LEBANESE AMERICAN UNIVERSITY

SET-BASED APPROACH FOR EFFICIENT EVALUATION AND ANALYSIS OF XACML POLICIES

By

HUSSEIN M. JEBBAOUI

A thesis
Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

School of Arts and Sciences
May 2014
THESIS APPROVAL FORM

Student Name: ________________________ Hussein Jebbaoui ________________________ I.D. #: ________________________ 201003548

Thesis Title: SBA-XACML: Set-based Approach for Efficient Evaluation and Analysis of XACML Policies

Program: Masters in Computer Science
Department: Computer Science and Mathematics

School: Arts and Sciences

The undersigned certify that they have examined the final electronic copy of this thesis and approved it in Partial Fulfillment of the requirements for the degree of:

Masters of Science in the major of Computer Science

Thesis Advisor’s Name: ________________________ Dr. Azzam Mourad ________________________ Signature ________________________ Date: 05/05/14

Committee Member’s Name: ________________________ Dr. Faisal Abu-Khzam ________________________ Signature ________________________ Date: 05/05/14

Committee Member’s Name: ________________________ Dr. Samer Habre ________________________ Signature ________________________ Date: 05/05/14
THESIS COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY NON-EXCLUSIVE DISTRIBUTION LICENSE

By signing and submitting this license, you (the author(s) or copyright owner) grants to Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. You agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. You also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Name: Hussein Jebbaoui

Signature: 

Date: Apr. 5, 2014
PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that:

- I have read and understood LAU’s Plagiarism Policy.
- I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.
- This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Hussein Jebbaoui

Signature: [Signature]

Date: Apr. 5, 2014
Dedication

To my family
ACKNOWLEDGMENTS

First of all, I would like to thank God for giving me the opportunity to complete this work successfully. I would like to express my gratitude and appreciation to my advisor, Dr. Azzam Mourad, for his excellent support and caring. It was a great privilege and honor to have worked under his guidance and provision. I would also like to thank him for his friendship and unconditional support.

I also would like to thank my committee members Dr. Samer Habre and Dr. Faisal Abu-Khzam for their time and efforts serving on the defense committee.

Most importantly, I would like to thank my family. They have been there for me through the years of my life, and their love, encouragement and support have made all the difference.
Set-Based Approach for XACML Evaluation and Analysis

Hussein Jebbaoui

ABSTRACT

Policy-based computing is taking an increasing role in governing the systematic interaction among distributed cloud and Web services. XACML has been known as the de facto standard widely used by many vendors for specifying access control policies. Accordingly, the size and complexity of XACML policies are significantly growing to cope with the evolution of web-based applications. This growth raised many concerns related to the efficiency of real-time decision process (i.e. policy evaluation) and the correctness of complex policies. This thesis is addressing both concerns through the elaboration of SBA-XACML, a novel set-based scheme that provides efficient evaluation and analysis of XACML policies. To the best of our knowledge, we are the first addressing both problems simultaneously. Our approach constitutes of elaborating (1) set-based language that covers all the XACML components and establish an intermediate layer to which policies are automatically converted, (2) policy evaluation module that provides better performance compared to the industrial standard Sun Policy Decision Point (PDP) and its corresponding ameliorations, and (3) policy analysis module that allows to detect flaws, conflicts and redundancies in XACML policies. Formal and practical experiments have been conducted on real-life and synthetic XACML policies in order to demonstrate the efficiency, relevance and scalability of our proposition. The experimental results explore that SBA-XACML evaluation of large and small sizes policies offers better performance than the current approaches, by a factor ranging between 2.4 and 15 times faster depending on policy size. Moreover, they show how SBA-XACML analysis module allows detecting access flaws, conflict and redundancy at policy and rule levels.

Keywords: Web Services Security, Set-Based Algebra, XACML, Policy Evaluation, Policy Analysis, Real-Time Decision, Access Control
# Table of contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I - Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivations and Problem Statement</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Objectives</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Thesis Organization</td>
<td>7</td>
</tr>
<tr>
<td>II - Background</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Web Services</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Information Security</td>
<td>11</td>
</tr>
<tr>
<td>2.4 Access Control</td>
<td>13</td>
</tr>
<tr>
<td>2.5 Web Service Security</td>
<td>15</td>
</tr>
<tr>
<td>2.6 XACML Description and Illustrative Example</td>
<td>17</td>
</tr>
<tr>
<td>2.6.1 XACML Evaluation and Authorization</td>
<td>17</td>
</tr>
<tr>
<td>2.6.2 XACML Policy Structure</td>
<td>19</td>
</tr>
<tr>
<td>2.6.3 Illustrative Example</td>
<td>21</td>
</tr>
<tr>
<td>2.7 Formal Description and Verification</td>
<td>25</td>
</tr>
<tr>
<td>2.7.1 Syntax and Semantics</td>
<td>25</td>
</tr>
<tr>
<td>2.7.2 Set Theory</td>
<td>27</td>
</tr>
<tr>
<td>2.7.3 Formal Verification</td>
<td>29</td>
</tr>
<tr>
<td>2.8 Related Work</td>
<td>29</td>
</tr>
<tr>
<td>2.8.1 Policy Evaluation</td>
<td>30</td>
</tr>
<tr>
<td>2.8.2 Policy Analysis</td>
<td>32</td>
</tr>
<tr>
<td>2.9 Conclusion</td>
<td>35</td>
</tr>
<tr>
<td>III - SBA-XACML Language and Evaluation</td>
<td>36</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>36</td>
</tr>
<tr>
<td>3.2 Approach Overview and Architecture</td>
<td>37</td>
</tr>
<tr>
<td>3.3 SBA-XACML Language Description</td>
<td>39</td>
</tr>
<tr>
<td>3.3.1 SBA-XACML Based Policy</td>
<td>40</td>
</tr>
<tr>
<td>3.3.2 SBA-XACML Request</td>
<td>44</td>
</tr>
<tr>
<td>3.3.3 SBA-XACML Response</td>
<td>45</td>
</tr>
<tr>
<td>3.4 Policy Evaluation Semantics</td>
<td>45</td>
</tr>
<tr>
<td>3.4.1 Match Function</td>
<td>46</td>
</tr>
</tbody>
</table>
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XACML Policy Structure</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Web Service Architecture</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Access Control Architecture</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>XACML Data Flow Diagram</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>XACML Rule and Policy Combining Algorithms</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>XACML Policy Language Model</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>SBA-XACML Architecture</td>
<td>37</td>
</tr>
<tr>
<td>8</td>
<td>Experimental Results on Real-World XACML Policies</td>
<td>68</td>
</tr>
<tr>
<td>9</td>
<td>Synthetic Policy Evaluation</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>Real Policy Evaluation</td>
<td>69</td>
</tr>
<tr>
<td>11</td>
<td>Policy Conversion</td>
<td>70</td>
</tr>
<tr>
<td>12</td>
<td>Synthetic Policy Evaluation with Policy Conversion</td>
<td>71</td>
</tr>
<tr>
<td>13</td>
<td>Real Policy Evaluation with Policy Conversion</td>
<td>71</td>
</tr>
<tr>
<td>Table Number</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>Match Function Semantics Rules</td>
<td>46</td>
</tr>
<tr>
<td>2</td>
<td>Evaluation Semantics Rules of a Policy Rule</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>Evaluation Semantics Rules of a Policy (RCA=Permit-Overrides)</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>Evaluation Semantics Rules of a Policy (RCA=Deny-Overrides)</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>Evaluation Semantics Rules of a Policy (RCA=First-Applicable)</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>Evaluation Semantics Rules of a PolicySet (PCA=Permit-Overrides)</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>Evaluation Semantics Rules of a PolicySet (PCA=Deny-Overrides)</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>Evaluation Semantics Rules of a PolicySet (PCA=First-Applicable)</td>
<td>53</td>
</tr>
<tr>
<td>9</td>
<td>Evaluation Semantics Rules of a PolicySet (PCA=Only-one-Applicable)</td>
<td>54</td>
</tr>
<tr>
<td>10</td>
<td>Results of Semantics-Based Policy Evaluation</td>
<td>65</td>
</tr>
<tr>
<td>11</td>
<td>Subset Function Semantics Rules</td>
<td>74</td>
</tr>
<tr>
<td>12</td>
<td>Intersection Function Semantics Rules</td>
<td>75</td>
</tr>
<tr>
<td>13</td>
<td>Rules of Access Flaw Detection Semantics</td>
<td>76</td>
</tr>
<tr>
<td>14</td>
<td>Rules of Redundancy Detection Semantics</td>
<td>77</td>
</tr>
<tr>
<td>15</td>
<td>Rules of Conflict Detection Semantics</td>
<td>78</td>
</tr>
</tbody>
</table>
Listings

2.1 SOAP Request message without WS-Security ......................... 16
2.2 SOAP Request message with WS-Security .......................... 17
2.3 XACML Policy for a Bank Service Part I ........................... 22
2.4 XACML Policy for a Bank Service Part II .......................... 23
2.5 XACML Access Request ............................................. 24
2.6 XACML Access Response ............................................. 25
3.1 SBA-XACML Policy for a Bank Service ............................ 62
3.2 SBA-XACML Access Request ....................................... 64
3.3 SBA-XACML Response ............................................... 66
4.1 SBA-XACML Policy for a Bank Service ............................ 84
Chapter One

Introduction

1.1 Motivations and Problem Statement

Distributed cloud and Web services are becoming very popular and constituting the primary techniques for data exchange between distributed systems and partners. However, they are still facing the risk of exploits due to the vast accessibility of these services over the Internet [2]. Moreover, critical services are emerging such as banking and other business transactions, which raise 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 [18]. XACML has been known as the de facto standard widely used by many vendors for specifying access control policies. It has been emerged as alternative solution to the traditional way of embedding policy
XACML is an XML-based standard for communicating and enforcing access control policies between services and servers [18]. The XACML based policy has complex structure partitioned into three layers as illustrated in Fig. 1: The top layer contains policy sets, the middle layer contains policies and the lower layer contains rules. Each of the three layers has its own target, which contains a set of subjects, resources and actions. Every policy set has a combining algorithm to make the final decision in case of a tie between its policies, and every policy has a combining algorithm to make the final decision in case of a tie between its rules. The PolicySet example illustrated in Fig. 1 contains one policy with three rules. The policy P1 (line 3) has a rule combining algorithm "Permit-Overrides". P1 is applicable to requests with Resource equal to ServiceA (line 6). Rule R1 (line 9) permits access to any request with Resource equal to ServiceA (line 14), Subject not specified and Action not specified. Rule R2 (line 19) permits access to requests with Resource equal to ServiceA (line 15), Subject equal to Joe (line 30) and Action not specified. Rule R3 (line 36) denies access to requests with Resource equal to ServiceA (line 42), Subject equal to Joe (line 47) and Action not specified. Any request for ServiceA will be allowed because rule R1 grants access to any request with Resource equal to ServiceA, while the remaining rules are disregarded. According to the current XACML engine [18], each request is submitted to the Policy Enforcement Point (PEP) that formulates it using XACML language. Consequently, the Policy Decision Point (PDP) checks at runtime the request with respect to the policy in order to determine access or deny decision. The final decision is enforced by the PEP. This whole process is referred to by policy evaluation.
With the growth of web-based applications, the size and complexity of XACML policies are significantly growing to cope with this evolution. Some real-life composed policies may nowadays embed hundreds and even thousands of rules. This growth raised many concerns related to the efficiency of real-time decision process and the correctness (i.e. flaw and conflict free) of complex policies. This thesis targets both problems. First, as aforementioned, XACML evaluation engine is responsible of verifying all the rules of all the participating policies, in addition to resolving their corresponding combining algorithms, in order to handle the decisions to the requests at runtime. Hence, enforcing large size XACML policies will decrease the efficiency of policy evaluation engine, and consequently may create performance bottleneck for the whole services. Several approaches [12,17,19,24] have been proposed to ameliorate the performance of policy evaluation process of the original
XACML engine [18]. However, these propositions entail major modification on the Sun PDP architecture [18] and assumptions in terms of continuous policy loading and cumulative reception of all requests, which do not always hold in real world environment and limit their efficiency and usefulness. More details about these limitations are presented in Chapter Two, Section 2.8. Hence, decreasing the overhead of XACML evaluation process still constitutes a real challenge.

Second, the complex structure of XACML makes policies candidate for insertion of possible flaws, conflicts and redundancies between policies and rules. For instance, considering the PolicySet example in Fig. 1, both Rules R1 and R2 of policy P1 have same Resource ServiceA and effect "Permit", with no Action specified. R1 does not limit access to specific set of subjects, while R2 limits access to requests with Subject equal to "Joe", hence it is safe to say that R2 is a subset of R1. The policy combining algorithm is "Permit-Overrides". Based on the given policy specification, R1 and R2 may lead to access flaw because 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. This contradicts with an access control policy objective that should enforce the more restricted rule. Moreover, rules R2 and R3 may lead to conflict since they are exactly the same but with opposite effects. 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). Although such manual remediation seems feasible for small scale policies, this is definitely problematic for large scale ones with hundreds and even thousands of policies and
rules. The problem grows more when several XACML policies from different parties are integrated and composed together (e.g. in business process model and community of web services), where contractions between combining algorithms are also risen. In this regard, few approaches have been proposed addressing XACML policy composition and analysis [5, 8, 10, 16, 25, 31, 32]. However, these propositions did not address the aforementioned flaws and conflicts problems and are still missing some XACML elements such as rule conditions and obligations.

1.2 Objectives

The main aim of this thesis is to provide an efficient solution for XACML analysis and real-time decision evaluation to cope with the growth of web applications and service in terms of size and complexity. The main corresponding objectives are:

- Address the XACML complexity problem by elaborating an alternative language with standardized structure unlike XACML.

- Elaborate an efficient evaluation approach for realtime decisions on XACML policy requests.

- Provide a formal analysis method for detecting access control flaws, conflicts and redundancies during the creation phase of access control policies.
1.3 Contributions

In this thesis, we address the aforementioned complexity, performance and analysis problems by elaborating a novel and complete formalization of XACML based on sets. Formal methods give advantages over other approaches because they provide a high level of assurance, which is a key factor for the security of critical services. The formal specification of policies and rules using sets is allowing us to efficiently perform evaluation and analysis tasks. To the best of our knowledge, we are the first targeting both concerns simultaneously. The proposed SBA-XACML scheme is composed of a formal language including an automatic converter and compiler, a policy evaluation module based on formal semantics, and a policy analysis module based on formal verification semantics. All the approach components have been implemented in one development framework that accepts XACML policies and requests as inputs, converts them automatically to SBA-XACML constructs when needed, performs analysis and reports the existence of flaws, and evaluates the requests and policies to provide the final access decision. The provided formal and practical experiments conducted on real-life and synthetic XACML policies explore the efficiency, relevance and scalability of our proposition for policy evaluation and security flaws detection. In this context, the main contributions of SBA-XACML and this thesis are three folds:

- Set-Based intermediate representation of XACML constructs into readable mathematical syntax that maintain the same XACML policy structure and account for all its elements and their sub elements including rule conditions, obligations, policy request and policy response. The corresponding language and compiler offer automatic
and optional conversion from XACML to SBA-XACML constructs.

- Formal semantics and its implemented algorithms that take advantage of the mathematical operations to provide efficient policy evaluation. Unlike current literature, the adopted approach maintains the same architecture of the industrial standard XACML Sun PDP [18] and respects the major properties and assumptions of real-life environments in terms of remote policy loading upon need and disjoint reception of requests from distributed parties. The presented experimental results explore that SBA-XACML evaluation of large and small sizes policies has better performance than Sun PDP [18] and its corresponding ameliorations [12, 17, 19, 24].

- Formal semantics and its implemented algorithms for XACML policy analysis that enable to detect access flaws, conflicts and redundancies at both policy and rule levels.

### 1.4 Thesis Organization

The remaining of this thesis is organized as follows:

In Chapter Two, we present an introduction on the concepts of Web services, information security, access control, Web services security, XACML, Formal Description and Verification. Afterwards, we discuss some of the related works done in the area of XACML.

In Chapter Three, we describe the proposed SBA-XACML approach and language. First, we present the approach schema and architecture, the language syntax and semantics.
Then, we state the algorithms for real-time decision evaluation. Finally, we share a study to show the effectiveness and results of our proposal with respect to other approaches.

In Chapter Four, we present the proposed SBA-XACML Analysis. First, we describe the policy analysis semantics. Then, we state the policy analysis algorithms. Finally, a complete case study is illustrated to demonstrate the usefulness our proposal.

In Chapter Five, we summarize the accomplishment, contributions, future work and list of publications gained from this thesis.
Chapter Two

Background

2.1 Introduction

This chapter is devoted to several concepts used in our research. First, we present an overview about Web services, architecture and components. Second, we discuss the importance of Information security and its objectives. Third, we briefly explain the process of access control and its architecture. Fourth, we provide an overview of Web service security and a demonstration example. Fifth, we present the XACML description and an illustrative example. Sixth, we discuss the formal description and verification. Finally, we present an overview of the literature related to our work and their limitations.
2.2 Web Services

Web services are very popular nowadays. They are the backbone of Internet. They provide the opportunity to shift away from highly coupled systems to more loosely dynamic systems. One outstanding advantage for web services is that they are application platform and technology independent. This advantage allows and encourages Web services-based applications to be freely joined with different components from different technologies. Web services fulfill a specific task or a set of tasks. They can be used alone or with other Web services to carry out a simple and complex transactions. Web services do not provide a graphical user interface for interaction with end users but instead it behaves as an API (Application Programming Interface) over the Web. Integration of Web services are fast, easy and cheap. Fig. 2 represents the architectural module of Web services.

As illustrated in Fig. refWeb Service Architecture, Web services are self-contained modular applications that can be described, published and invoked over the Web. They
are discovered via the UDDI registry and described via the WSDL file. The WSDL file provides requesters with a set of operations and their definitions. Below is a list of the main processes:

- **WSDL (Web Service Description Language)** is a file which contains the description of the web service operations it supports. This file is developed and published after the development of the Web service.

- **UDDI (Universal Description, Discovery and Integration)** is a registry of all Web service’s meta-data including a pointer to the WSDL description of a service.

- **Web service module** is the component for providing services.

- **Service consumer** is the service requester.

The communications between service consumers and Web services are done through SOAP (Simple Object Access Protocol) messages. The messaging structure is in XML (eXtensible Markup Language). Web services are very flexible and easy to develop and deploy. A Web service can call another Web service. They enable access to information regardless of methods and devices.

### 2.3 Information Security

Information security is the process of securing information from unauthorized access. Information security is concerned with data being transmitted from one host to another as well as the physical data which resides on a server or a drive. It is the prevention of any disclosure of information regardless of the state. In this thesis we are just concerned with
online information. There are several different methods which may cause theft, damage and disclosure to information. Some of the methods are: weak access control policies, malicious attacks, insiders, virus, etc.

Security measures must be taking to protect assets against attacks regardless of their kind and state. Systems must be built with security measures from the start. Security is not something that can be added to an existing system, it must be designed as a part of the system from the start. Systems should be built to resist attacks but at minimum, a system should be able to recover in case of a successful attack. Any system with good information security if its data is never lost, damaged or disclosed. A successful attack can damage reputation, lost revenue and future businesses. Risk analysis should be considered to evaluate the existing system security and to correct any weaknesses and to apply the proper security measures to prevent future attacks. It is very important that businesses perform risk analysis to have a clear idea of where they stand with respect to security. Risk analysis provides a report of where the weaknesses are and the type of threats they may face. In addition, risk analysis gives an estimate of how much assets worth to be able to spend on securing them.

Information security is the protection of information related assets. The main objectives of information security are: confidentiality, integrity, availability, authenticity and accountability.

- Confidentiality: prevent unauthorised disclosure of information.
- Integrity: prevent unauthorised modification of information.
- Availability: prevent unauthorised withholding of information or resources.
• Authenticity: verify the identity of an entity or source of information.

• Accountability: prove that an entity was involved in some actions

Information security is a wide topic but our focus in this thesis are the security issues which are related to Web services. Web services are fairly new technologies but many companies and agencies have been rushing in with many projects already been deployed and used in production. Web services are simple to build and deploy for data sharing and advertising. However, the security concerns are still the leading issue and the top investment areas for government agencies and companies.

2.4 Access Control

The heavy reliance on Internet communications for data exchange, access control is a mandatory measure to ensure that unauthorized users do not gain access and maintain minimum access to authorized users to perform their duties. Access control is a critical security measure because it involves both computer scientists and engineers to provide proper security design and implementation. Access is not just about requiring a user name and password when users request access to resources, they are much more complex. There are multiple methods and technologies that can be implemented to support and administer access for different areas. Access control mechanisms are always changing to be able to resist against a variety of attacks.

An access control system monitors access requests and implements policies which establish who can, or cannot, execute which actions on which resources [28]. Its process is
partitioned into three steps: identification, authentication and authorization. Identification is the step where the subject is identified to the requested service. Authentication is the second step of access control and it is the process of verifying the subject requesting the service. Authorization is the third step and it is determining the type of access to resources a subject can have. Fig. 3 shows a generic architectural module of access control.

Fig. 3 illustrates the different steps involved in the access control process. The first step is a user action entering a username and a password in a form. The second step takes place on the system which houses the credentials for user access. During this step, the authentication server matches both the username and password entered in the first step against an entry in the access database. If match found then the user is authenticated otherwise, access denied is returned. The third step is verifying the privileges of the user requesting access against the service requested. If the user’s access privilege is equivalent to the service’s requested then the user’s request is granted otherwise denied.

Access control is always evolving to address the security concerns to increase user confidence. It is shifting from the classical methods such as user name and pass phrases
towards biometrics. Biometric access control provides a higher level of securing access. The term 'biometrics' refers to a measurable characteristic that is unique to an individual such as fingerprints, facial structure, iris, etc.

2.5 Web Service Security

Web service security has been in the research spotlight and its main concern is providing better security for Web services with the objectives of confidentiality and integrity. Many related standards have been proposed and used and many others in theory. The aim of these is to provide web services with more robust security mechanisms. Protections required for Web services at different layers as distributed applications because they face the same security risks such as SQL Injection, Cross-Site scripting, broken authentication, Missing Function Level Access Control, etc [13]. However, they are more vulnerable because they expose more sensitive data between partners.

Web services use SOAP messages for data exchange between two end points but SOAP does not provide a mechanism for ensuring data integrity or confidentiality either at rest or during transit. Its communication security remains a critical task to be tackled and a costly investment due to the complexity of securing Web services because their descriptions of how they work is made public over the Internet. One of the leading security standards for providing security at the message level for Web services is WS-Security (Web Service Security).

WS-Security [1] is an extension to SOAP. It has been introduced and published by OASIS in order to apply security in Web Services through using existing standards and
specifications. WS-Security is not a stand alone solution for securing data exchange but it is a method that allows security standards to be used such as encryptions, digital signatures, etc. It supports whatever security related data to be defined in the header of the SOAP message. If XML Signature is used, the SOAP header can contain the Signature method, the key and the resulting signature value. Likewise, the header can contain the encryption information if any elements within the message are encrypted. WS-Security does not specify the format of the signature or encryption but instead, it specifies how one would embed the security information laid out by other specifications within a SOAP message. WS-Security is primarily a specification for an XML-based security meta-data container. The following is an example of SOAP to demonstrate the difference between secure and insecure messages.

Listing 2.1 shows a demonstration of SOAP messages without security applied [30]. Both username and password fields are inserted in plain text. In this case, anyone with a sniffer is easily able to capture the user and password without too much efforts.

Listing 2.1: SOAP Request message without WS-Security

```xml
<wsse:UsernameToken>
  <wsse:Username>scott</wsse:Username>
  <wsse:Password Type="wsse:PasswordText">password</wsse:Password>
</wsse:UsernameToken>
```

Listing 2.2 shows a demonstration of SOAP messages with WS-Security applied [30].

It shows a SOAP message with WS-Security applied to the password. The username is sent in plain text but the password is encrypted. The password digest is a SHA1 algorithm hash based on the concatenation of the password, message creation time and a nonce. SHA1 algorithm is one of the many available encryption algorithms for security.
Listing 2.2: SOAP Request message with WS-Security

```
<wsse:UsernameToken>
  <wsse:Username>scott</wsse:Username>
  <wsse:Password Type="wsse:PasswordDigest">KE6QugOpkPyT3Eo0SEgT30W4Keq</wsse:Password>
  <wsse:Nonce>5uW4ABku/m6/S5rnE+L7vg==</wsse:Nonce>
  <wsu:Created xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">2002-08-19T00:44:02Z
</wsu:Created>
</wsse:UsernameToken>
```

2.6 XACML Description and Illustrative Example

XACML (eXtensible Access Control Markup Language) [18] is the OASIS standard language for access control policies. XACML is the default access control for web services. Today, it is the most popular and well known access control mechanism. XACML is XML-based standard for communicating access control policies between services and provides XML schema for access control policies, requests and responses [9]. XACML is not just a standard for describing access control policy in XML but also is a complete application for evaluating any given request against a policy. We will discuss the XACML evaluation/authorization and the structure of access control policy. The XACML data flow is depicted in Fig. 4.

2.6.1 XACML Evaluation and Authorization

The XACML Evaluation and Authorization contain five processes: Policy Enforcement Point (PEP), Policy Administration Point (PAP), Policy Decision Point (PDP), Policy Information Point (PIP), and a context handler [18].

PAP: PAP is the repository for the policies and provides the policies to the PDP. PDP: The system entity which handles the evaluation and returns the authorization decision to PEP. PEP: The system entity that performs access control, by making decision requests and
enforcing authorization decisions. PIP: The system entity that acts the attribute authority. Context handler: Handles the data conversion and communications between PDP, PEP and PIP.

Fig. 4: XACML Data Flow Diagram

Fig. 4 illustrates processes listed above and the information flow. PAP writes policies and makes them available to the PDP. The PEP receives the request and sends it to the context handler. The Context handler translates the request and sends it to the PDP for a decision. The PDP requests additional attributes from the context handler during the evaluation. The Context Handler collects these attributes by the help of the PIP from the resources, subjects, and the environment. The PDP sends the XACML Response to the Context Handler after the evaluation is finished. The Context Handler converts the response message and send to PEP. If there are any obligations to be fulfilled based on the
authorization decision, the PEP handles them.

### 2.6.2 XACML Policy Structure

XACML access policy is ordered into 3 levels: `<PolicySet>`, `<Policy>` and `<Rule>`. Every level in the policy has a target. The target element is used to determine whether the `<PolicySet>`, `<Policy>` or `<Rule>` is applicable to the request. If the target does not match then NotApplicable is returned. The `<PolicySet>` element contains a set of `<Policy>` elements and a Policy Combining Algorithm (PCA) used for combining the results of the evaluation of individual policies. The `<Policy>` element contains a set of `<Rule>` elements and a Rule Combining Algorithm (RCA) used for combining the results of the evaluation of individual rules. The Combining Algorithm works the same way at the policy and rule levels. Fig. 5 illustrates the Combining Algorithms for both Policy and rule levels.

<table>
<thead>
<tr>
<th>Combining Algorithm</th>
<th>Behavior Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deny-Overrides</td>
<td>If a single rule or policy evaluates to Deny then regardless of the others the end result is “Deny”.</td>
</tr>
<tr>
<td>Permit-Overrides</td>
<td>If a single rule or policy evaluates to Permit then regardless of the others the end result is “Permit”.</td>
</tr>
<tr>
<td>First-Applicable</td>
<td>The end result is the result of the first applicable rule or policy.</td>
</tr>
<tr>
<td>Only-one-Applicable</td>
<td>Applies only at the policy level. If one and only one policy is applicable then the end result is the evaluation result of the applicable policy. If more than one policy is applicable then the end result is “Indeterminate”. If no policies are applicable then the end result is “NotApplicable”.</td>
</tr>
</tbody>
</table>

Fig. 5: XACML Rule and Policy Combining Algorithms

The `<PolicySet>` and `<Policy>` elements can have reference to other policies. They can
have obligations to fulfill whenever the decision is either permit or deny. A rule is the most elementary in the access policy. The main components of a rule are target, condition that are represented by subelements and effect which is included as an attribute of the Rule element. The condition attribute is a boolean function over subjects, resources and actions. If the conditions evaluate to true then the rule effect is returned otherwise NotApplicable is returned. Fig. 6 presents the policy language model.

![Policy Language Model Diagram]

**Fig. 6: XACML Policy Language Model**

Fig. 6 shows the policy structure and the relationship between the individual elements. The <PolicySet> element has a target, a Policy Combining Algorithm, a set of Policies and a Set of obligations. The <Policy> element has a target, a Rule Combining Algorithm, a set of Rules and a Set of obligations. The <Rule> element has a target, a rule effect, and a set of Conditions. The Target element is based on a set of subjects, resources and actions.
XACML is a well known mechanism for access control especially for Web applications and Web services. It supports business specific customizations. It is very flexible but it is very complex at the same time. It supports dynamic access control with policies applied at runtime but still has its disadvantages. As of a result of its complexity, it is hard to analyze in order to determine if any flaws exists in the based policy. Its richness in expressions makes it harder to create standardized tools for examining for conflicts, access flaws and redundancies. In addition, integrating multiple XACML policies is a nightmare.

2.6.3 Illustrative Example

In this section, we provide an XACML based policy, request, and response of the evaluation process according to XACML syntax and Sun PDP engine [18]. We will adopt this example throughout the thesis. A based policy for a Bank service is presented in listing 2.3 and listing 2.4. The policy set contains two policies $P_1$ and $P_2$. $P_1$ contains two rules $R_1$ and $R_2$. It has a rule combining algorithm permit – overrides. $R_1$ permits access to BankService/withdraw resource if the subject is Bob. $R_2$ denies access to any resources for any subjects. If policy $P_1$ evaluates to permit, it has an obligation to send an email to Customer_service@bank.com. $P_2$ contains three rules $R_3$, $R_4$ and $R_5$. It has a rule combining algorithm permit – overrides. $R_3$ permits access to BankService/deposit resource for any subjects. $R_4$ permits access to BankService/deposit resource if the subject is Joe. $R_5$ denies access to BankService/deposit resource if the subject is Joe.

The policy set ID and policy combining algorithm PCA are stated in line 3. Lines 4 to 51 contain the policy $P_1$. The rule combining algorithm RCA is permit – overrides (line
Listing 2.3: XACML Policy for a Bank Service Part I

```
<PolicySet xmlns="schema:os" PolicyCombiningAlgId="policy-combining-algorithm:permit-overrides" PolicySetId="PS1">
  <Policy PolicyId="P1" RuleCombiningAlgId="rule-combining-algorithm:permit-overrides">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="function:string-equal">
            <AttributeValue DataType="xml:string">Jerry</AttributeValue>
          </SubjectMatch>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="xml:string">Bob</AttributeValue>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="function:string-equal">
            <AttributeValue DataType="xml:string">BankService/withdraw</AttributeValue>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <AnyAction />
      </Actions>
    </Target>
    <Rule Effect="Permit" RuleId="R1">
      <Condition>
        <Apply FunctionId="function:and">
          <Apply FunctionId="function:string-equal">
            <ResourceAttributeDesignator AttributeId="resource:resource-id" DataType="xml:string">BankService/withdraw</ResourceAttributeDesignator>
          </Apply>
          <Apply FunctionId="function:string-equal">
            <SubjectAttributeDesignator AttributeId="subject:subject-id" DataType="xml:string">Bob</SubjectAttributeDesignator>
          </Apply>
        </Apply>
      </Condition>
    </Rule>
    <Rule Effect="Deny" RuleId="R2" />
    <Target/>
    <Obligations>
      <Obligation FulfillOn="Permit" ObligationId="Withdraw">
        <AttributeAssignment AttributeId="example:attribute:mailto" DataType="xml:string">Customer_service@bank.com</AttributeAssignment>
        <SubjectAttributeDesignator AttributeId="subject:subject-id" DataType="xml:string">subject:subject-id</SubjectAttributeDesignator>
        <ResourceAttributeDesignator AttributeId="resource:resource-id" DataType="xml:string">resource:resource-id</ResourceAttributeDesignator>
      </Obligation>
    </Obligations>
  </Policy>
</PolicySet>
```
4. Lines 5 to 29 is the policy target. Subjects equal to Jerry and Bob specified in lines 9 and 13. Resource BankService/withdraw is defined in line 21. Actions are defined to be any in line 27. Policy P1 contains two rules R1 and R2. Rule R1 starts in line 30 and ends in line 41. R1 has a permit rule effect. The rule conditions are the subject = Bob and resource = BankService/withdraw in lines 34 and 37 respectively. R2 defined in line 42. Deny anything to anyone. Policy P1 has an obligation to perform if the policy evaluates to permit. Obligations are defined in lines 44 to 50. The policy P2 starts in lines 51 to 85. It contains three rules R3, R4 and R5. The RCA of P2 is permit − overrides.
and is listed in line 52. No Target is defined for $P_2$. $R_3$ starts in line 54 and ends in line 60 with rule effect equal \textit{permit}. $R_3$ has one condition, which is the resource id must match \textit{BankService/deposit}. Rule $R_4$ starts in line 61 and ends in line 72 with rule effect equal \textit{permit}. $R_4$ conditions are that the resource id which must match \textit{BankService/deposit} and subject id \textit{Joe}. Rule $R_5$ starts in line 73 and ends in line 84 with rule effect equal \textit{Deny}. $R_5$ conditions states that the resource id must match \textit{BankService/deposit} and subject id \textit{Joe}.

Listing 2.5 contains the XACML request. The request is calling for a resource \textit{BankService/deposit} with a subject \textit{Bob} and action \textit{execute}. Lines 4, 9 and 14 contain subject, resource and action respectively.

\begin{verbatim}
Listing 2.5: XACML Access Request

[1]. <Request xmlns="context:schema:os" xmlns:xsi="XMLSchema-instance">
[2].  <Subject SubjectCategory="subject-category:access-subject">
[3].   <Attribute AttributeId="subject:subject-id" DataType="xml:string">
[4].     <AttributeValue>Bob</AttributeValue>
[5].   </Attribute>
[6].  </Subject>
[7].  <Resource>
[8].   <Attribute AttributeId="resource:resource-id" DataType="xml:string">
[9].     <AttributeValue>BankService/deposit</AttributeValue>
[10]. </Attribute>
[12]. <Action>
[13].   <Attribute AttributeId="action:action-id" DataType="xml:string">
[14].     <AttributeValue>execute</AttributeValue>
[15].   </Attribute>
[16]. </Action>
[17]. </Request>
\end{verbatim}

Listing 2.6 contains the XACML response to the request in Listing 2.5 against the based policy in listing 2.3 and listing 2.4. The response is the final decision for the resource \textit{BankService/deposit} with a subject \textit{Bob} and action \textit{execute}.  

24
### Listing 2.6: XACML Access Response

```xml
[1]. <Response>
[2]. <Result ResourceID="BankService/deposit">
[3]. <Decision>Permit</Decision>
[4]. <Status>
[6]. </Status>
[7]. </Result>
[8]. </Response>
```

## 2.7 Formal Description and Verification

A formal description consists of strings, symbols and rules forming a language. It is called formal because it is based on mathematical notations. The strings and symbols are grouped together according to a set of rules which are defined specifically for the language. A formal description language includes both language syntax and semantics. We use such languages to express rules and perform formal verification. In this section, we describe each of these components.

### 2.7.1 Syntax and Semantics

Language Syntax provides the structure of how strings and symbols are combined together according to the certain rules without any considerations to their meanings. Formal semantics constitutes of rigorous mathematical study of the meaning of languages and models of computation [22, 26]. It allows to prove the properties of a program. The formal semantics of a language is specified by a mathematical model that illustrates the possible computations described by the language. There are many approaches to formal semantics that belong to three major classes: Operational semantics, denotational semantics and axiomatic semantics. These three classes are presented in the increasing order of abstraction with respect to the concepts of meaning underlying them. The following is a brief
description for each one of them:

- Operational semantics describes the execution of the language directly rather than by translation. It somehow corresponds to interpretation, where the implementation language of the interpreter is a mathematical formalism. The operational semantics may define an abstract machine and give meaning to the transitions between its states. It may also be defined via syntactic transformations on phrases of the language itself.

- Denotational semantics translates each phrase in the language to another phrase in another language. It somehow corresponds to compilation, where the target language is a mathematical formalism.

- Axiomatic semantics gives meaning to phrases by expressing the logical axioms that apply to them. Axiomatic semantics does not distinguish between a phrase meaning and the logical formulas describing it. A phrase means exactly what can be proven about it in some logic.

Since this thesis presents an operational semantics for XACML policies evaluation and analysis, in the sequel we elaborate more about this approach and introduce the used structural operational semantics. Operational semantics is considered as a method to give meaning to programs and operations in a mathematically rigorous way. It describes how a valid process is interpreted as sequences of computational steps, which then constitute the meaning of the whole process. The final step in the terminating sequence returns the value of the process in the case of a functional process. A process could be also nondeterministic, in this context there may be many computation sequences and many return values.
Structural operational semantics is an approach proposed to give logical means in defining operational semantics [23]. It consists of defining the behavior of a process in terms of the behavior of its parts. Hence, it provides a structural, a syntax oriented and an inductive view on operational semantics. Computation is represented by means of deductive systems that turn the abstract machine into a system of logical inferences. This allows to apply formal analysis on the behavior of processes. The proofs of process properties are derived directly from the definitions of the language constructs because the semantics descriptions are based on deductive logic. With structural operational semantics, the behavior of a process is defined in terms of a set of transition relations. Such specifications take the form of inference rules. The valid transitions of a composite piece of syntax is defined into these rules in terms of the transitions of its components. 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, as follows [26]:

\[
\frac{\text{premise}_1 \quad \text{premise}_2 \quad \ldots \quad \text{premise}_n}{\text{conclusion}}
\]

If \( n=0 \), i.e., the number of premises is zero, then the line containing the premises is omitted, and we refer to the rule as an axiom.

2.7.2 Set Theory

Set theory is the study of sets. A set is a group of items known as elements [4]. When describing a set in Mathematics, all of its elements are listed in a row separated by commas
and enclosed with curly braces. The name of sets are usually presented in upper case.

For example, a set S containing the elements 1, 4, 2, 8 and 9 could be shown as follows: S = \{1, 4, 2, 8, 9\}. An element e in a set S is expressed in Mathematical notation: e ∈ S. The symbol ∈ in set notation means is a member of. Any set S is a subset of itself and expressed as S ⊆ S. The symbol ⊆ in set notation means a subset or equal.

A set can contain no elements. This type of sets is referred to as an empty set or null set and is denoted by the symbol ∅. An empty set is a subset of any set. A set A is a subset of a set B if every element in A is an element in B. It can be written as A ⊆ B to designate such relationship, but if the set B contains elements which are not in the set A then we can say that A is a proper set of B and it is written as A ⊂ B. Two sets A and B are equal if A ⊆ B and B ⊆ A. If the two sets A and B are equal, we write A = B to designate the equality relationship.

Operations which are performed on sets include union, intersection, cartesian product, etc. A union is a Mathematical operation for sets and it is denoted by the symbol ∪. The union of two sets is another set whose elements include the members of each original set. The common elements between the two sets are only counted once. For example, a set A = \{1, 2, 3\} and a set B = \{7, 2, 8\}. A ∪ B = \{1, 2, 3, 7, 8\}. An intersection is a Mathematical operation for sets and it is denoted by the symbol ∩. The intersection of two sets is another set whose elements are the common the elements of the two original sets. If the original two sets share no elements then the intersection of the two is the empty set. For example, a set A = \{1, 2, 3\} and a set B = \{7, 2, 8\}. A ∩ B = \{2\}. A cartesian product is a Mathematical operation and it is denoted by the symbol ×. A cartesian product of two sets returns a product set from the two sets. That is, for sets A and B, the Cartesian product A × B is the
set of all ordered pairs $a, b$ where $a \in A$ and $b \in B$. For example, a set $A = \{1,2\}$ and a set $B = \{7,8\}$. $A \times B = \{1,7,1,8,2,7,2,8\}$.

### 2.7.3 Formal Verification

The main purpose of providing a formal description is to perform formal verification on the described entity with respect to specific properties [7]. In other words, formal verification is a mathematical method which focuses on verifying the correctness of a system with respect to a set of specifications or properties. This process provides an opportunity to test and correct defects in the early phases of the development cycle. It has been increasingly used in the development of critical systems such as traffic lights, air traffic control, banking systems, etc. They have many advantages in providing higher quality and accuracy. However, they have few disadvantages such as difficult for unknowledgable personnel, expensive and time consuming. There are many methods for formal verification such as Process Algebra, Petri Nets, Temporal Logic and Finite State Machine, etc.

### 2.8 Related Work

Our approach is targeting both XACML policy evaluation and analysis. To the best of our knowledge, we are the first proposing a scheme for efficient policy evaluation, which offers simultaneously policy analysis mechanisms for flaws and conflicts detection. In this context, we provide in this section an overview of the related work in the literature addressing both problems independently.
2.8.1 Policy Evaluation

Regarding policy evaluation, several approaches have been proposed to provide efficient evaluation process. Liu et al. [12] proposed the XEngine which is a scheme for efficient XACML policy evaluation. It is an extension to the SUN Policy Decision Point (PDP) [18]. Their approach improves the performance of the PDP by numericalization and normalization of the XACML Policies. It consists of 3 steps. The first one is the conversion process of all the strings of XACML based policy and requests to numerical values. The second one is the normalization process which is the conversion of the output from the first step to hierarchical structure and conversion of combining algorithm to First-applicable. The third step is creating a tree structure from the second one. Their approach provides amelioration with respect to policy evaluation performance. However, they do not support obligations due to the conversion of all combining algorithms to first applicable [19]. Moreover, the major modification on the Sun PDP architecture and main assumptions of their experiments do not always hold in real world environment, which limit the efficiency and usefulness of their proposition. First, assuming that the policies are always loaded in the memory contradicts with the core concept of XACML [18] and is problematic for large size policies with hundreds and thousands of rules. The policies should be loaded upon request for a short period, where the policy repository can be accessed locally or remotely for security, privacy and memory restriction purposes. Second, our experiments with their tools show that the main overhead reduction is achieved when all the requests (i.e. up to 100,000 requests) are received, converted and loaded in the memory at the same time, then all of them evaluated against the already loaded policies. Again, such assumption does not always hold since
requests can be received from different parties at variant time-space. In this regard, the
provided experimental results explore that our approach provides better performance than
XEngine.

Marouf et al. [17] proposed a clustering and re-ordering techniques for optimizing
XACML performance. The proposed clustering method groups policies and rules based
on target subjects. The re-ordering process is based on statistical analysis of policy and
vibrant stream of requests. This process reduces the evaluation time because applicable
policies and rules are given higher priority to be evaluated first. Although this approach
seems interesting, the assumption of access requests following a consistent distribution and
policy re-ordering does not support obligations. Moreover, they share the same limitations
as XEngine in [12] in terms of major modification to Sun PDP architecture and experiments
assumptions. The provided results show that our approach offers better performance based
on their experiments in [17].

Ngo et al. [19] proposed Multi-Data-Types Interval Decision Diagrams for XACML
Evaluation Engine. Their approach is based on data interval partition aggregation along
with new decision diagram combinations. They claim that their proposed approach does not
only improve the evaluation response time, but also provides correctness and completeness
of evaluation. Their proposed approach seems interesting, however it is only experimented
on small scale policies up to 360 rules, unlike our and other approaches [12, 17].

Pina Ros et al. [24] proposed an optimization for XACML policies evaluation based
on two trees. The first tree is a matching tree which is created for a quick finding of
applicable rules. The second tree is a combining tree which is used for evaluation of the
applicable rules. They proposed a binary search algorithm for finding the matching tree.
This approach supports requests with multi-valued attributes, however the matching tree
does not support policies with multi-valued attributes.

Based on the study of the current literature with respect to policy evaluation perfor-
mance, it is trivial that this domain is still and will continue to be a challenging niche for
researchers. Our approach differs from the aforementioned ones in different aspects. First,
it is maintaining the same policy structure of XACML and architecture of Sun PDP, where
policies are converted into intermediate mathematical and readable syntax. This allowed us
to benefit from the formal description for efficient policy evaluation and analysis purposes.
Second, unlike all the current approaches, our scheme is respecting the major properties
and assumptions made by Sun PDP [18] with respect to real-life environment, where poli-
cies are loaded from local or remote location upon need, and the XACML requests are
received one at a time from distributed parties. Third, our experiments in section 3.6 show
that our proposition outperforms the current approaches.

2.8.2 Policy Analysis

Moving to policy analysis, several approaches have been proposed in this regard. Kolovski
et al. [10] 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 conflicts and do not
support multi-subject requests, complex attribute functions, rule Conditions and Only-One-Applicable combining algorithm.

Fisler et al. [8] 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, conflicts and redundancies, and does not work on all types of XACML policies.

Tschantz et al. [31] 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, their they do not address policy analysis with respect to access flaws, conflicts and redundancies.

Mazzoleni et al. [16] 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, conflicts and redundancy existence.

Bertino et al. [25] 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. [32] 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.

Our approach differs from the aforementioned ones in different aspects. First, it is providing a set-based algebra syntax and semantics that accounts for all the XACML elements, including rules conditions and obligations. Second, it is allowing us to provide a policy analysis framework for detecting different kind of flaws and conflicts in XACML policies, which are not yet addressed by the current propositions. Third, in addition to the analysis features, the elaborated algebra is offering us the capabilities to build efficient policy evaluation scheme.
2.9 Conclusion

In this chapter, we presented an overview of Web services, Information Security, Access Control, Web services security and XACML. We also introduced the formal verification concept and its importance. Finally, We summarised the literature related work around XACML and their limitations.
Chapter Three

SBA-XACML Language and Evaluation

3.1 Introduction

Access control is widely used in Web applications and Web services. It maintains the control of which principle (user or process) has access to which resources in a system and its policies are written in specification language such as XACML [18]. Today, XACML is most used access control mechanism for Web services. However, with the growth of web-based applications, the size and complexity of XACML policies are significantly growing to cope with this evolution. This growth raised many concerns related to the efficiency of real-time decision process. We proposed a set based language to simplify the complexity and increases the responses time for real-time evaluation.

The rest of the chapter is organized as follows. Section 3.2 gives an overview of the approach and architecture. Section 3.3 describes the SBA-XACML language and its components. Section 3.4 provides the formal semantics for policy evaluation. Section 3.5 details the evaluation algorithms. Section 3.6 provides the policy evaluation and performance
analysis. Finally, Section 3.7 concludes the chapter.

### 3.2 Approach Overview and Architecture

In this section, we present the overall architecture of our approach illustrated in Fig. 7.

![SBA-XACML Architecture](image)

Fig. 7: SBA-XACML Architecture

Our proposition includes the SBA-XACML Language, Compiler, Evaluation Module and Analysis Module. All the approach components have been implemented in one development framework that accepts XACML policies and requests as inputs, convert them to SBA-XACML when needed, and perform systematically and automatically all the analysis and evaluation processes. It may also accepts already converted or written SBA-XACML policies and requests. 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. He can also evaluate the requests and get the corresponding responses using the module embedding the evaluation algorithms.

**SBA-XACML Language & Compiler:** SBA-XACML is a formal language based on sets and composed of all the elements and constructs needed for the specification of XACML based policy, request and response. Sections 3.3 presents the complete definition and syntax of SBA-XACML elements and attributes. SBA-XACML compiler includes XACML parser and converter to SBA-XACML. It takes XACML policy set and request as inputs, parses their XACML elements and generates SBA-XACML constructs according to the language syntax and structure. The compiler is implemented using PHP (PHP Hypertext Preprocessor). It can be used independently to convert XACML to SBA-XACML, or as embedded in our framework with the policy evaluation and analysis modules to convert XACML to SBA-XACML at run time.

**Policy Evaluation Module:** This module allows to evaluate a SBA-XACML request against a set of SBA-XACML policies. It is composed of policy-level and rule-level evaluation algorithms (see Section 3.5) that realize the elaborated policy evaluation semantics presented in Section . The policy-level algorithm is responsible for evaluating the policies and triggers the rule-level one in order to evaluate the rules in each policy. It is implemented in PHP and accepts as inputs a SBA-XACML request and a policy set.

**Policy Analysis Module:** This module allows to analyse policies for detecting access control flaws, conflicts and redundancies. It is composed of policy-level and rule-level analysis algorithms that realize the elaborated analysis semantics both presented in Section
4.3. 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 the evaluation module as well. It is implemented in PHP and accepts as input a policy set.

### 3.3 SBA-XACML Language Description

A Set 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. SBA-XACML is a Set-based language. In this following, we present its constructs, operators and structure: \((PS; P (R; Rq); Rs \("Permit"; "Deny"; "NotApplicable"; "Indeterminate")); op (\(\subseteq; \subset; \land; >; \lor; \cap))\), 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.
  - \(Rq\) : represents a request.
  - \(Rs\) : represents a response that contains the final decision.
• "Permit", "Deny", "NotApplicable" and "Indeterminate" are policy constants and represent the final decision embedded in the response.

• $op$ : represents an operator.

  - $\subseteq$ : represents a subset or equal.
  - $\subset$ : represents a subset.
  - $\land$ : represents logical operator "and".
  - $\lor$ : represents logical operator "or".
  - $>$ : represents the precedence order between operations.
  - $\cap$ : represents the intersection between two sets.

In sections 3.3.1, 3.3.2 and 3.3.3, we present the SBA-XACML syntax for the based policy, request and response respectively.

3.3.1 SBA-XACML Based Policy

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 $Policy(s)$. $Policy$ contains $Rule(s)$. $PolicySets$ and $Policies$ have their $Obligations$ to fulfill whenever a $Response$ is reached to either a $Permit$ or $Deny$ decision. In the sequel, we present the definitions and syntax of all the elements.
3.3.1.1 Common Elements Definitions and Syntax

The following are the common elements that are used at the policy set, policy and rule levels.

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:

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

(Construct 1)

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

Obligations $OBLs$ contain one or more obligation(s) $OBL$. An obligation is an action that takes place after a decision has been reached to either Permit or Deny. It is mapped to SBA-XACML within the context of policy and policySet according to the following syntax:

$$OBLs = OBL – Set$$

(Construct 2)

$$OBL = \{OBLID, FFOn, \{\{AttID, DT, V\}\}\}$$

(Construct 3)

where $OBL – Set$ is the the set of obligation $OBL$ to be performed, $OBLID$ is the id identifying the obligation, $FFOn$ is the Fulfill On attribute which is used as a key to determine when the obligation must be enforced and it must be either permit or deny, $AttID$ is the attribute id of the obligation to be carried out, $DT$ is the data type and $V$ is the value. If the policy or policy set being evaluated matches the $FFOn$ attribute of its obligations then the obligations are passed to be enforced otherwise obligations are ignored.
3.3.1.2 PolicySet (PS) Definition and Syntax

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 SBA-XACML according to the following syntax:

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

(Construct 4)

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 (refer to Section 3.3.1.1 for details) and $TR$ is the target (refer to Section 3.3.1.1 for details).

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 and no obligations. The policy set $PS1$ is mapped to SBA-XACML as follows:

$$PS ::= \langle PS1, \{P1, P2\}, \{P1 > P2\}, \{deny – overrides\}, \{\}, \{\}, \{\{Bob\}, \{FileA\}, \{Read\}\} \rangle$$

3.3.1.3 Policy (P) Definition and Syntax

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, target and obligations. It is mapped to SBA-XACML according to the following syntax:
\[ P ::= < ID, SR, PR, RCA, OBLs, TR > \]  

(Construct 5)

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 (refer to Section 3.3.1.1 for details) and \( TR \) is the target (refer to Section 3.3.1.1 for details).

Example 2: Consider a policy \( P_1 \) with two rules \( R_1 \) and \( R_2 \). \( P_1 \) has a \( RCA = \text{permit} - \text{overrides} \), a \( \text{target subject} = \text{Bob} \), \( \text{resource} = \text{FileA} \) and \( \text{action} = \text{write} \) and without any obligations. The policy \( P_1 \) is mapped to SBA-XACML as follows:

\[ P ::= < P_1, \{ R_1, R_2 \}, \{ R_1 > R_2 \}, \{ \text{permit} - \text{overrides} \}, \{ \}, \{ \{ \text{Bob} \}, \{ \text{FileA} \}, \{ \text{write} \} \} > \]

3.3.1.4 Rule (R) Definition and Syntax

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:

\[ R ::= < ID, RC, TR, RE > \]  

(Construct 6)

where \( ID \) is the rule id, \( RC \) is the set of rule conditions, \( TR \) is the target (refer to Section 3.3.1.1 section for details), 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:
\[ RC = \{ \text{Applyfunction}, \{ \text{parameters} \} \} \]

(Construct 7)

where \text{Applyfunction} is the function used in evaluating the elements in the apply and \text{parameters} are the input to the function being applied, each of which is an evaluatble.

Example 3: Consider a rule \( R_1 \) with \text{ruleeffect} = \text{permit}. \( R_1 \) has no target defined. Its only condition is that anyone accessing \text{File} 1 is allowed at any time. The rule is mapped to SBA-XACML as follows:

\[
R ::=< R_1, \{ \text{string} - \text{equal}, \{ \text{ResourceAttributeDesignator}, \text{string}, \text{File} 1 \} \}, \{ \}, \{ \}, \{ \} \}, \{ \text{Permit} \} >
\]

3.3.2 SBA-XACML Request

A request \( Rq \) is a call for access to some resources. It is mapped to SBA-XACML according to the following syntax:

\[
Rq ::= < S_r, R_r, A_r >
\]

(Construct 8)

where \( S_r \) is the set of subjects, \( R_r \) is the set of resources and \( A_r \) is the set of actions.

Example 4: Consider a request calling for access with subject \text{Bob}, resource \text{ServerA} and action \text{read}. The request is mapped to SBA-XACML as follows:

\[
Rq ::= < \{ \text{Bob} \}, \{ \text{ServerA} \}, \{ \text{Read} \} >
\]
3.3.3 SBA-XACML Response

A response $Rs$ is a decision to a request against a based policy. It is mapped to SBA-XACML according to the following syntax:

$$Rs ::= < D, OBLs >$$

(Construct 9)

where $D$ is the decision of the response and $OBLs$ is the set of obligations to be executed within the response (refer to Section 3.3.1.1 section for details).

Example 5: The response to the request in Example 4 is mapped to SBA-XACML as follows:

$$Rs ::= < \{permit\}, {} >$$

3.4 Policy Evaluation Semantics

In this section, we present the formal semantics of a SBA-XACML policy evaluation following the above inference rule structure and deductive logic. Given a policy set $PS$ and a request $Rq$, the response $Rs$ is derived by the evaluation $\rightarrow_{eval}$ of all premises combined between each other using designated operators $op$ as follows:

$$\begin{align*}
(premise_1) & \quad op \quad (premise_2) \quad op \quad ... \quad op \quad (premise_n) \\
\hline
< PS, Rq > & \quad \rightarrow_{eval} \quad Rs
\end{align*}$$

The policy and rule evaluation semantics rules, which constitute the premises in the above rule, have also similar structure and follows the deductive logic until reaching the basic defined premise (i.e. condition). Throughout the rest of the thesis, please note the
difference between a semantic rule that express the evaluation 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. In this context, we start first by defining the MatchFunction semantics rule since it will be used throughout all levels.

### 3.4.1 Match Function

In this section, we present the matching semantics rules for a request \( Rq \) with subject set \( Sr \), resource set \( Rr \) and action set \( Ar \) against a target \( TR \) with subject set \( S \), resource set \( R \) and action set \( A \). The semantics of matching a request and a target is determined by comparing the request subject set \( Sr \) with target subject set \( S \), request resource set \( Rr \) with target resource set \( R \) and request action set \( Ar \) with target action set \( A \).

<table>
<thead>
<tr>
<th>Rule</th>
<th>Match Function Semantics Rules</th>
</tr>
</thead>
</table>
| Rule 1 | \(((Sr \cap S) \neq \emptyset) \land ((Rr \cap R) \neq \emptyset) \land ((Ar \cap A) \neq \emptyset)\)  
   \[ < TR, Rq > \vdash_{\text{match}} True \] |
| Rule 2 | \(((Sr \cap S) = \emptyset) \lor ((Rr \cap R) = \emptyset) \lor ((Ar \cap A) = \emptyset)\)  
   \[ < TR, Rq > \vdash_{\text{match}} False \] |

Rules 1 and 2 in Table 1 describe the different matching cases for a request \( Rq \) with a target \( TR \). In Rule 1 a target \( TR \) matches a request \( Rq \) if the requested subject set \( Sr \) intersects with the target subject set \( S \), the requested resource set \( Rr \) intersects with the target resource set \( R \) and the requested action set \( Ar \) intersects with the target action set \( A \).
In Rule 2 a target TR does not match a request Rq if the requested subject set Sr does not intersects with the target subject set S, the requested resource set Rr does not intersect with the target resource set R or the requested action set Ar does not intersect with the target action set A.

### 3.4.2 Rule Evaluation

In this section, we present the evaluation semantics rules for a request Rq at the rule level.

Semantics Rules 3, 4 and 5 in Table 2 describe the different evaluation cases for a policy rule R. In semantics Rule 3, a policy rule R evaluates a request Rq to Permit if the target matches with the request elements (see details in semantics Rule 1) and rule conditions RC evaluate to True and rule effect RE is Permit. In semantics Rule 4, a policy rule R evaluates a request Rq to Deny if the target matches with the request elements (see details in semantics Rule 1) and rule conditions RC evaluate to True and rule effect RE is Deny.
In semantics Rule 5, a policy rule $R$ evaluates a request $Rq$ to $NotApplicable$ if either the target does not match with the request elements (see details in semantics Rule 2) or rule conditions $RC$ evaluate to $False$.

### 3.4.3 Policy Evaluation

In this section, we present the evaluation semantics rules for a request $Rq$ at the policy level.

Table 3: Evaluation Semantics Rules of a Policy (RCA=Permit-Overrides)

<table>
<thead>
<tr>
<th>Rule Number</th>
<th>Rule Description</th>
</tr>
</thead>
</table>
| 6           | $\text{(RCA = Permit − Overrides)} \land$ \[
\begin{align*}
&(< TR, Rq > \vdash \text{match True}) \land (\exists R \in SR; < R, Rq > \xrightarrow{\text{eval}} \text{Permit}) \\
&\quad \implies < P, Rq > \xrightarrow{\text{eval}} \text{Permit}, OBLs
\end{align*}
\] |
| 7           | $\text{(RCA = Permit − Overrides)} \land$ \[
\begin{align*}
&(< TR, Rq > \vdash \text{match True}) \land (\forall R \in SR; < R, Rq > \xrightarrow{\text{eval}} \text{Deny}) \\
&\quad \implies < P, Rq > \xrightarrow{\text{eval}} \text{Deny}, OBLs
\end{align*}
\] |
| 8           | $\text{(RCA = Permit − Overrides)} \land$ \[
\begin{align*}
&((< TR, Rq > \vdash \text{match False}) \lor (\forall R \in SR; < R, Rq > \xrightarrow{\text{eval}} \text{NotApplicable})) \\
&\quad \implies < P, Rq > \xrightarrow{\text{eval}} \text{NotApplicable}
\end{align*}
\] |

Rules 6, 7 and 8 in Table 3 describe the cases where the rule combining algorithm (RCA) is $\text{Permit − Overrides}$. In Rule 6, a policy $P$ evaluates a request $Rq$ to $\text{Permit}$ with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule 1) and there exists a rule $R$ in the set of Rules $SR$ that evaluates to $\text{Permit}$ (see details in semantics Rule 3). In Rule 7, a policy $P$ evaluates a request $Rq$ to $\text{Deny}$ with a list of obligations $OBLs$ if the target matches with the request elements (see details in
semantics Rule 1) and all rules in the set of Rules SR that evaluates to Deny (see details in semantics Rule 4). In Rule 8, a policy $P$ evaluates a request $R_q$ to NotApplicable if either the target does not match with the request elements (see details in semantics Rule 2) or all rules in the set of Rules SR that evaluates to NotApplicable (see details in semantics Rule 5).

Rules 9, 10, 11 in Table 4 describe the cases where the rule combining algorithm (RCA) is Deny – Overrides. In Rule 9, a policy $P$ evaluates a request $R_q$ to Deny with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule 1) and there exists a rule $R$ in the set of Rules SR that evaluates to Deny (see details in semantics Rule 4). In rule 10, a policy $P$ evaluates a request $R_q$ to Deny with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule 1) and all rules in the set of Rules SR that evaluates to Permit (see details in semantics

<table>
<thead>
<tr>
<th>Table 4: Evaluation Semantics Rules of a Policy (RCA=Deny-Overrides)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(RCA = \text{Deny} - \text{Overrides}) \land$</td>
</tr>
<tr>
<td>$(&lt; TR, R_q &gt; \vdash_{\text{match}} \text{True}) \land (\exists R \in SR; &lt; R, R_q &gt; \xrightarrow{\text{eval}} \text{Deny})$</td>
</tr>
<tr>
<td>$&lt; P, R_q &gt; \xrightarrow{\text{eval}} \text{Deny}, OBLs$</td>
</tr>
<tr>
<td>(Rule 9)</td>
</tr>
<tr>
<td>$(RCA = \text{Deny} - \text{Overrides}) \land$</td>
</tr>
<tr>
<td>$(&lt; TR, R_q &gt; \vdash_{\text{match}} \text{True}) \land (\forall R \in SR; &lt; R, R_q &gt; \xrightarrow{\text{eval}} \text{Permit})$</td>
</tr>
<tr>
<td>$&lt; P, R_q &gt; \xrightarrow{\text{eval}} \text{Permit}, OBLs$</td>
</tr>
<tr>
<td>(Rule 10)</td>
</tr>
<tr>
<td>$(RCA = \text{Deny} - \text{Overrides}) \land$</td>
</tr>
<tr>
<td>$((&lt; TR, R_q &gt; \vdash_{\text{match}} \text{False}) \lor (\forall R \in SR; &lt; R, R_q &gt; \xrightarrow{\text{eval}} \text{NotApplicable}))$</td>
</tr>
<tr>
<td>$&lt; P, R_q &gt; \xrightarrow{\text{eval}} \text{NotApplicable}$</td>
</tr>
<tr>
<td>(Rule 11)</td>
</tr>
</tbody>
</table>
Rule 3). In rule 11, a policy \( P \) evaluates a request \( Rq \) to \textit{NotApplicable} if either the target does not match with the request elements (see details in semantics Rule 2) or all rules in the set of Rules \( SR \) that evaluates to \textit{NotApplicable} (see details in semantics Rule 5).

Table 5: Evaluation Semantics Rules of a Policy (RCA=First-Applicable)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RCA = \textit{First - Applicable}) \land \langle TR, Rq \rangle \dashv \text{match} \land (\exists R1, R2 \in SR: ((PR = R1 &gt; R2) \land (&lt; R1, Rq &gt; eval \text{Permit})) \land \langle P, Rq \rangle eval \text{Permit}, OBLs)</td>
<td>(Rule 12)</td>
</tr>
<tr>
<td>(RCA = \textit{First - Applicable}) \land \langle TR, Rq \rangle \dashv \text{match} \land (\exists R1, R2 \in SR: ((PR = R1 &gt; R2) \land (&lt; R1, Rq &gt; eval \text{Deny})) \land \langle P, Rq \rangle eval \text{Deny}, OBLs)</td>
<td>(Rule 13)</td>
</tr>
<tr>
<td>(RCA = \textit{First - Applicable}) \land ((&lt; TR, Rq &gt; \dashv \text{match} False) \lor (\forall R \in SR; &lt; R, Rq &gt; eval \text{NotApplicable}) \land \langle P, Rq \rangle eval \text{NotApplicable})</td>
<td>(Rule 14)</td>
</tr>
</tbody>
</table>

Rules 12, 13 and 14 in Table 5 describe the cases where the rule combining algorithm (RCA) is \textit{First - Applicable}. In Rule 12, a policy \( P \) evaluates a request \( Rq \) to \textit{Permit} with a list of obligations \textit{OBLs} if the target matches with the request elements (see details in semantics Rule 1) and there exists two rules \( R1 \) and \( R2 \) in the set of rules \( SR \) such that the precedence \( PR = R1 > R2 \) and \( R1 \) evaluates to \textit{Permit} (see details in semantics Rule 3). In Rule 13, a policy \( P \) evaluates a request \( Rq \) to \textit{Deny} with a list of obligations \textit{OBLs} if the target matches with the request elements (see details in semantics Rule 1) and
there exists two rules $R_1$ and $R_2$ in the set of rules $SR$ such that the precedence $PR = R_1 > R_2$ and $R_1$ evaluates to $Deny$ (see details in semantics Rule 4). In Rule 13, a policy $P$ evaluates a request $Rq$ to $NotApplicable$ if either the target does not match with the request elements (see details in semantics Rule 2) or all rules in the set of Rules $SR$ that evaluates to $NotApplicable$ (see details in semantics Rule 5).

### 3.4.4 PolicySet Evaluation

In this section, we present the evaluation rules for a request $Rq$ at the policy set level.

**Table 6: Evaluation Semantics Rules of a PolicySet (PCA=Permit-Overrides)**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>$(PCA = Permit − Overrides) \land (\langle TR, Rq \rangle \vdash \text{match} \ True) \land (\exists P \in SP; \langle P, Rq \rangle \xrightarrow{eval} \text{Permit}) \quad \text{(Rule 15)}$</td>
</tr>
<tr>
<td></td>
<td>$\langle PS, Rq \rangle \xrightarrow{eval} \text{Permit, OBLs}$</td>
</tr>
<tr>
<td>16</td>
<td>$(PCA = Permit − Overrides) \land (\langle TR, Rq \rangle \vdash \text{match} \ True) \land (\forall P \in SP; \langle P, Rq \rangle \xrightarrow{eval} \text{Deny}) \quad \text{(Rule 16)}$</td>
</tr>
<tr>
<td></td>
<td>$\langle PS, Rq \rangle \xrightarrow{eval} \text{Deny, OBLs}$</td>
</tr>
<tr>
<td>17</td>
<td>$(PCA = Permit − Overrides) \land (\langle TR, Rq \rangle \vdash \text{match} \ False) \lor (\forall P \in SP; \langle P, Rq \rangle \xrightarrow{eval} \text{NotApplicable}) \quad \text{(Rule 17)}$</td>
</tr>
<tr>
<td></td>
<td>$\langle PS, Rq \rangle \xrightarrow{eval} \text{NotApplicable}$</td>
</tr>
</tbody>
</table>

Rules 15, 16 and 17 in Table 6 describe the cases where the policy combining algorithm $(PCA)$ is $Permit − Overrides$. In Rule 15, a policy set $PS$ evaluates a request $Rq$ to $Permit$ with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule 1) and there exists a policy $P$ in the set of policies $SP$ that
evaluates to Permit (see details in semantics Rules 6, 10, 12). In Rule 16, a policy set $PS$ evaluates a request $Rq$ to Deny with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule 1) and all policies in the set of policies $SP$ that evaluates to Deny (see details in semantics Rules 7, 9, 13). In Rule 17, a policy set $PS$ evaluates a request $Rq$ to NotApplicable if either the target does not match with the request elements (see details in semantics Rule 2) or all policies in the set of policies $SP$ that evaluates to NotApplicable (see details in semantics Rules 8, 11, 14).

Table 7: Evaluation Semantics Rules of a PolicySet (PCA=Deny-Overrides)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 18</td>
<td>$(PCA = \text{Deny} - \text{Overrides}) \land (\text{match}_\text{True} \land (\exists P \in SP; &lt; P, Rq &gt; \overset{\text{eval}}{\longrightarrow} \text{Deny})) \implies &lt; PS, Rq &gt; \overset{\text{eval}}{\longrightarrow} \text{Deny}, OBLs$</td>
</tr>
<tr>
<td>Rule 19</td>
<td>$(PCA = \text{Deny} - \text{Overrides}) \land (\text{match}_\text{True} \land (\forall P \in SP; &lt; P, Rq &gt; \overset{\text{eval}}{\longrightarrow} \text{Permit}) \implies &lt; PS, Rq &gt; \overset{\text{eval}}{\longrightarrow} \text{Permit}, OBLs$</td>
</tr>
<tr>
<td>Rule 20</td>
<td>$(PCA = \text{Deny} - \text{Overrides}) \land ((\text{match}_\text{False} \lor (\forall P \in SP; &lt; P, Rq &gt; \overset{\text{eval}}{\longrightarrow} \text{NotApplicable})) \implies &lt; PS, Rq &gt; \overset{\text{eval}}{\longrightarrow} \text{NotApplicable}$</td>
</tr>
</tbody>
</table>

Rules 18, 19 and 20 in Table 7 describe the cases where the policy combining algorithm (PCA) is Deny – Overrides. In Rule 18, a policy set $PS$ evaluates a request $Rq$ to Deny with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule 1) and there exists a policy $P$ in the set of policies $SP$ that evaluates to Deny (see details in semantics Rules 7, 9, 13). In Rule 19, a policy set $PS$ evaluates a request $Rq$...
to *Permit* with a list of obligations *OBLs* if the target matches with the request elements (see details in semantics Rule 1) and all policies in the set of policies *SP* that evaluates to *Permit* (see details in semantics Rules 6, 10, 12). In Rule 20, a policy set *PS* evaluates a request *Rq* to *NotApplicable* if either the target does not match with the request elements (see details in semantics Rule 2) or all policies in the set of policies *SP* that evaluates to *NotApplicable* (see details in semantics Rules 8, 11, 14).

Table 8: Evaluation Semantics Rules of a PolicySet (PCA=First-Applicable)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.1</td>
<td></td>
</tr>
</tbody>
</table>
\[
(PCA = First - Applicable) \land \\
(< TR, Rq > \vdash True) \land \\
(\exists P_1, P_2 \in SP; ((PR = P_1 > P_2) \land (< P_1, Rq > \rightarrow_{eval} Permit)))
\]

< PS, Rq > \rightarrow_{eval} Permit, OBLs |

| 22.1 | 
\[
(PCA = First - Applicable) \land \\
(< TR, Rq > \vdash True) \land \\
(\exists P_1, P_2 \in SP; ((PR = P_1 > P_2) \land (< P_1, Rq > \rightarrow_{eval} Deny)))
\]

< PS, Rq > \rightarrow_{eval} Deny, OBLs |

| 23.1 | 
\[
(PCA = First - Applicable) \land \\
((< TR, Rq > \vdash False) \lor (\forall P \in SP; < P, Rq > \rightarrow_{eval} NotApplicable))
\]

< PS, Rq > \rightarrow_{eval} NotApplicable |

Rules 21, 22 and 23 in Table 8 describe the cases where the policy combining algorithm *(PCA)* is *First – Applicable*. In Rule 21, a policy set *PS* evaluates a request *Rq* to *Permit* with a list of obligations *OBLs* if the target matches with the request elements (see details in semantics Rule 1) and there exists a policy *P1* and *P2* in the set of policies *SP* such that
the precedence order $PR = P1 > P2$ and $P1$ evaluates to $Permit$ (see details in semantics Rules 6, 10, 12). In Rule 22, a policy set $PS$ evaluates a request $Rq$ to $Permit$ with a list of obligations $OBLs$ if the target matches with the request elements (see details in semantics Rule 1) and all policies in the set of policies $SP$ that evaluates to $Deny$ (see details in semantics Rules 7, 9, 13). In Rule 23, a policy set $PS$ evaluates a request $Rq$ to $NotApplicable$ if either the target does not match with the request elements (see details in semantics Rule 2) or all policies in the set of policies $SP$ that evaluates to $NotApplicable$ (see details in semantics Rules 8, 11, 14).

Table 9: Evaluation Semantics Rules of a PolicySet (PCA=Only-one-Applicable)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Evaluation Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 24</td>
<td>$(PCA = Only - one - Applicable) \land \leftarrow&lt; TR, Rq &gt; \vdash match True \land (\exists P \in SP; &lt; P, Rq &gt; \rightarrow eval Permit) \rightarrow eval PS, Rq \rightarrow eval Permit, OBLs$</td>
</tr>
<tr>
<td>Rule 25</td>
<td>$(PCA = Only - one - Applicable) \land \leftarrow&lt; TR, Rq &gt; \vdash match True \land (\exists P \in SP; &lt; P, Rq &gt; \rightarrow eval Deny) \rightarrow eval PS, Rq \rightarrow eval Deny, OBLs$</td>
</tr>
<tr>
<td>Rule 26</td>
<td>$(PCA = Only - one - Applicable) \land \leftarrow&lt; TR, Rq &gt; \vdash match False \lor (\forall P \in SP; &lt; P, Rq &gt; \rightarrow eval NotApplicable) \rightarrow eval PS, Rq \rightarrow eval NotApplicable$</td>
</tr>
<tr>
<td>Rule 27</td>
<td>$(PCA = Only - one - Applicable) \land \leftarrow&lt; TR, Rq &gt; \vdash match True \land (\exists P1, P2 \in SP; (&lt; P1, Rq &gt; \rightarrow eval Deny \lor Permit) \land (&lt; P2, Rq &gt; \rightarrow eval Deny \lor Permit)) \rightarrow eval PS, Rq \rightarrow eval Indeterminate$</td>
</tr>
</tbody>
</table>
Rules 24, 25, 26 and 27 in Table 9 describe the cases where the policy combining algorithm (PCA) is \textit{Only \textendash one \textendash Applicable}. In Rule 24, a policy set \( PS \) evaluates a request \( Rq \) to \textit{Permit} with a list of obligations \( OBLs \) if the target matches with the request elements (see details in semantics Rule 1) and there exists one and only policy \( P \) in the set of policies \( SP \) that evaluates to \textit{Permit} (see details in semantics Rules 6, 10, 12). In Rule 25, a policy set \( PS \) evaluates a request \( Rq \) to \textit{Deny} with a list of obligations \( OBLs \) if the target matches with the request elements (see details in semantics Rule 1) and there exists one and only policy \( P \) in the set of policies \( SP \) that evaluates to \textit{Deny} (see details in semantics Rules 7, 9, 13). In Rule 26, a policy set \( PS \) evaluates a request \( Rq \) to \textit{NotApplicable} if either the target does not match with the request elements (see details in semantics Rule 2) or all policies in the set of policies \( SP \) that evaluates to \textit{NotApplicable} (see details in semantics Rules 8, 11, 14). In Rule 27, a policy set \( PS \) evaluates a request \( Rq \) to \textit{Indeterminate} if the target matches with the request elements (see details in semantics Rule 1) and there exists at least two policies \( P1 \) and \( P2 \) in the set policies \( SP \) such that \( P1 \) evaluates to either \textit{Permit} or \textit{Deny} and \( P2 \) evaluates to either \textit{Permit} or \textit{Deny}.

3.5 Policy Evaluation Algorithms

In this section, we present the algorithms realizing the SBA-XACML policy evaluation semantics. We divided the evaluation module into three algorithms. Each one of them evaluates the request at a separate layer in the based policy. The rule evaluation algorithm is presented in Algorithm 1, the policy evaluation algorithm in Algorithm 2 and the policy set evaluation algorithm in Algorithm 3.
3.5.1 Rule Evaluation Algorithm

The rule evaluation algorithm in Algorithm 1 evaluates the request at the lowest level in the policy set. It takes two inputs: a rule \( R \) and a request \( R_q \). The output is the rule decision which is \( \text{Deny} \), \( \text{Permit} \), or \( \text{NotApplicable} \).

**Algorithm 1 Rule_Evaluation(\( \hat{R}, R_q \))**

**Input**: 1) A Rule \( R \) with Target \( TR = \{S,R,A\} \) and (2) A Request \( R_q = \{Sr,Rr,Ar\} \)

**Output**: Rule decision \( \in \{\text{RE}, \text{NotApplicable}\} \) where \( \text{RE} \in \{\text{Permit, Deny}\} \)

1: // Rule Applicability Check
2: \text{if} \ ((Sr \cap S) \neq \emptyset) \land ((Rr \cap R) \neq \emptyset) \land ((Ar \cap A) \neq \emptyset) \text{ then}
3: // loop through Rule Conditions
4: \text{for} \ i := 1 \text{ to } m \text{ do}
5: //At least one Rule Condition evaluated to false
6: \text{if} \ \exists \ i, 0 < i \leq m, RC = "false" \text{ then}
7: \text{return} \ NotApplicable;
8: \text{else}
9: //All Rule Conditions evaluated to true
10: \text{return} \ RE;
11: \text{end if}
12: \text{end for}
13: \text{else}
14: //Rule NotApplicable to the request
15: \text{return} \ NotApplicable;
16: \text{end if}

The Rule Evaluation in Algorithm 1 takes two inputs: a rule \( R \) and a request \( R_q \). It begins by checking whether the rule is applicable to the request (line 2). The Applicability check is done by comparing the request set of subjects, set of resources and set of actions against the rule target. If the applicability check returns true then the rule conditions are evaluated (line 6), otherwise "NotApplicable" is returned to the Policy Evaluation in Algorithm 2 (line 15). Rule effect is returned if all rule conditions evaluate to true (line 10) otherwise "NotApplicable" is returned to the Policy Evaluation in Algorithm 2 (line 7).
3.5.2 Policy Evaluation Algorithm

The policy evaluation algorithm in Algorithm 2 evaluates the request at the middle layer. It calls the rule evaluation Algorithm 1 to handle the evaluation at the lower layer. It takes two inputs: a policy \( P \) and a request \( Rq \). The output is the policy decision which is \( Deny \), \( Permit \), or \( NotApplicable \).

The policy evaluation algorithm in Algorithm 2 takes two inputs: a policy \( P \) and a request \( Rq \). The algorithm is composed of three steps: the applicability check of the policy, the evaluation of rules and rule combining algorithm \( RCA \). The applicability check is done by matching the request subjects, resources and actions with policy target (line 1). If (line 1) returns true, we call step 2, otherwise \( NotApplicable \) is returned to the policy set evaluation in Algorithm 3. The evaluation of rules is done by the order they are listed in the policy. The rule evaluation algorithm Algorithm 1 is called in line 3. The response returned is passed to step 3, where \( RCA \) can have one of the following values: \( Permit - Overrides \), \( Deny - Overrides \) or \( First - Applicable \). If the \( RCA \) is \( Permit - Overrides \) and step 2 returns one \( Permit \), then the returned response is \( Permit \). If \( RCA \) is \( Permit - Overrides \) and step 2 returns \( Deny \) for all rules, then the returned response is \( Deny \). If \( RCA \) is \( Permit - Overrides \) and step 2 returns \( NotApplicable \) for all rules, then the returned response is \( NotApplicable \). If \( RCA \) is \( Deny - Overrides \) and step 2 returns one \( Deny \) then the returned response is \( Deny \). If \( RCA \) is \( Deny - Overrides \) and step 2 returns \( Permit \) for all rules, then the returned response is \( Permit \). If \( RCA \) is \( Deny - Overrides \) and step 2 returns \( NotApplicable \) for all rules, then the returned response is \( NotApplicable \). If \( RCA \) is \( First - Applicable \) and step 2 returns one \( Deny \) or \( Permit \), then the returned
Algorithm 2 Policy_Evaluation\((P, Rq)\)

**Input**: 1) A Policy \(P\) with Target \(TR = \{S, R, A\}\) and (2) A Request \(Rq = \{Sr, Rr, Ar\}\)

**Output**: Policy decision \(\in\{\text{Deny, Permit, NotApplicable}\}\)

1: \textbf{if} \((S \cap S) \neq \emptyset) \land ((R \cap R) \neq \emptyset) \land ((A \cap A) \neq \emptyset)\) then
2: \hspace{1em} \textbf{for} \(i := 1\) to \(m\) do
3: \hspace{2em} // Call Rule Evaluation Algorithm
4: \hspace{3em} RE\(_i\) =
5: \hspace{4em} \textbf{if} (RCA = "Deny-Overrides") then
6: \hspace{5em} // if at least one rule evaluates to Deny
7: \hspace{6em} if \(\exists i, 0 < i \leq m, RE_i = "Deny"\) then
8: \hspace{7em} \textbf{return} Deny;
9: \hspace{5em} \textbf{else}\n10: \hspace{6em} // If all rules evaluate to permit
11: \hspace{7em} if \(\forall i, 0 < i \leq m, RE_i = "Permit"\) then
12: \hspace{8em} \textbf{return} Permit;
13: \hspace{6em} \textbf{else}
14: \hspace{7em} \textbf{return} NotApplicable;
15: \hspace{5em} \textbf{end if}
16: \hspace{4em} \textbf{end if}
17: \hspace{3em} \textbf{end if}
18: \hspace{2em} \textbf{if} (RCA = "Permit-Overrides") then
19: \hspace{3em} // At least one rule evaluates to Permit
20: \hspace{4em} if \(\exists i, 0 < i \leq m, RE_i = "Permit"\) then
21: \hspace{5em} \textbf{return} Permit;
22: \hspace{3em} \textbf{else}\n23: \hspace{4em} // All deny
24: \hspace{5em} if \(\forall i, 0 < i \leq m, RE_i = "Deny"\) then
25: \hspace{6em} \textbf{return} Deny;
26: \hspace{5em} \textbf{else}
27: \hspace{6em} \textbf{return} NotApplicable;
28: \hspace{4em} \textbf{end if}
29: \hspace{3em} \textbf{end if}
30: \hspace{2em} \textbf{if} (RCA = "First-Applicable") then
31: \hspace{3em} //First Applicable rule evaluates to permit
32: \hspace{4em} if \(\exists i, 0 < i \leq m, RE_i = "Permit"\) then
33: \hspace{5em} \textbf{return} Permit;
34: \hspace{3em} \textbf{else}\n35: \hspace{4em} //First Applicable rule evaluates to deny
36: \hspace{5em} if \(\exists i, 0 < i \leq m, RE_i = "Deny"\) then
37: \hspace{6em} \textbf{return} Deny;
38: \hspace{5em} \textbf{else}
39: \hspace{6em} \textbf{return} NotApplicable;
40: \hspace{4em} \textbf{end if}
41: \hspace{3em} \textbf{end if}
42: \hspace{2em} \textbf{end if}
43: \hspace{1em} \textbf{end for}
44: \hspace{1em} \textbf{else}
45: \hspace{2em} //Policy is not Applicable by target
46: \hspace{3em} \textbf{return} NotApplicable;
47: \hspace{1em} \textbf{end if}
response is *Deny* or *Permit* respectively. If RCA is *First—Applicable* and step 2 returns *NotApplicable* for all rules, then the returned response is *NotApplicable*.

### 3.5.3 PolicySet Evaluation Algorithm

The policy set evaluation algorithm in Algorithm 3 calls the policy evaluation algorithm Algorithm 2 to handle the evaluation at the middle layer. The algorithm takes two inputs: a policy set $PS$ and a request $Rq$. The output is the final response to the request $Rs$.

The PolicySet Evaluation in Algorithm 3 takes two inputs: a policy set $PS$ and a request $Rq$. The algorithm is composed of three steps: the applicability check of the policy set, the evaluation of policies and policy combining algorithm $PCA$. The applicability check is done by evaluating the request subjects, resources and actions with the policy set target in line 2. If it returns true, then step 2 is called otherwise $Rs = NotApplicable$ is returned as the request response. The evaluation of policies is done by the order they are listed in the based policy. The policy evaluation algorithm Algorithm 2 is called in line 6. The returned response is passed to step 3, where the $PCA$ can have one of the following values: *Permit—Overrides, Deny—Overrides, First—Applicable* or *Only—one—Applicable*. If the $PCA$ is *Permit — Overrides* and step 2 returns one *Permit*, then the returned response is *Permit*. If the $PCA$ is *Permit — Overrides* and step 2 returns *Deny* for all policies, then the returned response is *Deny*. If the $PCA$ is *Permit — Overrides* and step 2 returns *NotApplicable* for all policies, then the returned response is *NotApplicable*. If the $PCA$ is *Deny — Overrides* and step 2 returns one *Deny*, then the returned response is *Deny*. If the $PCA$ is *Deny — Overrides* and step 2 returns *Permit* for all policies,
Algorithm 3 PolicySet_Evaluation(PS, Rq) Part 1

**Input**: (1) A PolicySet PS with Target TR = {S,R,A} and (2) Request Rq = {Sr,Rr,Ar}

**Output**: Request response Rs ∈ {Permit, Deny, NotApplicable, Indeterminate}

1: // PolicySet Applicability Check
2: if \((Sr \cap S) \neq \emptyset \land (Rr \cap R) \neq \emptyset \land (Ar \cap A) \neq \emptyset\) then
3:  // loop through Policies
4: for \(i := 1 \text{ to } m\) do
5:  // Call Policy Evaluation function
6:  PEi = POLICY_EVALUATION(Pi, Rq);
7:  if (PCA = "Deny-Overrides") then
8:    // if at least one policy evaluates to Deny
9:    if \(\exists i, 0 < i \leq m, \text{PEi} = \text{"Deny"}\) then
10:      return Deny;
11:    else
12:      // If all policies evaluate to Permit
13:      if \(\forall i, 0 < i \leq m, \text{PEi} = \text{"Permit"}\) then
14:        return Permit;
15:      else
16:        return NotApplicable;
17:      end if
18:    end if
19:  end if
20: if (PCA = "Permit-Overrides") then
21:  // if at least one policy evaluates to Permit
22:  if \(\exists i, 0 < i \leq m, \text{PEi} = \text{"Permit"}\) then
23:    return Permit;
24:  else
25:    // If all policies evaluate to Deny
26:    if \(\forall i, 0 < i \leq m, \text{PEi} = \text{"Deny"}\) then
27:      return Deny;
28:    else
29:      return NotApplicable;
30:  end if
31: end if
32: if (PCA = "First-Applicable") then
33:  // if at least one policy evaluates to Deny
34:  if \(\exists i, 0 < i \leq m, \text{PEi} = \text{"Deny"}\) then
35:    return Deny;
36:  else
37:    // if at least one Policy evaluates to Permit
38:    if \(\exists i, 0 < i \leq m, \text{PEi} = \text{"Permit"}\) then
39:      return Permit;
40:    else
41:      return NotApplicable;
42:  end if
43: end if
44: end if
Algorithm 3 PolicySet_Evaluation(PS, Rq) Part 2

46: //PCA = Only-one-Applicable
47: if (PCA = "Only-one-Applicable") then
48:   // iff one policy evaluates to Deny
49:   if \( \exists \ i, 0 < i \leq m, \text{PEi} = "Deny" \) then
50:     return Deny;
51:   end if
52:   // Iff one Policy evaluates to permit
53:   if \( \exists \ i, 0 < i \leq m, \text{PEi} = "Permit" \) then
54:     return Permit;
55:   end if
56:   // If all policies evaluate to NotApplicable
57:   if \( \forall \ i, 0 < i \leq m, \text{PEi} = "NotApplicable" \) then
58:     return NotApplicable;
59:   else
60:     // More than one applicable policies
61:     return Indeterminate;
62:   end if
63: end if
64: end for
65: else
66:   // PolicySet NotApplicable to the request
67:   return NotApplicable;
68: end if

then the returned response is Permit. If the PCA is Deny – Overrides and step 2 returns NotApplicable for all policies, then the returned response is NotApplicable. If the PCA is First – Applicable and step 2 returns one Deny or Permit, then the returned response is Deny or Permit respectively. If the PCA is First – Applicable and step 2 returns NotApplicable for all policies, then the returned response is NotApplicable. If the PCA is Only – One – Applicable and only one policy is applicable, the response is either Deny or Permit, otherwise the response is Indeterminate.
3.6 Case Study: SBA-XACML Policy Evaluation and Performance Analysis

In this section, we first present a case study illustrating the usability of SBA-XACML policy evaluation process through semantics rules and experiments. Then, we provide the results of the performance analysis comparing our results to the current approaches. We will be utilizing the XACML example presented in Chapter Two, Section 2.6.3.

3.6.1 SBA-XACML Policy Evaluation

In this section, we provide the generated SBA-XACML based policy and request, and the response of the evaluation process according to the SBA-XACML policy evaluation semantics in Section 3.4. Listing 3.1 contains the generated SBA-XACML based policy corresponding to the XACML one in Listings 2.3 and 2.4 of Chapter Two, Section 2.6.3.

Listing 3.1: SBA-XACML Policy for a Bank Service

```
[1].PS::=<PS1,{P1,P2},{P1>P2},{Permit-overrides},{},{{},{{},{{}}}}>
[2].P::=<P1,{R1,R2},{R1>R2},{Permit-overrides},{}{Withdraw, Permit, {mailto, string,
Customer_service@bank.com}, {subject-id, string, subject-id}, {resource-id, string, resource
-id}}},{{Jerry, Joe}, {BankService/withdraw}, {Any}}>
[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, {ResourceAttributeDesignator, string, BankService/deposit
}},{ },{ },{},{{Permit}>
[7].R::=<R4, {and,{string-equal, {ResourceAttributeDesignator, string, BankService/deposit
}},{string-equal, {SubjectAttributeDesignator, subject-id, string, Joe}}}),{ },{ },{},{{
Permit}}>
[8].R::=<R5, {and,{string-equal, {ResourceAttributeDesignator, string, BankService/deposit
}},{string-equal, {SubjectAttributeDesignator, subject-id, string, Joe}}}),{ },{ },{},{Deny}>
```

Line 1 is the policy set $PS$. The policy set ID is $PS1$. It has two policies $P1$ and $P2$. $P1$ is ordered before $P2$. The policy combining algorithm is $Permit – Overrides$. 

62
PS1Bank 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 Withdraw policy. 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 one obligation to perform and the target subjects are Bob and Jerry, resource is BankService/withdraw and actions are any. Line 3 is the rule R1. The rule ID is R1. R1 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. R1 has a permit effect. Line 4 is the rule R2. The rule ID is R2. R2 has no conditions. R2 has no target specified. R2 has a deny effect. Line 5 is the deposit policy. The policy ID is P2. It has three rules R3, R4 and R5. The precedence order is R3, R4 and R5. The rule combining algorithm is permit – overrides. P2 has no obligation to perform and the target elements are not defined. Line 6 is the rule R3. The rule ID is R3. R3 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. R3 has a permit effect. Line 7 is the rule R4. The rule ID is R4. R4 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. R4 has a permit effect. Line 8 is the rule R5. The rule ID is R5. R5 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. R5 has a deny effect.

Listing 3.2 contains the generated SBA-XACML request corresponding to one in Listing 2.5 of Chapter Two. The request subject is equal to Bob, resource equal
BankService/Deposit and action equal execute.

Listing 3.2: SBA-XACML Access Request

\[ [1].Rq1::=\langle\{\text{Bob}\}, \{\text{BankService/Deposit}\}, \{\text{execute}\} \rangle \]

Based on the SBA-XACML policy evaluation semantics in Section 3.4 and its implemented algorithms in Section 3.5, the elaborated framework will evaluate the request \text{Rq1} in Listing 3.2 with respect to the based policy \text{PS1} presented in Listing 3.1 generated from Listings 2.3 and 2.4. Since the evaluation of each semantics rule is based on evaluating its premises, we will describe the evaluation steps in order by the premises of policy sets, policies and rules as summarized in Table 10. To avoid repetition and for space limitation, we will present only the matching semantics rules that affect the final decision. The rules in Table 10 should be read from bottom to top as follows:

(1) The based policy is composed of a PolicySet \text{PS1}. It has \text{PCA} = \{\text{Permit-Overrides}\}, its target \text{TR} matches request \text{Rq1} as illustrated in (2) and it has a policy \text{P2} that evaluates to \text{Permit} as depicted in (3). Hence, based on the semantics Rule 15 that applies in this case, all the three premises are satisfied and the final decision is \text{Permit}.

(2) \text{PS1} has no target defined which means \text{TR} = \{\} or \text{TR} = \{S = \text{Any} , R = \text{Any}, A = \text{Any}\}. \text{Rq1} = \{Sr = \text{Bob} , Rr = \text{BankService/Deposit}, Ar = \text{execute}\}. By applying semantics Rule 1, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore \text{PS1} matches the request \text{Rq1}.

(3) Policy \text{P2} is composed of three rules. It has \text{RCA} = \text{Permit-Overrides}, its target \text{TR} matches with the target of request \text{Rq1} as illustrated in (4) and it has a rule \text{R3} that
Table 10: Results of Semantics-Based Policy Evaluation

<table>
<thead>
<tr>
<th>Step</th>
<th>Rule</th>
<th>Premise</th>
<th>Match</th>
<th>Eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>(6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*evaluates to Permit as depicted in (5). Hence, based on the semantics Rule 6 that applies in this case, all the three premises are satisfied and the evaluation of P2 with respect to Rq1 is Permit.*

**P2** has no target TR defined which means TR = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule 1, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore P2 matches the request Rq1.

**P2** has no target TR defined which means TR = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule 1, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore P2 matches the request Rq1.

**P2** has no target TR defined which means TR = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule 1, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore P2 matches the request Rq1.

**P2** has no target TR defined which means TR = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule 1, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore P2 matches the request Rq1.

(4) P2 has no target TR defined which means TR = {S = Any, R = Any, A = Any}. Rq1 = {Sr = Bob, Rr = BankService/Deposit, Ar = execute}. By applying semantics Rule 1, Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore P2 matches the request Rq1.

(5) The target TR of R3 matches with the target of request Rq1 as illustrated in (6). R3 has
one rule condition $RC = \{\text{string-equal,\{RAD, string, BankService/deposit\}}\}$, which means the resource requesting access must be equal to BankService/Deposit. $RC = True$ because the Resource $R$ of $Rq1$ is equal to BankService/Deposit. The rule effect $RE$ of $R3$ is Permit ($RE=\text{Permit}$). Hence, based on the semantics Rule 3 that applies in this case, all the three premises are satisfied and the evaluation of $R3$ with respect to $Rq1$ is $\text{Permit}$.

(6) $R3$ has no target defined which means $TR = \{S = \text{Any}, R = \text{Any}, A = \text{Any}\}$. $Rq1 = \{Sr = Bob, Rr = \text{BankService/Deposit}, Ar = \text{execute}\}$. By applying semantics Rule(1), Bob is a subset of Any, BankService/Deposit is a subset of Any and execute is a subset of Any, therefore $R3$ matches the request $Rq1$.

The response to the request in Listing 3.2 against the based policy in Listing 3.1 is presented in Listing 3.3. The evaluation results of our approach always returns the same results given by XACML Sun PDP [18].

Listing 3.3: SBA-XACML Response

\[ \langle \{\text{permit}\},\{\} \rangle \]

3.6.2 Experiments and Performance Analysis

We have implemented the SBA-XACML framework using PHP. Our experiments were carried out on a notebook running Windows XP SP3 with 3.50GB of memory and dual core 2.8GHz Intel processor. The experiments were performed at 100,000 tests each and the average number was calculated and used. They were conducted on both real world and
synthetic policies to show the scalability and performance on very large ones. Synthetic policies are created in such a way that every policy and every rule in the policy set is evaluated to reach the final decision (i.e. taking always the worst case). The Synthetic policy sets range from 400 to 4000 rules which are split evenly over 100 policies. In order to be able to exhaust the entire policy set, we specified (1) a policy combining algorithm Deny-Overrides, (2) rule combining algorithm Deny-Overrides for each policy, (3) the deny rule as the last rule in the policy and (4) non empty target element. Please note that moderate specification (i.e. decision is taking early without checking all the rules) of the synthetic policies will lead to better performance. We compare our proposed framework to the commercial XACML engine Sun PDP [18] and XEngine [12].

The processing time of Sun PDP consists of XACML policy loading, request loading and request evaluation to provide the decision. There is no pre-processing time for Sun PDP. As for XEngine, the pre-processing time consists of policy loading, numericalization and normalization, while processing time consists of request loading, numericalization and evaluation to provide the decision. Regarding our approach, the pre-processing consists of converting policy set from XACML to SBA-XACML, which is optional and executed only once when deploying the policies. The processing time includes (1) accepting a request and converting it to SBA-XACML, (2) loading policies and (3) evaluating the request to providing the decision. We repeated this policy evaluation process for 100,000 different requests with and without the pre-processing procedures and provided the average evaluation time of synthetic and real world policies.

We chose not to use the experiments methodology used by XEngine because it does not reflect real world environments. Their tools and experiments show that all the requests
(i.e. up to 100,000 requests) are received, converted and loaded in the memory at the same
time, then evaluated against the already loaded policies. Again as aforementioned, such
assumption does not always hold since requests can be received from different parties at
variant time-space. Fig. 8 contains three real world policies. We included the number
of rules in each policy set, the average pre-processing time (conversion time) for SBA-
XACML and XEngine, the average processing time for single-valued and multi-valued
requests for SBA-XACML, XEngine and Sun PDP.

<table>
<thead>
<tr>
<th>Policy</th>
<th>#Rules</th>
<th>Conversion Time (msec)</th>
<th>Average Processing Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SBA-XACML</td>
<td>XEngine</td>
</tr>
<tr>
<td>IIIA027</td>
<td>2</td>
<td>20</td>
<td>290</td>
</tr>
<tr>
<td>IIIA028</td>
<td>4</td>
<td>27</td>
<td>313</td>
</tr>
<tr>
<td>Continue-a</td>
<td>298</td>
<td>94</td>
<td>562</td>
</tr>
</tbody>
</table>

Fig. 8: Experimental Results on Real-World XACML Policies

3.6.2.1 Policy Evaluation Experimental Results

In the following, we discuss the experimental results for single-valued and multi-valued
requests on both synthetic and real world policies. Fig. 9 shows the results for synthetic
policy evaluations for single-valued and multi-valued requests. For single-valued requests,
Fig. 9a shows that our approach is faster than both the XEngine and Sun PDP by 3.2 and 8
times respectively for policy sets with 400 rules, and by 2.4 and 2.7 times faster for policy
sets with 4000 rules. For multi-valued requests, Fig. 9b shows that our approach is faster
than both the XEngine and Sun PDP by 3.5 and 9.4 times respectively for policy sets with
400 rules, and by 2.5 and 3.5 times faster for policy sets with 4000 rules.
Fig. 9: Synthetic Policy Evaluation

Fig. 10: Real Policy Evaluation

Fig. 10 shows the results for real world policy evaluation for single-valued and multi-valued requests. Fig. 10a explores that our approach is 8 times faster than XEngine and 39 times faster than the Sun PDP for small policies with less than 10 rules, while it is 7.6 faster than XEngine and 3 times faster than Sun PDP on policies with 300 rules. Fig. 10b shows that our approach is faster by 2.4 times and 6.3 than XEngine and Sun PDP respectively for a policy set with 298 rules, while it is 3.5 times and 15 times faster than XEngine and Sun PDP respectively for small policies with less than 10 rules.

3.6.2.2 Policy Evaluation Experimental Results including Pre-Processing

In the following, we discuss the conversion time from XACML to SBA-XACML with respect to the XEngine conversion time. The conversion time is referred to as pre-processing time in [12]. The conversion procedure is optional in our approach and required only during the policy deployment. However, the XEngine approach requires this step because the
conversion includes numericalization and normalization and it is the core of their proposal to improve the evaluation response time. Fig. 11a and Fig. 11b show the conversion time for synthetic and real world policies respectively. Assuming the Synthetic policies are in XACML, the XEngine pre-processing time consumes 2.3 times more than our approach for a policy set with 4000 rules, and 8.5 times more for a policy set with 400 rules. For real policies, the XEngine requires 8 times more than our approach for policy sets with 298 rules, and 10 times more for policy sets with less than 10 rules.

Fig. 12 shows the results of the overall processing and pre-processing (i.e. conversion) time of single-valued and multi-valued requests against synthetic policies ranging from 400 to 4000 rules. Fig. 12a shows that our approach outperformed the XEngine by 6 times for small policy sets with 400 rules, and by 2 times for large policy sets with 4000 rules. Fig. 12b illustrates that our approach outperformed the XEngine by 6.1 times for small policy sets with 400 rules, and by 2 times for large policy sets with 4000 rules.

Fig. 13 shows the results for the overall processing and pre-processing (i.e. conversion) for single-valued and multi-valued requests against real policies ranging from 2 to 298 rules. Fig. 13a shows that SBA-XACML outperformed the XEngine by 9 times for small policy sets with less than 10 rules and by 4.2 times for medium size policy sets with 300 rules. Fig. 13b illustrates that our approach outperformed the XEngine by 9 times for small...
Fig. 12: Synthetic Policy Evaluation with Policy Conversion

Fig. 13: Real Policy Evaluation with Policy Conversion

policy sets with less than 10 rules, and by 4 times for policy sets with 300 rules.

3.7 Conclusion

In this chapter, we addressed the problems related XACML complexity and real-time decision evaluation efficiency. In the context, we proposed an SBA-XACML language and a model which supports the evaluation of both XACML and SBA-XACML policies. The language is based on sets and it covers all XACML elements and attributes without any compromise or extension. In addition, we provided the semantics and its corresponding algorithms for policy evaluation. We realized and demonstrate the viability of our proposition by (1) Implementing the model and (2) developing a case study and (3) performing the experimental analysis. Our results show that our solution reduces the response by more than 50 percent when compared to Sun PDP [18] and Xengine [12].
Chapter Four

SBA-XACML Analysis

4.1 Introduction

Creating XACML policies is a simple task to do with existing tools but there is no verification process to alert for access flaws. How do we know if the new policies created do not create conflicts and grant access to the wrong users? How do we verify if recently deleted rules or policies do not create access flaws or conflicts? XACML documents can become very large and complex even when you do not have a complex system. As a result, verifying access policies can become an impossible task to do. The only possible solution for XACML to verify if policies are free of access flaws, conflicts and redundancies, is to test the access of every possible scenario. It is not logical to test the entire system every time you add a new rule or delete a policy. The proposed approach enables policy administrators to automatically analyze access policies and detect access flaws, conflicts and redundancies without much efforts and resources.
The rest of the chapter is organized as follows. In Section 4.2, we present the analysis semantics. In Section 4.3, we illustrate the SBA-XACML analysis algorithms. In Section 4.4, we exhibit a case study to show the effectiveness of our proposition. In Section 4.5, we conclude the chapter.

4.2 Policy Analysis Semantics

In this section, we present the formal semantics of a SBA-XACML policy analysis following the above inference rule structure and deductive logic. Given a policy $P$, the analysis report $RP$ is derived by the evaluation $\rightarrow_{eval}$ of all premises combined between each other using designated operators $op$ as follows:

\[
(premise_1) \ op \ (premise_2) \ op \ ... \ op \ (premise_n)
\]

\[
< P1, P2 > \rightarrow_{eval} RP
\]

The policy and rule analysis semantics rules, which constitute the premises in the above rule, have also similar structure and follows the deductive logic until reaching the basic defined premise (i.e. condition). Throughout the rest of the thesis, please note the difference between a semantic rule that express 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. In this context, we start first by defining the $Subset$ and $IntersectionFunction$ semantics rule since it will be used throughout all levels.
4.2.1 Subset & Intersection Function

In this section, we present the subset and intersection semantics rules for SBA-XACML rule \( R_1 \) with subject set \( S_1 \), resource set \( R_1 \) and action set \( A_1 \) and SBA-XACML rule \( R_2 \) with subject set \( S_2 \), resource set \( R_2 \) and action set \( A_2 \). The semantics is determined by comparing the subject set \( S_1 \) with subject set \( S_2 \), resource set \( R_1 \) with resource set \( R_2 \) and action set \( A_1 \) with action set \( A_2 \).

Table 11: Subset Function Semantics Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((S_1 \subseteq S_2) \land (R_1 \subseteq R_2) \land (A_1 \subseteq A_2))</td>
<td>(&lt;(TR_1, TR_2)&gt; \vdash True_{subset})</td>
</tr>
<tr>
<td>2</td>
<td>((S_1 \not\subseteq S_2) \lor (R_1 \not\subseteq R_2) \lor (A_1 \not\subseteq A_2))</td>
<td>(&lt;(TR_1, TR_2)&gt; \vdash False_{subset})</td>
</tr>
</tbody>
</table>

Rules 1 and 2 in Table 11 describe the different cases of subset rules. In Rule 1, A target \( TR_1 \) is a subset of target \( TR_2 \) if target \( TR_1 \) subject set \( S_1 \) is a subset of target \( TR_2 \) subject set \( S_2 \) and target \( TR_1 \) resource set \( R_1 \) is a subset of target \( TR_2 \) resource set \( R_2 \) and target \( TR_1 \) action set \( A_1 \) is a subset of target \( TR_2 \) action set \( A_2 \). In Rule 2, A target \( TR_1 \) is not a subset of target \( TR_2 \) if target \( TR_1 \) subject set \( S_1 \) is not a subset of target \( TR_2 \) subject set \( S_2 \) or target \( TR_1 \) resource set \( R_1 \) is not a subset of target \( TR_2 \) resource set \( R_2 \) or target \( TR_1 \) action set \( A_1 \) is a subset of target \( TR_2 \) action set \( A_2 \).

Rules 3 and 4 in Table 12 describe the different cases of intersection rules. In Rule 3, two targets \( TR_1 \) and \( TR_2 \) intersect if target \( TR_1 \) subject set \( S_1 \) and target \( TR_2 \) subject set \( S_2 \) share common elements and target \( TR_1 \) resource set \( R_1 \) and target \( TR_2 \) resource set \( R_2 \)
### Table 12: Intersection Function Semantics Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>(((S_1 \cap S_2 \neq \emptyset) ) &amp; (((R_1 \cap R_2 \neq \emptyset)) &amp; (((A_1 \cap A_2 \neq \emptyset)))</td>
<td>(&lt; TR_1, TR_2 &gt; \vdash intersect True)</td>
</tr>
<tr>
<td>((S_1 \cap S_2 = \emptyset) \lor ((R_1 \cap R_2 = \emptyset) \lor ((A_1 \cap A_2 = \emptyset))</td>
<td>(&lt; TR_1, TR_2 &gt; \vdash intersect False)</td>
</tr>
</tbody>
</table>

share common elements and target \(TR1\) action set \(A1\) and target \(TR2\) action set \(A2\) share common elements. In Rule 4, two targets \(TR1\) and \(TR2\) do not intersect if target \(TR1\) subject set \(S1\) and target \(TR2\) subject set \(S2\) share no common elements or target \(TR1\) resource set \(R1\) and target \(TR2\) resource set \(R2\) share no common elements or target \(TR1\) action set \(A1\) and target \(TR2\) action set \(A2\) share no common elements.

### 4.2.2 Access Flaw Detection

In this section, we present the policy access flaw analysis semantics rules for SBA-XACML policy set.

Semantics Rules 5, 6, 7 and 8 in Table 13 describe the different access flaw analysis cases for a policy set \(PS\). In Rule 5, two rules \(R1\) and \(R2\) return flaw if \(R2\) target \(TR2\) is a subset of rule \(R1\) target \(TR1\) (see details in semantics Rule 1) and rule \(R2\) rule condition \(RC2\) is a subset of rule \(R1\) rule condition \(RC1\) and both rules \(R1\) and \(R2\) have the same rule effect \(RE1\) equal \(RE2\). In Rule 6, for every pair of rules \(R1\) and \(R2\) in policy \(P\) such that \(R1\) and \(R2\) are appended to the Flaw Set \(FS\) if \(R1\) and \(R2\) evaluate to flaw (see details in semantics rule 5). In Rule 7, given a pair of policies \(P1, P2\) in policy set \(PS\), \(P1\) and \(P2\) are appended
Table 13: Rules of Access Flaw Detection Semantics

\[
\begin{align*}
\langle TR2, TR1 \rangle \vdash & \text{subset True} \land (RC2 \subseteq RC1) \land (RE1 = RE2) \\
& \Rightarrow \text{Flaw}_{R1, R2} \\
(\forall R1, R2 \in SR; FS \leftarrow FS \cup \langle R1, R2 \rangle \Rightarrow \text{Flaw}_{R1, R2})
\end{align*}
\]

(Rule 5)

\[
\begin{align*}
\langle P1, P2 \rangle \Rightarrow & \text{Flaw}_{P1, P2} \\
(\forall P \in SP; FS \leftarrow FS \cup \langle P \rangle \Rightarrow \text{Flaw}_{P1, P2}) \cup (\forall P1, P2 \in SP; FS \leftarrow FS \cup \langle P1, P2 \rangle \Rightarrow \text{Flaw}_{P1, P2})
\end{align*}
\]

(Rule 6)

\[
\begin{align*}
\langle PS \rangle \Rightarrow & \text{Flaw}_{PS, FA} \\
(\forall P \in SP; FS \leftarrow FS \cup \langle P \rangle \Rightarrow \text{Flaw}_{PS, FA})
\end{align*}
\]

(Rule 7)

\[
\begin{align*}
& \Rightarrow \text{Flaw}_{R1, R2} \\
(\forall R1, R2 \in SR; FS \leftarrow FS \cup \langle R1, R2 \rangle \Rightarrow \text{Flaw}_{R1, R2})
\end{align*}
\]

(Rule 8)

to the Flaw Set FS if the rule combining of P1 is equal to the rule combining of P2 and the targets of P1 and P2 intersect (see details in semantics Rule 3) and there exists R1 in P1 and R2 in P2 such that R1 and R2 evaluate to flaw and appended to the Flaw Set FS (see details in semantics Rule 5). In Rule 8, given policy set PS, the Flaw Set FS is the union of all flaws between any two flawed rules R1, R2 in one policy and between two flawed rules R1, R2 from two different policies and every two flawed policies P1 and P2. (see details in semantics rule 6,7).

4.2.3 Redundancy Detection

In this section, we present the policy Redundant analysis semantics rules for SBA-XACML policy set.

Semantics Rules 9,10,11 and 12 in Table 14 describe the different redundant analysis cases for a policy set PS. In Rule 9, two rules R1 and R2 are redundant if R1 target TR1 and rule R2 target TR2 intersect (see details in semantics Rule 3) and rule R1 rule condition
Table 14: Rules of Redundancy Detection Semantics

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>(&lt; TR_2, TR_1 &gt; \triangleright intersect True \land (\exists RC_2 \cap RC_1 \neq 0) \land (RE_1 = RE_2) \rightarrow &lt; R_1, R_2 &gt; \rightarrow RA(R_1, R_2))</td>
</tr>
<tr>
<td>10</td>
<td>(\forall R_1, R_2 \in SR; RS \leftarrow RS \cup (&lt; R_1, R_2 &gt; \rightarrow RA(R_1, R_2)))</td>
</tr>
<tr>
<td>11</td>
<td>((RCA.P_1 = RCA.P_2) \land (&lt; TR_1, TR_2 &gt; \triangleright intersect True) \land (\forall R_1 \in SR_1, R_2 \in SR_2; RS \leftarrow RS \cup (&lt; R_1, R_2 &gt; \rightarrow RA(R_1, R_2)))</td>
</tr>
<tr>
<td>12</td>
<td>(\forall P \in SP; RS \leftarrow RS \cup (&lt; P &gt; \rightarrow RA(RS)) \cup (\forall P_1, P_2 \in SP; RS \leftarrow RS \cup &lt; P_1, P_2 &gt; \rightarrow RA(RS)))</td>
</tr>
</tbody>
</table>

RC1 and rule R2 rule condition RC2 intersects and both rules R1 and R2 have the same rule effect RE1 equal RE2. In Rule 10, for every pair of rules R1 and R2 in policy P such that R1 and R2 are appended to the Redundant Set RS if R1 and R2 are redundant (see details in semantics rule 9). In Rule 11, given a pair of policies P1, P2 in policy set PS, P1 and P2 are appended to the Redundant Set RS if the rule combining of P1 is equal to the rule combining of P2 and the targets of P1 and P2 intersect (see details in semantics Rule 3) and there exists R1 in P1 and R2 in P2 such that R1 and R2 are redundant and appended to the Redundant Set RS (see details in semantics Rule 9). In Rule 12, given policy set PS, the Redundant Set RS is the union of all redundancies between any two redundant rules R1, R2 in one policy and between two redundant rules R1, R2 from two different policies and every two redundant policies P1 and P2. (see details in semantics rule 10,11).
4.2.4 Conflict detection

In this section, we present the policy conflict analysis semantics rules for SBA-XACML policy set.

Table 15: Rules of Conflict Detection Semantics

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
</table>
| Rule 13 | \[
\begin{align*}
< T_R1,T_R2 > & \vdash \text{intersect True} \land (\{\text{RC}_1 \land \text{RC}_2 \neq \emptyset\} \land (\text{RE}_1 \neq \text{RE}_2)) \\
\text{RC}_1 \cap \text{RC}_2 & \neq \emptyset \\
\text{RE}_1 & \neq \text{RE}_2 \\
< R_1,R_2 > & \rightarrow \text{Conflict}_{R_1,R_2} \\
(\forall R_1,R_2 \in \text{SR}; \text{RS} \leftarrow \text{CS} \cup (\langle R_1,R_2 > \rightarrow \text{Conflict}_{R_1,R_2})) \\
\end{align*}
\] |

| Rule 14 | \[
\begin{align*}
< P > & \rightarrow \text{CS} \quad \text{PCA} \\
(\forall P_1,P_2 \in \text{SP}; \text{CS} \leftarrow \text{CS} \cup (\langle P_1 > \rightarrow \text{CS}) \cup (\langle P_2 > \rightarrow \text{CS}) \cup (\langle P_1,P_2 > \rightarrow \text{CS})) \\
\end{align*}
\] |

| Rule 15 | \[
\begin{align*}
< P_1,P_2 > & \rightarrow \text{CS} \quad \text{PCA} \\
(\forall P_1,P_2 \in \text{SP}; \text{CS} \leftarrow \text{CS} \cup (\langle P_1 > \rightarrow \text{CS}) \cup (\langle P_2 > \rightarrow \text{CS}) \cup (\langle P_1,P_2 > \rightarrow \text{CS})) \\
\end{align*}
\] |

| Rule 16 | \[
\begin{align*}
< P_1,P_2 > & \rightarrow \text{CS} \quad \text{PCA} \\
(\forall P_1,P_2 \in \text{SP}; \text{CS} \leftarrow \text{CS} \cup (\langle P_1 > \rightarrow \text{CS}) \cup (\langle P_2 > \rightarrow \text{CS}) \cup (\langle P_1,P_2 > \rightarrow \text{CS})) \\
\end{align*}
\] |

Semantics Rules 13,14,15 and 16 in Table 15 describe the different conflict analysis cases for a policy set PS. In Rule 13, two rules R1 and R2 conflict if R1 target TR1 and rule R2 target TR2 intersect (see details in semantics Rule 3) and rule R1 rule condition RC1 and rule R2 rule condition RC2 intersects and rule R1 with effect RE1 which is the opposite of rule 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 (see details in semantics rule 13). 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 and the targets of P1 and P2 intersect (see details in semantics Rule 3) and there exists R1 in P1 and R2 in P2 such that R1 and R2 conflicted and appended to
the Conflict Set $CS$ (see details in semantics Rule 13). In Rule 16, given policy set $PS$, the Conflict Set $CS$ is the union of all conflicts between any two conflicting rules $R1$, $R2$ in one policy and between two conflicting rules $R1$, $R2$ from two different policies and every two conflicting policies $P1$ and $P2$. (see details in semantics rule 14,15).

4.3 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 4, (2) the Policy Analysis Algorithm in Algorithm 5 and (3) PolicySet Analysis Algorithm in Algorithm 6.

4.3.1 Rule Analysis Algorithm

In this subsection, We present the Rule Analysis Algorithm in Algorithm 4. It takes rules $R1$ and $R2$ as input. It checks for flaws, conflicts and redundancies between $R1$ and $R2$. The output is the Flaw, Conflict, Redundant or null.

The Rule Analysis in algorithm 4 takes two rules $R1$ and $R2$ as input and compares their targets, Rule conditions and rule effects to determine if there exists any flaws, conflicts and redundancies between the two rules. It returns the proper response to the Policy Analysis Algorithm in Algorithm 5. 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 $R1$ (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 control flaw and it should be removed. If the subject set of $R1$
Algorithm 4 Rule_Analysis($R_1$, $R_2$)

Input : Two Rules $R_1$ with Target $TR_1 = \{S_1, R_1, A_1\}$, rule condition $RC_1$, rule effect $RE_1$ and $R_2$ with Target $TR_2 = \{S_2, R_2, A_2\}$, rule condition $RC_2$, rule effect $RE_2$

Output : Rule analysis $\in \{\text{Flaw, Conflict, Redundant or Null}\}$

1: Flaw check if a rule $R_2$ is a subset of $R_1$
2: // Is $R_2$ target is a subset or equal to $R_1$ target
3: if $(S_2 \subseteq S_1) \land (R_2 \subseteq R_1) \land (A_2 \subseteq A_1)$ then
4: // Check rule conditions for $R_1$ and $R_2$
5: if $(RC_2 \subseteq RC_1)$ then
6: // Check if $R_1$ and $R_2$ have the same effect
7: if $(RE_1 = RE_2)$ then
8: // $R_2$ is a subset of $R_1$
9: return "Flaw";
10: end if
11: end if
12: end if
13: // Do the targets for $R_1$ and $R_2$ share common subjects, resources, actions
14: if $((S_1 \cap S_2) \neq \emptyset) \land ((R_1 \cap R_2) \neq \emptyset) \land ((A_1 \cap A_2) \neq \emptyset)$ then
15: // Check if rule conditions for $R_1$ and $R_2$ intersect
16: if $(RC_1 \cap RC_2) \neq \emptyset)$ then
17: // check if $R_1$ and $R_2$ have opposite effect
18: if $(RE_1 \neq RE_2)$ then
19: // $R_1$ and $R_2$ Conflict with each other
20: return "Conflict";
21: else
22: // $R_1$ and $R_2$ have same effect
23: return "Redundant";
24: end if
25: end if
26: end if
27: return ;

intersect with subject set of $R_2$, resource set of $R_1$ intersect with resource set of $R_2$ and
action set of $R_1$ intersect with action set of $R_2$ (line 13) and $R_1$ and $R_2$ have opposite effect (line 17) then $R_1$ conflicts with $R_2$ but if $R_1$ and $R_2$ have the same effect then $R_1$ and $R_2$ are redundant. Empty set is returned if no issues were found between the two rules (line 26).
4.3.2 Policy Analysis Algorithm

In this subsection, We present the policy analysis algorithm in Algorithm 5. It takes two policies \( P1 \) and \( P2 \) as input. The output is Flaw Set \( FS \) of all flaws at rule and policy level, Conflict Set \( CF \) and Redundancy Set \( RS \).

The policy analysis algorithm in Algorithm 5 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 (lines 2-18). It calls the rule analysis algorithm in Algorithm 4 on (line 5) to check every two rules for flaws, conflicts and redundancies. The returned response from the Rule Analysis Algorithm is appended to the proper set (lines 7-15). The second part of the algorithm checks for flaws, conflicts and redundancies between rules from different policies if the rule combining of both policies \( P1 \) and \( P2 \) have the same combining algorithms and the targets of both policies \( P1 \) and \( P2 \) intersect which means 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 20-43). It calls the rules analysis on (line 26) to check every two rules \( R1 \) from \( P1 \) and \( R2 \) from \( P2 \) for flaws, conflicts and redundancies. The returned response from the Rule Analysis is appended to the proper set.

4.3.3 PolicySet Analysis Algorithm

In this subsection, We present the PolicySet Analysis Algorithm in Algorithm 6. It takes a policy set \( PS \) as input. It calls the Policy Analysis Algorithm presented in Algorithm 5 to
Algorithm 5 Policy_Analysis($P_1, P_2$)

**Input**: Policy $P_1$ with Target $TR_1 = \{S_1, R_1, A_1\}$ and $P_2$ with Target $TR_2 = \{S_2, R_2, A_2\}$

**Output**: Flaw Set $FS$, Conflict Set $CS$ and Redundancy Set $RS$

1: //Check rules in each policy
2: for $l := 1$ to $P_1$ Number of Rules - 1 do
3: for $i := 1$ to $P_1$ Number of Rules do
4: RA = RULE_ANALYSIS($R_{li}, R_{lj}$);
5: // RA response
6: if (RA = "Flaw") then
7: $FS = FS \cup $ Flaw$R_{li}, R_{lj}$;
8: end if
9: if (RA = "Redundant") then
10: $RS = RS \cup $ Redundant$R_{li}, R_{lj}$;
11: end if
12: if (RA = "Conflict") then
13: $CS = CS \cup $ Conflict$R_{li}, R_{lj}$;
14: end if
15: end for
16: end for
17: end for
18: //Compare Rule Combining of $P_1$ and $P_2$
19: if ($RCA_{P_1} = RCA_{P_2}$) then
20: //Common subjects, Resources and Actions from $P_1$ and $P_2$
21: if (($S_1 \cap S_2) \neq \emptyset ) \land ((R_1 \cap R_2) \neq \emptyset ) \land ((A_1 \cap A_2) \neq \emptyset )$ then
22: // Check rules for Conflicts, Flaws and Redundancies
23: for $l := 1$ to $P_1$ Number of Rules do
24: for $m := 1$ to $P_2$ Number of Rules do
25: RA = RULE_ANALYSIS($R_{li}, R_{mm}$);
26: // RA response
27: if (RA = "Flaw") then
28: $FS = FS \cup $ Flaw$R_{li}, R_{mm}$;
29: $FS = FS \cup $ Flaw$P_1, P_2$;
30: end if
31: if (RA = "Conflict") then
32: $CS = CS \cup $ Conflict$R_{li}, R_{mm}$;
33: $CS = CS \cup $ Conflict$P_1, P_2$;
34: end if
35: if (RS = "Redundant") then
36: $RS = RS \cup $ Redundant$R_{li}, R_{mm}$;
37: $RS = RS \cup $ Redundant$P_1, P_2$;
38: end if
39: end for
40: end for
41: end if
42: end if
43: return ;
analyze the policies at the middle layer in the based policy. The output the analysis report which contains all Flaws at the policy and rule level $FS$, Conflicts $CS$ and Redundancies $RS$.

**Algorithm 6 PolicySet_Analysis($PS$)**

**Input**: A Policy Set PS  
**Output**: Analysis report

1: // Initialize sets for Flaws, Redundancies and Conflicts  
2: Global $FS = \emptyset$; $RS = \emptyset$; $CS = \emptyset$;  
3: //Loop through policies in PS  
4: for $i := 1$ to $PS_{NumberofPolices}$ 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 6 takes a policy set $PS$ as input and produces a report of all access flaws between policies and rules. It initializes global set $FS, CS$ and $RS$ on (line 2) for appending flaws, conflicts and redundancies found at both policy and rule levels. It calls the Policy Analysis Algorithm in Algorithm 5 on (line 7) for checking flaws, conflicts and redundancies between every two policies.

### 4.4 Case Study: SBA-XACML Policy Analysis

In this section, we present a case study illustrating the usability of SBA-XACML policy analysis process through semantics. We will be utilizing the XACML example presented in Chapter Two, Section 2.6.3, Listings 2.3 and 2.4. Listing 4.1 contains the generated SBA-XACML based policy corresponding the listings mentioned above.
Listing 4.1: 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, {ResourceAttributeDesignator, 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}>
```

Line 1 is the policy set PS. The policy set ID is PS1. It has two policies P1 and P2. P1 is ordered before P2. The policy combining algorithm is Permit – Overrides. PS1 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 P1. The policy ID is P1. It has two rules R1 and R2. R1 is ordered before R2. The rule combining algorithm is deny – Overrides. P1 has no obligations and no target. Line 3 is the rule R1. The rule ID is R1. R1 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. R1 has a permit effect. Line 4 is the rule R2. The rule ID is R2. R2 has no conditions. R2 has no target specified. R2 has a deny effect. Line 5 is the policy P2. The policy ID is P2. It has three rules R3, R4 and R5. The precedence order is R3, R4 and R5. The rule combining algorithm is permit – overrides. P2 has no obligation to perform and the target elements are not defined. Line 6 is the rule R3. The rule ID is R3. R3 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. R3 has a permit effect. Line 7 is the rule R4. The rule ID is R4. R4 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. R4 has a permit effect. Line 8 is the rule R5. The rule ID is R5. R5 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. R5 has a deny effect.

Based on the SBA-XACML policy analysis semantics in Section 4.2, the elaborated framework will analyze the based policy PSI presented in Listings 4.1 for access flaws. Since the analysis of each semantics rule is based on analyzing its premises, we will describe the analysis steps in order by the premises of policy sets, policies and rules.

### 4.4.1 Access Flaw Detection

In this section, we show the access flaw analysis steps and provide the results for policy set PSI based on the formal analysis semantics presented in 4.2.

PolicySet PSI flaw analysis starts here, hence Our options are limited to Rule 8

#### Step 1. (PSI-Premise1 Evaluation)

Rule 8 premise 1 requires the analysis of every policy individually to determine the results of premise 1. PSI has two policies P1 and P2. Since P1 takes a precedence order over P2 then the analysis starts with P1. The evaluation of PSI-Premise1 limits our options to Rule 6.

#### Step 1.1. (P1-Analysis)

P1 Analysis starts here.
Step 1.1.1. (P1-Premise1 Evaluation)

Rule 6 requires analysis of rules to determine the results. P1 has two rules R1 and R2.

The results of P1-premise1 depends on the evaluation of Rule 5.

Step 1.1.1.1 (R1,R2 - Analysis)

Rules R1 and R2 Analysis starts here.

Step 1.1.1.1.1 (R1,R2-Premise1 Evaluation)

Both R1 and R2 have no targets defined which means

TR1 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\} and TR2 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\}.

By applying subset semantics Rule(1):

\[(\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\})\]

\[\underbrace{<(TR2, TR1)>}_{\text{subset}} \vdash \text{True}\]

Targets TR1 and TR2 are the same therefore TR2 is subset or equal to TR1.

Step 1.1.1.1.2 (R1,R2-Premise2 Evaluation)

Rule R2 has no conditions defined, RC2 = \{} and rule R1 has 2 conditions, RC1 = \{\text{and,}\{\text{string-equal,RAD,string,BankService/withdraw}\},\{\text{string-equal,SAD,subject-id,string,Bob}\}\}, which means that the resource must be equal to BankService/withdraw and subject must be equal to Bob. RC2 is not a subset of RC1 because RC2 has no restrictions on any resources or subjects.

Premise 1 of R1,R2 is satisfied, while Premise 2 is not. At this stage, the evaluation of Premise 3 is not needed anymore since the semantics Rule (5) applies:

Based on the response from Rule (5), the returned response for R1 and R2 is null.
(\langle T R 2, T R 1 > \quad \vdash_{\text{subset}} T r u e \rangle \land (R C 2 \not\subseteq R C 1)

< R 1, R 2 > \rightarrow_{R . F A} \text{null}

Rules \text{R1} and \text{R2} Analysis ends here.

Since the analysis of Rules \text{R1} and \text{R2} returned null, therefore no flaws added to the Flaw Set \text{FS}. The results from Rule (6) \text{FS} = \{\}.

Policy \text{P1} Analysis ends here.

Flaw Set \text{FS} = \{\}.

Step 1.2. (P2-Analysis)

\text{P2} Analysis starts here.

Step 1.2.1. (P2-Premise1 Evaluation)

Rule 6 requires analysis of rules to determine the results. \text{P1} has three rules \text{R3}, \text{R4} and \text{R5}. The results of P2-premise1 depends on the evaluation of Rule 5.

Step 1.2.1.1 (R3,R4-Analysis)

Rules \text{R3} and \text{R4} Analysis starts here.

Step 1.2.1.1.1 (R3,R4-Premise1 Evaluation)

Both \text{R3} and \text{R4} have no targets defined which means TR3 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\} and TR4 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\}. By applying subset semantics Rule(1):

(\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\})

\langle (T R 4, T R 3) \quad \vdash_{\text{subset}} T r u e \rangle

Targets \text{TR3} and \text{TR4} are the same therefore \text{TR4} is subset or equal to \text{TR3}.
Step 1.2.1.1.2 (R3,R4-Premise2 Evaluation)

Rule $R3$ has one condition defined,

$RC3 = \{\text{string-equal}, \{\text{RAD}, \text{string}, \text{BankService/deposit}\}\}$, which means the resource must be equal to BankService/deposit and rule $R4$ has two conditions,

$RC4 = \{\text{and}, \{\text{string-equal}, \{\text{RAD}, \text{string}, \text{BankService/withdraw}\}\}, \{\text{string-equal}, \{\text{SAD}, \text{subject-id}, \text{string}, \text{Joe}\}\}\}$, which means that the resource must be equal to BankService/withdraw and subject must be equal to Joe. $RC4$ is a subset of $RC3$ because both require the resource to be BankService/withdraw and $RC4$ limits the subject while $RC3$ accepts any subject.

Step 1.2.1.1.3 (R3,R4-Premise3 Evaluation)

Both $R3$ and $R4$ have the same effect. All premises of Rule (5) are satisfied. By applying Rule (5):

$$
(< TR4, TR3 > \models True) \land (RC4 \subseteq RC3) \land (RE3 = RE4) \\
\therefore < R3, R4 > \rightarrow_{R,FA} Flaw_{R3,R4}
$$

Based on the response from Rule (5), the returned response for $R3$ and $R4$ is $Flaw_{R3,R4}$.

Rules $R3$ and $R4$ Analysis ends here.

Flaw Set $FS = \{Flaw_{R3,R4}\}$.

Step 1.2.1.2 (R3,R5-Analysis)

Rules $R3$ and $R5$ Analysis starts here.
Step 1.2.1.2.1 (R3,R5-Premise1 Evaluation)

Both $R3$ and $R5$ have no targets defined which means

$TR3 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\}$ and $TR5 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\}$.

By applying subset semantics Rule(1):

\[
\begin{align*}
(\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\}) \\
\hline
\end{align*}
\]

\[
< (TR5, TR3) > \quad \downarrow_{\text{subset}} \quad True
\]

Targets $TR3$ and $TR5$ are the same therefore $TR5$ is subset or equal to $TR3$.

Step 1.2.1.2.2 (R3,R5-Premise2 Evaluation)

Rule $R3$ has one conditions defined,

$RC3 = \text{string-equal,RAD,string,BankService/deposit}$, which means the resource must be equal to BankService/deposit and rule $R5$ has 2 conditions,

$RC5 = \{\text{and},\{\text{string-equal},\{\text{RAD},\text{string},\text{BankService/withdraw}\}\},$

$\{\text{string-equal},\{\text{SAD,subject-id,string},\text{Joe}\}\}\}$, which means that the resource must be equal to BankService/withdraw and subject must be equal to Joe. $RC5$ is a subset of $RC3$ because both require the resource to be BankService/withdraw and RC5 limits the subject while RC3 accepts any subject.

Step 1.2.1.2.3 (R3,R5-Premise3 Evaluation)

Rule $R3$ has the opposite effect of rule $R5$. $R3,R5$ Premises 1 and 2 are satisfied, while Premise 3 is not. Applying semantics Rule (5):

Based on the response from Rule (5), the returned response for $R3$ and $R5$ is null.
\((< TR5, TR3 > \vdash True) \land (RC5 \subseteq RC3) \land (RE3 \neq RE5)\)

\(< R3, R5 > \xrightarrow{R.FA} null\)

Rules R3 and R5 Analysis ends here.

**Step 1.2.1.3 (R4,R5-Analysis)**

Rules R4 and R5 Analysis starts here.

**Step 1.2.1.3.1 (R4,R5-Premise1 Evaluation)**

Both R4 and R5 have no targets defined which means

\(TR4 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\}\) and \(TR5 = TR3 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\}\).

By applying subset semantics Rule(1):

\((\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\}) \land (\{\text{Any}\} \subseteq \{\text{Any}\})\)

\(< (TR5, TR4) > \vdash True\)

Targets \(TR4\) and \(TR5\) are the same therefore \(TR5\) is subset or equal to \(TR4\).

**Step 1.2.1.3.2 (R4,R5-Premise2 Evaluation)**

Rule R4 has two conditions defined,

\(RC4 = \{\text{and},\{\text{string-equal},\{RAD,\text{string},\text{BankService/withdraw}\}\}\},\)

\(\{\text{string-equal},\{SAD,\text{subject-id},\text{string},\text{Joe}\}\}\}\), which means the resource must be equal to BankService/deposit and subject equal to Joe and rule R5 has 2 conditions,

\(RC5 = \{\text{and},\{\text{string-equal},\{RAD,\text{string},\text{BankService/withdraw}\}\}\},\)

\(\{\text{string-equal},\{SAD,\text{subject-id},\text{string},\text{Joe}\}\}\}\), which means that the resource must be equal to BankService/withdraw and subject must be equal to Joe. \(RC5\) is equal
to $RC4$, therefore $RC5$ is a subset or equal to $RC4$.

**Step 1.2.1.3.3 (R4,R5-Premise3 Evaluation)**

Rule $R4$ has the opposite effect of rule $R5$. $R4,R5$ Premises 1 and 2 are satisfied, while Premise 3 is not. Applying semantics Rule (5):

\[
\langle TR5, TR4 \rangle \vdash_{\text{subset}} \text{True} \land (RC5 \subseteq RC4) \land (RE4 \neq RE5)
\]

\[
< R4, R5 > \rightarrow_{R.FA} \text{null}
\]

Based on the response from Rule (5), the returned response for $R4$ and $R5$ is null.

Rules $R4$ and $R5$ Analysis ends here.

Policy $P2$ Analysis ends here.

Flaw Set $FS = \{Flaw_{R3,R4}\}$.

**Step 2. (PS1-Premise2 Evaluation)**

Rule 8 requires the analysis of every pair of policies to determine the results of premise 2. $PS1$ has two policies $P1$ and $P2$, therefore the evaluation can continue. The evaluation of PS1-Premise2 limits our options to Rule 7.

**Step 2.1. (P1,P2-Analysis)**

$P1,P2$ Analysis starts here.

**Step 2.1.1. (P1,P2-Premise1 Evaluation)**

Rule 7 premise 1 requires both $P1$ and $P2$ to have the same combining algorithms. $P1$ has $RCA = \{\text{deny-overrides}\}$ and $P2$ has $RCA = \{\text{permit-overrides}\}$ therefore premise
1 of Rule (7) is not satisfied. Based on the response from Rule (7), the returned response is null.

\textit{P1,P2 Analysis ends here.}

\text{Flaw Set FS} = \{\text{Flaw}_{R3,R4}\}.

\text{PS1 analysis ends here.}

\text{Flaw Set FS} = \{\text{Flaw}_{R3,R4}\}.

The result from the semantics analysis for \textit{PS1} show that \textit{R3} and \textit{R4} cause access flaw.

\section*{4.4.2 Redundancy Detection}

In this section, we show the redundancy analysis steps and provide the results for policy set \textit{PS1} based on the formal analysis semantics presented in 4.2.

\textit{PolicySet PS1 redundancy analysis starts here hence our options are limited to Rule 12}

\textbf{Step 1. (PS1-Premise1 Evaluation)}

Rule \textbf{12} requires the analysis of every policy individually to determine the results of premise 1. \textit{PS1} has two policies \textit{P1} and \textit{P2}. Since \textit{P1} takes a precedence order over \textit{P2} then the analysis starts with \textit{P1}. The evaluation of PS1-Premise1 limits our options to Rule \textbf{10}.

\textbf{Step 1.1. (P1-Analysis)}

\textit{P1 Analysis starts here.}
Step 1.1.1. (P1-Premise1 Evaluation)

Rule 10 requires analysis of rules to determine the results. P1 has two rules R1 and R2. The results of P1-premise1 depends on the evaluation of Rule 9.

Step 1.1.1.1 (R1,R2-Analysis)

Rules R1 and R2 Analysis starts here.

Step 1.1.1.1.1 (R1,R2-Premise1 Evaluation)

Both R1 and R2 have no targets defined which means

\[ TR1 = \{ \{ \text{Any} \}, \{ \text{Any} \}, \{ \text{Any} \} \} \] and \[ TR2 = \{ \{ \text{Any} \}, \{ \text{Any} \}, \{ \text{Any} \} \} \].

By applying Intersection semantics Rule(3):

\[
\left( (\{ \text{Any} \} \cap \{ \text{Any} \}) \neq \emptyset \right) \land \left( (\{ \text{Any} \} \cap \{ \text{Any} \}) \neq \emptyset \right) \land \left( (\{ \text{Any} \} \cap \{ \text{Any} \}) \neq \emptyset \right)
\]

\[ \vdash \text{intersect True} \]

Targets TR1 and TR2 are the same therefore TR1 intersect with TR2.

Step 1.1.1.1.2 (R1,R2-Premise2 Evaluation)

Rule R1 has 2 conditions,

\[ RC1 = \{ \text{and,} \{ \text{string-equal,} \{ \text{RAD,string,BankService/withdraw} \} \}, \{ \text{string-equal,} \{ \text{SAD,subject-id,string,Bob} \} \} \} \]

which means that the resource must be equal to BankService/withdraw and subject must be equal to Bob and rule R2 with RC2 has no conditions defined, RC2 = {}. RC1 intersect RC2 because RC2 has no restrictions on any resources or subjects.

Step 1.1.1.1.3 (R1,R2-Premise3 Evaluation)

Rule R1 has a permit effect and rule R2 has a deny effect, therefore premise 3 is not satisfied. Applying semantics Rule (9):

93
\[
(\langle TR1, TR2 \rangle \models \text{True}) \land ((RC2 \cap RC1) \neq \emptyset) \land (RE1 \neq RE2) \]

\[
\langle R1, R2 \rangle \overset{\text{null}}{\rightarrow}_{\text{R,RA}}
\]

Based on the response from Rule (9), the returned response for \( R1 \) and \( R2 \) is null.

Rules \( R1 \) and \( R2 \) Analysis ends here.

Policy \( P1 \) Analysis ends here.

Redundant Set \( RS = \emptyset \).

**Step 1.2. (P2-Analysis)**

\( P2 \) Analysis starts here.

**Step 1.2.1. (P2-Premise1 Evaluation)**

Rule 10 requires analysis of rules to determine the results. \( P1 \) has three rules \( R3, R4 \) and \( R5 \). The results of P2-premise1 depends on the evaluation of Rule 9.

**Step 1.2.1.1 (R3,R4-Analysis)**

Rules \( R3 \) and \( R4 \) Analysis starts here.

**Step 1.2.1.1.1 (R3,R4-Premise1 Evaluation)**

Both \( R3 \) and \( R4 \) have no targets defined which means

\( TR3 = \{ \{ \text{Any} \}, \{ \text{Any} \}, \{ \text{Any} \} \} \) and \( TR4 = \{ \{ \text{Any} \}, \{ \text{Any} \}, \{ \text{Any} \} \} \).

By applying Intersection semantics Rule(3):

Targets \( TR3 \) and \( TR4 \) are the same therefore \( TR3 \) intersect with \( TR4 \).
\[((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \text{Any}) \neq \emptyset)\) < (\text{TR3, TR4}) \Rightarrow \vdash \text{intersect True}

**Step 1.2.1.1.2 (R3,R4-Premise2 Evaluation)**

Rule R3 has one conditions defined,

RC3 = \{\text{string-equal,\{RAD,string,BankService/deposit\}}\}, which means

the resource must be equal to BankService/deposit and rule R4 has two conditions,

RC4 = \{\text{and,\{string-equal,\{RAD,string,BankService/withdraw\}\}}, \{\text{string-equal,\{SAD,subject-id,string,Joe\}}\}\}, which means that the resource must

be equal to BankService/withdraw and subject must be equal to Joe. RC3 inter-

sects with RC4 because both require the resource to be BankService/withdraw and

RC4 limits the subject while RC3 accepts any subject.

**Step 1.2.1.1.3 (R3,R4-Premise3 Evaluation)**

Both rules R3 and R4 have the same effect. All premises of Rule (9) are satisfied.

Applying Rule (9):

\(< \text{TR3, TR4} > \vdash \text{intersect True} \land (\text{RC4} \cap \text{RC3}) \neq \emptyset \land (\text{RE3} = \text{RE4})\) < R3, R4 > \xrightarrow{R.RA} \text{Redundant}_{R3,R4}

Based on the response from Rule (9), the returned response for R3 and R4 is

\text{Redundant}_{R3,R4}.

Rules R3 and R4 Analysis ends here.
Redundant Set \( RS = \{Redundant_{R3,R4}\} \).

**Step 1.2.1.2** (R3,R5-Analysis)

Rules \( R3 \) and \( R5 \) Analysis starts here.

**Step 1.2.1.2.1** (R3,R5-Premise1 Evaluation)

Both \( R3 \) and \( R5 \) have no targets defined which means

\[
TR3 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\} \text{ and } TR5 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\}.
\]

By applying the Intersection semantics Rule(3):

\[
< (TR3, TR5) > \vdash intersect True
\]

Targets \( TR3 \) and \( TR5 \) are the same therefore \( TR3 \) intersect with \( TR5 \).

**Step 1.2.1.2.2** (R3,R5-Premise2 Evaluation)

Rule \( R3 \) has one conditions defined,

\( RC3 = \{\text{string-equal},\{\text{RAD},\text{string},\text{BankService/deposit}\}\} \), which means the resource must be equal to BankService/deposit and rule \( R5 \) has 2 conditions,

\( RC5 = \{\text{and},\{\text{string-equal},\{\text{RAD},\text{string},\text{BankService/withdraw}\}\}, \{\text{string-equal},\{\text{SAD},\text{subject-id},\text{string},\text{Joe}\}\}\} \), which means that the resource must be equal to BankService/withdraw and subject must be equal to Joe. \( RC3 \) intersect with \( RC5 \) because both require the resource to be BankService/withdraw and \( RC5 \) limits the subject while \( RC3 \) accepts any subject.

**Step 1.2.1.2.3** (R3,R5-Premise3 Evaluation)

Rule \( R3 \) has the opposite effect of rule \( R5 \). \( R3,R5 \) Premises 1 and 2 are satisfied,
while Premise 3 is not. Applying semantics Rule (9):

\[
(< TR3, TR5 > ⊬ True) ∧ ((RC3 \cap RC5) \neq \emptyset) ∧ (RE3 \neq RE5)
\]

\[
< R3, R5 > \rightarrow_{null}
\]

Based on the response from Rule (9), the returned response for \( R3 \) and \( R5 \) is null.

Rules \( R3 \) and \( R5 \) Analysis ends here.

Redundant Set \( RS = \{\text{Redundant}_{R3, R4}\} \).

**Step 1.2.1.3 (R4,R5-Analysis)**

Rules \( R4 \) and \( R5 \) Analysis starts here.

**Step 1.2.1.3.1 (R4,R5-Premise1 Evaluation)**

Both \( R4 \) and \( R5 \) have no targets defined which means

\( TR4 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\} \) and \( TR5 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\} \).

By applying the Intersection semantics Rule(3):

\[
((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) ∧ ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) ∧ ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset)
\]

\[
< (TR4, TR5) > \rightarrow True
\]

Targets \( TR4 \) and \( TR5 \) are the same therefore \( TR5 \) intersects with \( TR4 \).

**Step 1.2.1.3.2 (R4,R5-Premise2 Evaluation)**

Rule \( R4 \) has two conditions defined,
RC4 = \{\text{and,\{string-equal,\{RAD,string,BankService/withdraw\}\}},
\{\text{string-equal,\{SAD,subject-id,string,Joe\}\}}\}, which means the
resource must be equal to BankService/deposit and subject equal to Joe and rule
R5 has 2 conditions, RC5 = \{\text{and,\{string-equal,\{RAD,string,BankService/withdraw\}\}},
\{\text{string-equal,\{SAD,subject-id,string,Joe\}\}}\}, which means that the resource must
be equal to BankService/withdraw and subject must be equal to Joe. RC5 is equal
to RC4, therefore RC5 intersects with RC4.

**Step 1.2.1.3.3 (R4,R5-Premise3 Evaluation)**

Rule R4 has the opposite effect of rule R5. R4,R5 Premises 1 and 2 are satisfied,
while Premise 3 is not. Applying semantics Rule (9):

\[
\left( < TR_4, TR_5 > \vdash \text{True} \right) \land \left( (RC_4 \cap RC_5) \neq \emptyset \right) \land \left( RE_4 \neq RE_5 \right)
\]

\[
\frac{\quad < R_4, R_5 > \longrightarrow \text{null}}{R,R_A}
\]

Based on the response from Rule (9), the returned response for R4 and R5 is null.

Rules R4 and R5 Analysis ends here.

Policy P2 Analysis ends here.

Redundant Set RS = \{Redundant_{R3,R4}\}.

**Step 2. (PS1-Premise2 Evaluation)**

Rule 12 requires the analysis of every pair of policies to determine the results of premise
2. PS1 has two policies P1 and P2, therefore the evaluation can continue. The evaluation
of PS1-Premise2 limits our options to Rule 11.
Step 2.1. (P1,P2-Analysis)

P1,P2 Analysis starts here.

Step 2.1.1. (P1,P2-Premise1 Evaluation)

Rule 11 requires both P1 and P2 to have the same combining algorithms. P1 has RCA = {deny-overrides} and P2 has RCA = {permit-overrides} therefore premise 1 of Rule (11) is not satisfied. Based on the response from Rule (11), the returned response is null.

P1,P2 Analysis ends here.

Redundant Set RS = \{Redundant_{R3,R4}\}.

PS1 analysis ends here.

The results from the semantics analysis for PS1 show that R3 and R4 are redundant.

Redundant Set RS = \{Redundant_{R3,R4}\}.

4.4.3 Conflict Detection

In this section, we show the conflict analysis steps and provide the results for policy set PS1 based on the formal analysis semantics presented in 3.4.

PolicySet PS1 conflict analysis starts here hence our options are limited to Rule 16

Step 1. (PS1-Premise1 Evaluation)

Rule 16 requires the analysis of every policy individually to determine the results of premise 1. PS1 has two policies P1 and P2. Since P1 takes a precedence order over P2
then the analysis starts with $P1$. The evaluation of PS1-Premise1 limits our options to Rule 14.

**Step 1.1. (P1-Analysis)**

$P1$ Analysis starts here.

**Step 1.1.1. (P1-Premise1 Evaluation)**

Rule 14 requires analysis of rules to determine the results. $P1$ has two rules $R1$ and $R2$. The results of $P1$-premise1 depends on the evaluation of Rule 13.

**Step 1.1.1.1 (R1,R2-Analysis)**

Rules $R1$ and $R2$ Analysis starts here.

**Step 1.1.1.1.1 (R1,R2-Premise1 Evaluation)**

Both $R1$ and $R2$ have no targets defined which means

$\text{TR1} = \{\text{Any},\text{Any},\text{Any}\}$ and $\text{TR2} = \{\text{Any},\text{Any},\text{Any}\}$.

By applying Intersection semantics Rule(3):

$$((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \text{Any}) \neq \emptyset)$$

$$\prec (\text{TR1},\text{TR2}) \succ \vdash_{\text{intersect}} \text{True}$$

Targets $TR1$ and $TR2$ are the same therefore $TR1$ intersect with $TR2$.

**Step 1.1.1.1.2 (R1,R2-Premise2 Evaluation)**

Rule $R1$ has 2 conditions defined,

$\text{RC1} = \{\text{and},\{\text{string-equal},\{\text{RAD},\text{string},\text{BankService/withdraw}\}\},$

$\{\text{string-equal},\{\text{SAD},\text{subject-id},\text{string},\text{Bob}\}\}\}$, which means that the resource must be equal to BankService/withdraw and subject must be equal to Bob and rule $R2$
with $RC2$ has no conditions defined, $RC2 = \{\}$. $RC1$ intersect $RC2$ because $RC2$ has no restrictions on any resources or subjects.

**Step 1.1.1.3 (R1,R2-Premise3 Evaluation)**

Rule $R1$ has a permit effect and rule $R2$ has a deny effect, therefore premise 3 is satisfied. Applying semantics Rule (13):

$$
(< TR1, TR2 > \vdash_{intersect} True) \land ((RC2 \cap RC1) \neq \emptyset) \land (RE1 \neq RE2)
$$

$$
< R1, R2 > \xrightarrow{R,CA} Conflict_{R1,R2}
$$

Based on the response from Rule (13), the returned response for $R1$ and $R2$ is $Conflict_{R1,R2}$.

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

Since the analysis of Rules $R1$ and $R2$ returned $Conflict_{R1,R2}$, therefore it is added to the Conflict Set.

Policy $P1$ Analysis ends here.

Conflict Set $CS = \{Conflict_{R1,R2}\}$.

**Step 1.2. (P2-Analysis)**

$P2$ Analysis starts here.

**Step 1.2.1. (P2-Premise1 Evaluation)**

Rule 14 requires analysis of rules to determine the results of premise 1. $P1$ has three
rules $R3$, $R4$ and $R5$. The results of P2-premise1 depends on the evaluation of Rule 13.

**Step 1.2.1.1 (R3,R4-Analysis)**

Rules $R3$ and $R4$ Analysis starts here.

**Step 1.2.1.1.1 (R3,R4-Premise1 Evaluation)**

Both $R3$ and $R4$ have no targets defined which means

$TR3 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\} \text{ and } TR4 = \{\{\text{Any}\},\{\text{Any}\},\{\text{Any}\}\}.$

By applying the Intersection semantics Rule(3):

\[
((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \text{Any}) \neq \emptyset) < (TR3, TR4) \vdash intersect True
\]

Targets $TR3$ and $TR4$ are the same, therefore $TR3$ intersect with $TR4$.

**Step 1.2.1.1.2 (R3,R4-Premise2 Evaluation)**

Rule $R3$ has one conditions defined,

$RC3 = \{\text{string-equal},\{\text{RAD},\text{string},\text{BankService/deposit}\}\},$ which means

the resource must be equal to BankService/deposit and rule $R4$ has two conditions,

$RC4 = \{\text{and},\{\text{string-equal},\{\text{RAD},\text{string},\text{BankService/withdraw}\}\},$

$\{\text{string-equal},\{\text{SAD},\text{subject-id},\text{string},\text{Joe}\}\}\},$ which means that the resource must

be equal to BankService/withdraw and subject must be equal to Joe. $RC3$ intersects with $RC4$ because both require the resource to be BankService/withdraw and $RC4$ limits the subject while $RC3$ accepts any subject.

**Step 1.2.1.1.3 (R3,R4-Premise3 Evaluation)**

102
Both $R3$ and $R4$ have the same effect. Premises 1 and 2 are satisfied but premise 3 is not. Applying Rule (13):

\[
< TR3, TR4 > \vdash True \land ((RC3 \cap RC4) \neq \emptyset) \land (RE3 = RE4)
\]

\[
< R3, R4 > \xrightarrow{R,CA} null
\]

Based on the response from Rule (9), the returned response for $R3$ and $R4$ is null.

Rules $R3$ and $R4$ Analysis ends here.

Conflict Set $CS = \{Conflict_{R1, R2}\}$.

**Step 1.2.1.2 (R3,R5-Analysis)**

Rules $R3$ and $R5$ Analysis starts here.

**Step 1.2.1.2.1 (R3,R5-Premise1 Evaluation)**

Both $R3$ and $R5$ have no targets defined which means

$TR3 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\}$ and $TR5 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\}$.

By applying the Intersection semantics Rule(3):

\[
((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset)
\]

\[
< (TR3, TR5) > \vdash True
\]

Targets $TR3$ and $TR5$ are the same therefore $TR3$ intersect with $TR5$.

**Step 1.2.1.2.2 (R3,R5-Premise2 Evaluation)**

Rule $R3$ has one conditions defined,
RC3 = \{\text{string-equal,\{RAD,string,BankService/deposit\}}\}, which means the resource must be equal to BankService/deposit and rule R5 has 2 conditions, RC5 = \{\text{and,\{string-equal,\{RAD,string,BankService/withdraw\}\}}, \{\text{string-equal,\{SAD,subject-id,string,Joe\}}\}\}, which means that the resource must be equal to BankService/withdraw and subject must be equal to Joe. RC3 intersect with RC5 because both require the resource to be BankService/withdraw and RC5 limits the subject while RC3 accepts any subject.

**Step 1.2.1.2.3 (R3,R5-Premise3 Evaluation)**

Rule R3 has the opposite effect of rule R5. R3,R5 Premises 1, 2 and 3 are satisfied. Applying semantics Rule (13):

\[
\begin{align*}
\langle TR_3, TR_5 \rangle \vdash \text{True} & \land (\text{RC3} \cap \text{RC5}) \neq \emptyset \land (\text{RE3} \neq \text{RE5}) \\
\hline
\langle R_3, R_5 \rangle & \rightarrow_{R,CA} \text{Conflict}_{R3,R5}
\end{align*}
\]

Based on the response from Rule (13), the returned response for R3 and R5 is Conflict_{R3,R5}.

Rules R3 and R5 Analysis ends here.

Conflict Set CS = \{Conflict_{R1,R2}, Conflict_{R3,R5}\}.

**Step 1.2.1.3 (R4,R5-Analysis)**

Rules R4 and R5 Analysis starts here.

**Step 1.2.1.3.1 (R4,R5-Premise1 Evaluation)**

Both R4 and R5 have no targets defined which means
\( TR4 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\} \) and \( TR5 = \{\{\text{Any}\}, \{\text{Any}\}, \{\text{Any}\}\} \).

By applying the Intersection semantics Rule(3):

\[
((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset) \land ((\{\text{Any}\} \cap \{\text{Any}\}) \neq \emptyset)
\]

\[
\langle TR4, TR5 \rangle \triangleright \text{intersect True}
\]

Targets \( TR4 \) and \( TR5 \) are the same therefore \( TR4 \) intersects with \( TR5 \).

**Step 1.2.1.3.2 (R4,R5-Premise2 Evaluation)**

Rule \( R4 \) has two conditions defined,

\( RC4 = \{\text{and}, \{\text{string-equal}, \{\text{RAD}, \text{string}, \text{BankService/withdraw}\}\}, \{\text{string-equal}, \{\text{SAD}, \text{subject-id}, \text{string}, \text{Joe}\}\}\} \),

which means

the resource must be equal to BankService/deposit and subject equal to Joe

and rule \( R5 \) has 2 conditions,

\( RC5 = \{\text{and}, \{\text{string-equal}, \{\text{RAD}, \text{string}, \text{BankService/withdraw}\}\}, \{\text{string-equal}, \{\text{SAD}, \text{subject-id}, \text{string}, \text{Joe}\}\}\} \),

which means that

the resource must be equal to BankService/withdraw and subject must be equal to Joe. \( RC5 \) is equal to \( RC4 \), therefore \( RC4 \) intersects with \( RC5 \).

**Step 1.2.1.3.3 (R4,R5-Premise3 Evaluation)**

Rule \( R4 \) has the opposite effect of rule \( R5 \). R4,R5 Premises 1, 2 and 3 are satisfied.

Applying semantics Rule (13):

Based on the response from Rule (13), the returned response for \( R4 \) and \( R5 \) is \( Conflict_{R4,R5} \).
\[(< TR4, TR5 > \implies \text{True}) \land ((RC4 \cap RC5) \neq \emptyset) \land (RE4 \neq RE5)\]

\[< R4, R5 > \rightarrow Conflict_{R4, R5}\]

Rules \(R4\) and \(R5\) Analysis ends here.

Policy \(P2\) Analysis ends here.

Conflict Set \(CS = \{Conflict_{R1, R2}, Conflict_{R3, R5}, Conflict\ R4, R5\}\).

**Step 2. (PS1-Premise2 Evaluation)**

Rule \(16\) requires the analysis of every pair of policies to determine the results of premise 2. \(PS1\) has two policies \(P1\) and \(P2\), therefore the evaluation can continue. The evaluation of \(PS1\)-Premise2 limits our options to Rule \(15\).

**Step 2.1. (P1,P2-Analysis)**

\(P1,P2\) Analysis starts here.

**Step 2.1.1. (P1,P2-Premise1 Evaluation)**

Rule \(15\) premise 1 requires both \(P1\) and \(P2\) to have the same combining algorithms. \(P1\) has \(RCA = \{\text{deny-overrides}\}\) and \(P2\) has \(RCA = \{\text{permit-overrides}\}\) therefore premise 1 of Rule \(15\) is not satisfied. Based on the response from Rule \(15\), the returned response is null.

\(P1,P2\) Analysis ends here.

Conflict Set \(CS = \{Conflict_{R1, R2}, Conflict_{R3, R5}, Conflict\ R4, R5\}\).

PS1 analysis ends here.
Conflict Set \( CS = \{Conflict_{R1,R2}, Conflict_{R3,R5}, Conflict_{R4,R5}\} \).
The results from the semantics analysis for \( PS1 \) show that \( R1 \) conflicts with \( R2 \), \( R3 \) conflicts with \( R5 \) and \( R4 \) conflicts with \( R5 \).

4.5 Conclusion

In this chapter, we addressed the problems related XACML policy correctness (i.e. flaw and conflict free). In the context, we proposed a model which supports the analysis of both XACML and SBA-XACML policies. It is an automatic analysis approach for detecting access flaws, conflicts and redundancies between rules and policies. In addition, we provided the semantics and its corresponding algorithms for policy analysis. We realized and demonstrate the viability of our proposition by implementing the model and developing a case study to express the effectiveness of our proposition.
Chapter Five

Conclusion

This thesis addressed the problems related to the efficiency of real-time decision process and correctness of XACML policies. In this context, we elaborated a novel set-based scheme called SBA-XACML, which provides efficient evaluation and analysis of XACML policies. The SBA-XACML representation of policies maintains the same XACML structure and accounts for all its elements and their sub elements including rule conditions, obligations, policy request and policy response. Moreover, the policy evaluation module, which embeds formal semantics and its implemented algorithms, takes advantage of the mathematical operations to provide efficient decision process. Unlike current literature, it holds the same architecture of the industrial standard XACML Sun PDP ([18]) and respects the major properties and assumptions of real-life environments in terms of remote policy loading upon need and disjoint reception of requests from distributed parties. The corresponding experimental results explore that SBA-XACML evaluation of large and small sizes policies has better performance than Sun PDP ([18]) and its corresponding ameliorations [12, 17, 19, 24]. Finally, the policy analysis module, which also embeds formal
semantics and its implemented algorithms, allows to detect access flaws, conflict and redundancy at policy and rule levels. In the sequel, we present a brief summary of the thesis contributions:

- Set-Based intermediate representation of XACML constructs into readable mathematical syntax that maintain the same XACML policy structure and account for all its elements and their sub elements. The corresponding language and compiler offer automatic and optional conversion from XACML to SBA-XACML constructs.

- Formal semantics and its implemented algorithms that take advantage of the mathematical operations to provide efficient policy evaluation. The presented experimental results explore that SBA-XACML evaluation of large and small sizes policies has better performance than Sun PDP [18] and its corresponding ameliorations [12, 17, 19, 24].

- Formal semantics and its implemented algorithms for SBA-XACML policy analysis that enable to detect access flaws, conflicts and redundancies at both policy and rule levels.

**Future Work**

Our future work is to install our framework into a real world environment and monitor its performance on mixed size policies small and large and to broaden our policy analysis to detect more access control flaws, conflicts and redundancies.
List of Publications

The following is the list of publications derived from the thesis work:

Conference Paper


Draft Paper

"Towards a Set-Based Approach for Efficient Evaluation and Analysis of XACML Policies".
Bibliography


