checks for them between rules from different policies. The returned responses from the Rule Analysis calls are appended to the proper sets.

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, Conflict Set CS and Redundancy Set RS
 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
```

(continued on next page)

```
5:          if (RULE_ANALYSIS ($R_{li}, R_{lj}$) = “Flaw”) then
6:            FS = FS ∪ $Flaw_{R_{li},R_{lj}}$;
7:          end if
8:          if (RULE_ANALYSIS ($R_{li}, R_{lj}$) = “Redundant”) then
9:            RS = RS ∪ $Redundant_{R_{li},R_{lj}}$;
10:         end if
11:         if (RULE_ANALYSIS ($R_{li}, R_{lj}$) = “Conflict”) then
12:           CS = CS ∪ $Conflict_{R_{li},R_{lj}}$;
13:         end if
14:       end for
15:     end for
16: end for
17: //Check rules in both P1 and P2
18: if ($RCA_{P1} = RCA_{P_2}$) then
19:   if $(((S1 \cap S2) \neq \emptyset) \wedge ((R1 \cap R2) \neq \emptyset) \wedge ((A1 \cap A2) \neq \emptyset))$ then
20:     for $l := 1$ to $P1_{\_NumberofRules}$ do
21:       for $m := 1$ to $P2_{\_NumberofRules}$ do
22:         if (RULE_ANALYSIS($R_l, R_m$)= “Flaw”) then
23:           FS = FS ∪ $Flaw_{R_l,R_m}$;
24:           FS = FS ∪ $Flaw_{P_1,P_2}$;
25:         end if
26:         if (RULE_ANALYSIS($R_l, R_m$) = “Conflict”) then
27:           CS = CS ∪ $Conflict_{R_l,R_m}$;
28:           CS = CS ∪ $Conflict_{P_1,P_2}$;
29:         end if
30:         if (RULE_ANALYSIS($R_l, R_m$) = “Redundant”) then
31:           RS = RS ∪ $Redundant_{R_l,R_m}$;
32:           RS = RS ∪ $Redundant_{P_1,P_2}$;
33:         end if
34:       end for
35:     end for
36:   end if
37: end if
38: return;
```

4.3. PolicySet Analysis algorithm

The PolicySet Analysis algorithm is presented in Algorithm 3. It takes a policy set *PS* as input and produces a report of all access flaws, conflicts and redundancies between policies and rules. It initializes global set *FS*, *CS* and *RS* and calls the Policy Analysis algorithm in Algorithm 2 on for checking and appending flaws, conflicts and redundancies at both policy and rule levels, in each policy and between every two policies.

Algorithm 3. PolicySet_Analysis(*PS*)

```
Input: A Policy Set PS
Output: Analysis report
1: Global FS = ∅; RS = ∅; CS = ∅;
2: for $i := 1$ to $PS_{NumberofPolices-1}$ do
3:   for $j := i + 1$ to $PS_{NumberofPolices}$ do
4:     PA = POLICY_ANALYSIS($P_i, P_j$);
5:   end for
6: end for
```

5. Case study: Semantics-based policy analysis

In this section, we present a case study illustrating the practicality of SBA-XACML policy analysis process through semantics rules. Listing 1 contains the generated SBA-XACML based policy corresponding to the XACML policy example presented in Fig. 1.

Line 1 is the policy set *PS*. The policy set *ID* is *PS*1. It has two policies *P*1 and *P*2. *P*1 is ordered before *P*2. The policy combining algorithm is *permit – overrides*. *PS*1 has no reference to other policies. It has no obligations to perform and the target subjects, resources and actions are any. Line 2 is the policy *P*1. The policy *ID* is *P*1. It has two rules *R*1 and *R*2. *R*1 is ordered before *R*2. The rule combining algorithm is *deny – overrides*. *P*1 has no obligations and no target. Line 3 is the rule *R*1. The rule *ID* is *R*1. *R*1 has a set of conditions. The conditions are: the subject *ID* must be equal to *Bob* and the resource *ID* must be equal to *BankService/Withdraw*. The target subjects, resources and actions are any. *R*1 has a *permit* effect. Line 4 is the rule *R*2. The rule *ID* is *R*2. *R*2 has no conditions. *R*2 has no target specified. *R*2 has a *deny* effect. Line 5 is the policy *P*2. The policy *ID* is *P*2. It has three rules *R*3, *R*4 and *R*5. The precedence order is *R*3, *R*4 then *R*5. The rule combining algorithm is *permit – overrides*. *P*2 has no obligation to perform and the target elements are not defined. Line 6 is the rule *R*3. The rule *ID* is *R*3. *R*3 has one condition. The condition states that the resource *ID* must be equal to *BankService/Deposit*. The target subjects, resources and actions are not specified. *R*3 has a *permit* effect. Line 7 is the rule *R*4. The rule *ID* is *R*4. *R*4 has a set of conditions. The conditions are: the subject *ID* must be equal to *Joe* and the resource *ID* must be equal to

SBA-XACML Policy for a Bank Service

```
[1].PS::=<PS1,{P1,P2},{P1>P2},{Permit-overrides},{},{},{{},{},{}}>
[2].P::=<P1,{R1,R2},{R1>R2},{Deny-overrides},{},{},{{},{},{}}>
[3].R::=<R1,{{and,{string-equal,{ResourceAttributeDesignator,string,
     BankService/withdraw}},{string-equal,{SubjectAttributeDesignator,subject
     -id,string,Bob}}}},{{},{},{}},{Permit}>
[4].R::=<R2,{},{{},{},{}},{Deny}>
[5].P::=<P2,{R3,R4,R5},{R3>R4>R5},{Permit-overrides},{},{},{{},{},{}}>
[6].R::=<R3,{string-equal,{RAD,string,BankService/deposit}},{{},{},{}},{
     Permit}>
[7].R::=<R4,{{and,{string-equal,{RAD,string,BankService/deposit}},{string-
     equal,{SubjectAttributeDesignator,subject-id,string,Joe}}}},{{},{},{}},{
     Permit}>
[8].R::=<R5,{{and,{string-equal,{RAD,string,BankService/deposit}},{string-
     equal,{SubjectAttributeDesignator,subject-id,string,Joe}}}},{{},{},{}},{
     Deny}>
```

Listing 1. SBA-XACML policy for a bank service.

$$\cfrac{\cfrac{\cfrac{\cfrac{(\{Any\} \subseteq \{Any\}) \wedge (\{Any\} \subseteq \{Any\}) \wedge (\{Any\} \subseteq \{Any\})}{(S2 \subseteq S1) \wedge (R2 \subseteq R1) \wedge (A2 \subseteq A1)}}{(<TR2,TR1> \vdash_{subset} True)} \quad \cfrac{\text{False}}{(RC2 \subseteq RC1)} \quad \cfrac{\text{permit} \neq \text{deny}}{(RE1 = RE2)}}{(\exists\ R1,R2 \in SR; <R1,R2> \xrightarrow[R.FA]{} null)}}{(\exists\ P1 \in SP; <P1> \xrightarrow[P.FA]{} null)} \quad \mathbf{(2)}$$

$$\cfrac{\cfrac{\cfrac{\cfrac{(\{Any\} \subseteq \{Any\}) \wedge (\{Any\} \subseteq \{Any\}) \wedge (\{Any\} \subseteq \{Any\})}{(S4 \subseteq S3) \wedge (R4 \subseteq R3) \wedge (A4 \subseteq A3)}}{(<TR4,TR3> \vdash_{subset} True)} \quad \cfrac{\text{True}}{(RC4 \subseteq RC3)} \quad \cfrac{\text{permit}=\text{permit}}{(RE3 = RE4)}}{(\exists\ R3,R4 \in SR; <R3,R4> \xrightarrow[R.FA]{} Flaw_{R3,R4})}}{(\exists\ P2 \in SP; <P2> \xrightarrow[P.FA]{} Flaw_{R3,R4})} \quad \mathbf{(3)}$$

•

•

•

$$\frac{RCA.P1 \neq RCA.P2}{(\exists\ P1,P2 \in SP; <P1,P2> \xrightarrow[P.FA]{} null)} \quad \mathbf{(4)}$$

$$\frac{\mathbf{(2)}(\exists\ P1 \in SP; <P1> \xrightarrow[P.FA]{} null) \quad \mathbf{(3)}(\exists\ P1,P2 \in SP; <P1,P2> \xrightarrow[P.FA]{} null) \quad \mathbf{(4)}(\exists\ P2 \in SP; <P2> \xrightarrow[P.FA]{} Flaw_{R3,R4})}{<PS1> \xrightarrow[PS.FA]{} Flaw_{R3,R4}} \quad \mathbf{(1)}$$

Fig. 3. Flaw detection analysis.

BankService/*Deposit*. The target subjects, resources and actions are not specified. *R*4 has a *permit* effect. Line 8 is the rule *R*5. The rule *ID* is *R*5. *R*5 has a set of conditions. The conditions are: the subject *ID* must be equal to *Joe* and the resource *ID* must be equal to *BankService*/*Deposit*. The target subjects, resources and actions are not specified. *R*5 has a *deny* effect.

Based on the SBA-XACML policy analysis semantics in Section 3, the elaborated framework will analyze the based policy *PS*1 presented in Listing 1 for detecting access flaws, conflicts and redundancies. The result of the analysis shows that *PS*1 has $Flaw_{R3,R4}$, $Conflict_{R3,R5}$ and $Redundant_{R3,R4}$. To avoid repetition and for space limitation, we will only present the access flaws detection and three different cases where flaws exist between two rules, no flaws between two rules and no flaws between two policies in PS1. The analysis for conflict and redundancy detection is performed in similar way. The analysis of each semantics rule in Fig. 3 is based on analyzing its premises, hence they should be read from bottom up, i.e at the level of policy sets, policies and then rules as follows:

(1) The based policy is composed of a PolicySet *PS*1. *PS*1 contains a flaw because there exists a policy *P*2 in *PS*1 such that *P*2 contains a flaw between rules *R*3 and *R*4 as depicted in **(3)**. Hence, based on the semantics **(Rule 8)** that applies in this case and combines the results of all the premises, the final response is the flaw set *FS* containing $Flaw_{R3,R4}$.

(2) Policy *P*1 is composed of two rules *R*1 and *R*2, which cause no flaw by applying semantics **(Rule 5)**. First, *Premise*1 evaluates to *True* because *R*1 and *R*2 have no targets defined, which means *TR*1 = {S = Any, R = Any, A = Any} and *TR*2 = {S = Any, R = Any, A = Any}. Second, *R*2 rule conditions *RC*2 is not a subset of *R*1 rule conditions *RC*1 since *R*2 has no condition, while *R*1 has two conditions that limit the access to resource equals to *BankService*/*Withdraw* and subject equals to *Joe*. Hence, *Premise*2 evaluates to *False*. This fact denies the presence of a flaw, where the decision is based on the restricted rule that precedes the general one. Hence, the response is *null* and there is no need to continue checking the remaining premises.

(3) Policy *P*2 is composed of three rules *R*3, *R*4 and *R*5. Rules *R*3 and *R*4 cause flaw by applying semantics **(Rule 5)**. First, *Premise*1 evaluates to *True* because *R*3 and *R*4 have no targets defined, which means *TR3* = {S = Any, R = Any, A = Any} and *TR4* = {S = Any, R = Any, A = Any}. Second, *R*4 rule conditions *RC*4 is a subset of *R*3 rule conditions *RC*3 since both rules require the resource to be *BankService*/*deposit*, while *R*4 limits the subject to *Joe*. Hence, *Premise*2 evaluates to *True*. This fact constitutes the first sign for a flaw, where the decision is based on the general rule that precedes the restricted one. Third, *Premise*3 evaluates to *True* because both rules *R*3 and *R*4 have the same effect *Permit*. All premises of semantics **(Rule 5)** evaluate to true, therefore policy *P*2 has access flaw between rules *R*3 and *R*4. The same analysis steps will be applied between *R*3 and *R*5 and between *R*4 and *R*5, without any existence of flaws. We did not include them to avoid repetition. Hence, the response is the flaw set *FS* containing $Flaw_{R3,R4}$.

(4) Policies *P*1 and *P*2 do not cause flaw. By Applying semantics **(Rule 7)**, *Premise*1 evaluates to *False* because the rule combining algorithm *RCA*1 of *P*1 is equal to $deny-overrides$, which is different than the rule combining algorithm *RCA*2 of *P*2, which is equal to $permit-overrides$. Hence, the response is *null* and there is no need to continue checking the remaining premises and analysing the rules of *P*1 and *P*2.

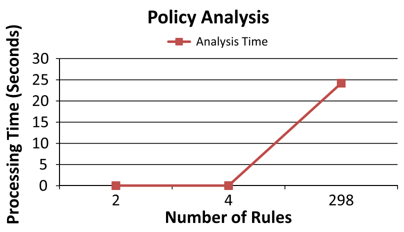
6. Discussion and experimental results

In this section, we examine the results of our experiments for analyzing policies for access flaws, conflicts and redundancies. The SBA-XACML framework is implemented in PHP. The experiments were carried out on a notebook running Windows XP SP3 with 3.50 GB of memory and dual core 2.8 GHz Intel processor. The tests were conducted on both real world and synthetic policies to show the scalability and performance whether small or large. The real policies utilized in the experiments are small and mid-sized ones ranging between 2 and 298 rules. The synthetic policies are small and large ranging between 400 and 4000 rules. The flawed, conflicted and redundant rules were injected at random with different rate from 1 to 5 per every 10 rules. This process has been repeated hundreds of times with both real and synthetic policies.

The first set of experiments has been performed to assess the detection rate of our approach. At each trial, extensive testing has been performed in order to make sure that the proposed detection mechanisms are able to successfully detect all the

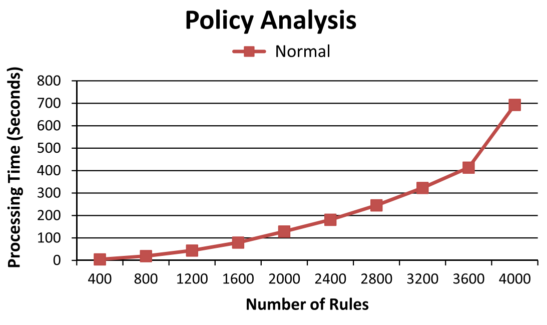
Policy	Number of Rules	Analysis Time (Seconds)
III027	2	0.03
III028	4	0.04
Continue-a	298	24.2

(a) Real Policy Analysis Statistics

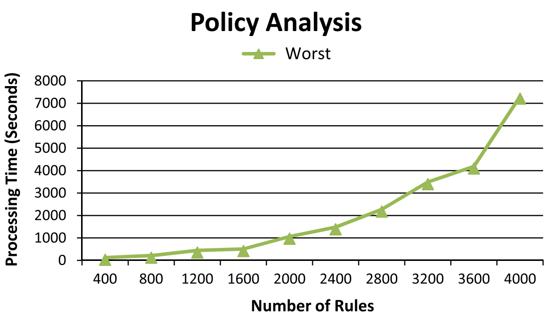


(b) Real Policy Results Analysis

Fig. 4. Real policy analysis results.



(a) Synthetic Normal Case



(b) Synthetic Worst Case

Fig. 5. Synthetic policy analysis results.

injected flaws, conflicts and redundancies. The results of our experiments gave 100% detection rate. The second set of experiments were conducted to assess the performance and efficiency of the analysis process. The processing time is the average time calculated based on 100 K run per every policy set. Our analysis experiments were performed based on two scales, the normal case and the worst case. The normal case considers having policies injected with flaws, conflicts and redundancies up to 10%, while the worst case injected up to 50%. The analysis processing time is static and includes the conversion of the policy set from XACML to SBA-XACML, which is optional and executed only once when deploying policies.

Fig. 4a and b show the experimental results for analyzing real life policies utilizing the normal scale, with injecting flaws, conflicts and redundancies up to 10%. Policies with 4 rules require less than 40 m-s to complete the analysis process, while policies with 298 rules require on average about 24.2 s to complete.

Fig. 5a presents the statistics for analyzing synthetic policies utilizing two different scales, the normal and the worst. Fig. 5a shows the processing time when 1 out 10 rules/policies cause access flaws, conflicts or redundancies. At 400 rules, the analysis processing time consumes 5.54 s to complete. At 1200 rules, it consumes 44 s. And at 4000 rules, it consumes 692 s.

Fig. 5b shows the processing time when five out of ten rules/policies cause access flaws, conflicts or redundancies. The policies are designed in such a way that not only rules within each policy are verified, but also rules from different policies are verified as well due to similarities between policy targets and rule combining algorithms. The statistics show that policies with 400 rules can be analyzed with 29 s, 1200 rules with 352 s and 4000 rules with 7210 s. The worst case processing times seem reasonable considering the policy size and the percentage of flaws, conflicts and redundancies injected. However, such cases are very unlikely to incur in real world policies. The results of these experiments explore the efficiency in terms of performance for reasonable size policies. The addition overhead is affordable, even though such analysis is performed most of the time offline.

7. Related work

In this section, we provide an overview of the related work in the literature addressing XACML policy analysis, in addition to some policy evaluation approaches. In this regard, Kolovski et al. [11] 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. 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. [13] 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. However, their proposal does not address policy analysis with respect to access flaws, and does not work on all types of XACML policies. Tschantz and Krishnamurthi [14] present a set of properties for examining the reasonability of access control policies under enlarged requests, policy growth, and policy decomposition. However, they do not address policy analysis with respect to access flaws. Mazzoleni et al. [15] 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. This approach focuses on policy integration from different parties and do not address policy analysis for flaws.

Rao et al. [16] introduced an algebra for fine-grained integration that supports specification of a large variety of integration constraints. They introduced a notion of completeness and proved that their algebra is complete with respect to this notion. 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. Wijesekera and Jajodia[17] have proposed algebra for manipulating access control policies at a higher level, where the operations of the algebra are abstracted from their specification details. However, they do not address XACML and do not provide implementation for their algebra. Bonatti et al. [18] introduced the concept of policy composition under constraints and proposed algebra for composing access control policies using a variable free authorization terms such as subject, object and action. However, this approach focuses on policy composition from distributed parties and do not target XACML.

In another context related to policy evaluation for efficient decision process, few approaches have been proposed such [22–25]. Based on the study of the current literature, it is trivial that both domains are still and will continue to be a challenging niche for researchers. To the best of our knowledge, none of the current approaches did address the presence of access flaws, conflicts and redundancies between policies, and did consider the logical meaning of rules that reflect the objectives of a policy. In this regard, our approach differs by providing a novel detection mechanisms that study the meaning of policy rules through semantics verification by inference rule structure and deductive logic.

8. Conclusion and future work

The proposed approach addressed the problem of flaws, conflicts and redundancies presence between the rules of large-size and complex XACML policies. In this context, the contribution of this work is the elaboration of a set-based scheme that provides formal specification of XACML policies and semantics-based detection built on top of it to efficiently perform analysis tasks. Our approach improves the related literature in two different aspects. First, it offers an abstract set-based language that addresses the complex constructs of XACML while maintaining similar policy structure that covers all its constructs. Second, it embeds a novel detection mechanism that analyzes the meaning of policy rules through semantics verification by inference rule structure and deductive logic. The aforementioned theoretical outcomes were realized by developing practical algorithms embedded into a framework modules. The performed experiments on real-life and synthetic policies illustrate the relevance and efficiency of our approach for detecting flaws, conflicts and redundancies within acceptable overhead. Moreover, the step-by-step policy analysis depicts the applicability of the semantics-rules to identify in each of the aforementioned cases the contradictions between policies/rules. Please visit http://www.azzammourad.org/#projects to download the framework.

For future work, we can benefit from SBA-XACML to potentially elaborate policy analysis semantics based on the meaning of rules for detecting other types of flaws and identifying access contradictions that can influence the grouping decision between different Web services.

Acknowledgment

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

References

[1] Bhalla N, Kazerooni S. Web services vulnerabilities, 2007. <http://www.blackhat.com/presentations/bh-europe-07/Bhalla-Kazerooni/Whitepaper/bh-eu-07-bhalla-WP.pdf>.
[2] Mourad A, Ayoubi S, Yahyaoui H, Otrok H. New approach for the dynamic enforcement of Web Services Security. In: Proceedings of the eighth annual conference on privacy, security and trust (PST 2010), 2010. p. 189–96.
[3] Atkinson B, et al. Web services security (WS-Security), 2006. <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss>.
[4] Yahyaoui H, Mourad A, AlMulla M, Yao L, Sheng QZ. A synergy between context-aware and AOP to achieve highly adaptable Web services. J Serv Orient Comput 2012;6(4):379–92.
[5] Tout H, Mourad A, Otrok H. XrML-RBLicensing approach adopted to the BPEL process of composite web services. J Serv Orient Comput 2013;7(3):217–30.
[6] Ayoubi S, Mourad A, Otrok H, Shahin A. New XACML-AspectBPEL approach for composite web services security. Int J Web Grid Serv 2013;9(2):127–45.
[7] Mourad A, Ayoubi S, Yahyaoui H, Otrok H. A novel aspect-oriented BPEL framework for the dynamic enforcement of web services security. Int J Web Grid Serv 2012;8(4):361–85.
[8] Moses T. OASIS eXtensible Access Control Markup Language(XACML), OASIS Standard 2.0., 2011. <http://www.oasis-open.org/committees/xacml/>.
[9] Karakoc E, Senkul P. Composing semantic web services under constraints. Expert Syst Appl 2009;36(8):11021–9.
[10] Mizouni R, Abdel Serhani M, Dssouli R, Benharref A, Taleb I. Performance evaluation of mobile web services. In: Proceedings of the 9th IEEE European conference on web services (ECOWS 2011), 2011. p. 184–91.
[11] Kolovski V, Hendler J, Parsia B. Analyzing web access control policies. In: Proceedings of the 16th international conference on world wide web (WWW '07), 2007. p. 677–86.
[12] Li N, Hwang J, Xie J. Multiple-implementation testing for XACML implementations. In: Proceedings of the 2008 workshop on testing, analysis, and verification of web services and applications, 2008. p. 27–33.
[13] Fisler K, Krishnamurthi S, Meyerovich L, Tschantz M. Verification and change impact analysis of access-control policies. In: Proceedings of 27th international conference on software engineering (ICSE), 2005. p. 196–205.
[14] Tschantz M, Krishnamurthi S. Towards reasonability properties for access-control policy languages. In: Proceedings of the eleventh ACM symposium on Access control models and technologies (SACMAT2006), 2006. p. 160–69.
[15] Mazzoleni P, Bertino E, Crispo B. XACML policy integration algorithms: not to be confused with XACML policy combination algorithms!. In: Proceedings of the 11th ACM symposium on access control models and technologies (SACMAT2006), 2006. p. 219–27.
[16] Rao P, Lin D, Bertino E, Li N, Lobo J. An algebra for fine-grained integration of XACML policies. In: Proceedings of the 14th ACM symposium on access control models and technologies (SACMAT2009), 2009. p. 63–9.
[17] Wijesekera D, Jajodia S. A propositional policy algebra for access control. ACM Trans Inform Syst Secur (TISS) 2003;6(2):286–325.
[18] Bonatti P, Vimercati SDCD, Samarati P. An algebra for composing access control policies. ACM Trans Inform Syst Secur (TISS) 2002;5(1):1–35.
[19] Mourad A, Jebbaoui H. SBA-XACML: set-based approach providing efficient policy decision process for accessing web services. J Experts Syst Appl 2014;42(1):165–78.
[20] Plotkin GD. A structural approach to operational semantics. J Logic Algebr Program 2004:17–139.
[21] Slonneger K, Kurtz BL. Formal syntax and semantics of programming language: a laboratory based approach. Springer; 1995.

[22] Liu AX, Chen F, Hwang J, Xie T. XEngine: a fast and scalable XACML policy evaluation engine. In: Proceedings of the SIGMETRICS international conference on measurement and modeling of computer systems, 2008. p. 265–76.
[23] Marouf S, Shehab M, Squicciarini A, Sundareswaran S. Adaptive reordering and clustering based framework for efficient XACML policy evaluation. IEEE Trans Serv Comput 2011;4(4):300–13.
[24] Pina Ros S, Lischka M, Gómez Mármol F. Graph-based XACML evaluation. In: Proceedings of the 17th ACM symposium on access control models and technologies (SACMAT12), 2012. pp. 83–92.
[25] Ngo C, Makkes M, Demchenko Y, de Laat C. Multi-data-types interval decision diagrams for XACML evaluation engine. In: Proceedings of the 11th international conference on privacy, security and trust (PST 2013), 2013. p. 257–66.

Hussein Jebbaoui received his M.Sc. degree in Computer Science from the Lebanese American University. The topics of his research activities are Web services security and XACML policy evaluatin and analysis.

Azzam Mourad is an assistant professor of Computer Science at the Lebanese American University. He holds a Ph.D. in ECE from Concordia University and M.Sc. degree in Computer Science from Laval University. He is currently working on information security, web services, vehicular networks, and formal semantics. He is serving as TPC and reviewers of several prestigious conferences and journals.

Hadi Otrok holds an associate professor position in the Department of ECE at Khalifa University. He received his Ph.D. in ECE from Concordia University. He works on network and computer security, game theory and mechanism design. He chaired several security-related conferences. Moreover, he is a TPC member of several prestigious conferences and reviewer of several IEEE and Elsevier journals.

Ramzi A. Haraty is an associate professor in the Department of Computer Science and Mathematics at the Lebanese American University. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 110 books, book chapters, and journal and conference paper publications.