CERIAS Tech Report 2010-35

EXAM - An Environment for XACML Policy Analysis and Management

by Prathima Rao
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Prathima Rao

Entitled
EXAM - An Environment for XACML Policy Analysis and Management

For the degree of Doctor of Philisophy

Is approved by the final examining committee:

Elisa Bertino
Chair

Ninghui Li

Sunil Prabhakar

Luo Si

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Elisa Bertino

Approved by: William J. Gorman 06/21/2010
Head of the Graduate Program Date

Graduate School Form 20
(Revised 1/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:
EXAM - An Environment for XACML Policy Analysis and Management

For the degree of Doctor of Philosophy

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Teaching, Research, and Outreach Policy on Research Misconduct (VIII.3.1)*, October 1, 2008.*

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Prathima Rao
Printed Name and Signature of Candidate

06/22/2010
Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/viii_3_1.html

EXAM: AN ENVIRONMENT FOR XACML POLICY ANALYSIS AND MANAGEMENT

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Prathima R. Rao

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2010

Purdue University

West Lafayette, Indiana

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

To my dearest parents and husband for their unconditional love and support.

ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to my thesis advisor Professor Elisa Bertino for her guidance and support throughout the course of this thesis. Under her mentorship I have had the opportunity to meet and collaborate with several accomplished researchers in the field of information security. I have learnt a great deal about relevant research from my association with her. Her accomplishments and dedication to research has been and will be a source of inspiration.

I would like to thank Dr. Dan Lin, currently Assistant Professor at Missouri State University, for her involvement in this thesis while she was working as a post doctoral researcher at Purdue. I have enjoyed and learnt a lot from our long discussions and interactions during the course of co-authoring several research papers together. She is a dear friend and colleague.

I would like to express my gratitude to Professor Ninghui Li from Purdue and Dr. Jorge Lobo from IBM TJ Watson Research Center. The thesis has greatly benefitted from their contributions and insightful comments .

I would like to thank Dr. Gabriel Ghinita who as a post doctoral researcher at Purdue contributed to the development of the visualization component of EXAM and offered several useful critiques to improve EXAM. I would like to thank Professor Fariborz Farahmand for his collaboration on the policy similarity usability survey.

I would like to thank committee members Professor Sunil Prabhakar and Professor Luo Si for their comments and feedback.

I would like to express my immense gratitude to Professor. Bernie Engel, Head of the Department of Agricultural and Biological Engineering (ABE), for providing me with financial assistantship to support most part of my PhD program. As a research assistant at ABE, I enjoyed working with the post doctoral researchers Dr. Dibyajyothi Tripathi and Dr. James Hunter. I would also like to thank Professor Suresh Jagannathan, Professor Tony

Hosking and Professor Jan Vitek of Secure Systems Software Lab who provided me with the opportunity to start my PhD program at Purdue Computer Science and Professor Venkat Venkatasubramanian of Chemical Engineering Department for providing me with financial assistantship for the first year.

I am deeply indebted to my parents for the quality of life and education they have provided me. I am here today because of their love, hard work, sacrifice and encouragement. They are role models for my life. I am very grateful to my dearest husband Dr. Deepak Bobbarjung for his unconditional love and support. His patience and positive attitude have helped me through difficult times. Meeting him is the best thing to have happened to me at Purdue. I would also like to thank my sister, brother-in-law and parents-in-laws for being there for me always.

I would like to thank Ashish Kundu, Ashish Kamra, Dr. Abhilasha Bhargav-Spantzel, Dr. Balachandra Bhadravati Krishnamurthy and Dr. Mercan Topkara with whom I have shared useful discussions . I would like to thank my friends Dr. Muralikrishna Ramanathan, Dr. Mehmet Koyuturk , Dr. Gunnur Karakurt , Dr. Sagar Pandit , Dr. Gauri Pradhan and Mummoorthy Murugesan for lots of fun and memorable times at Purdue. I would like to thank Professor Prabhu Nagabhushana and family for their affection and good food during my stay at Purdue.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Rao, Prathima Rama. Ph.D., Purdue University, August 2010. EXAM: An Environment for XACML Policy Analysis and Management. Major Professor: Elisa Bertino.

As distributed collaborative applications and architectures are adopting policy based management for tasks such as access control, network security and data privacy, the management and consolidation of a large number of policies is becoming a crucial component of such policy based systems. In large-scale distributed collaborative applications like web services, there is need for analysing policy interaction and performing policy integration. In this thesis, we propose and implement EXAM, a comprehensive environment for policy analysis and management, which can be used to perform a variety of functions such as policy property analyses, policy similarity analysis, policy integration etc. As part of this environment we have proposed two novel policy similarity analysis techniques along with fine-grained integration algebra (FIA) for integrating 3-valued access control policies. We also provide a framework for specifying policy integration constraints using FIA and generating an integrated policy. Our work focuses on analysis of access control policies written in the dialect of XACML (Extensible Access Control Markup Language) [1]. We consider XACML policies because XACML is a rich language which can represent many policies of interest to real world applications and is gaining widespread adoption in the industry.

1 INTRODUCTION

The widespread deployment of distributed applications and architectures such as Web Services, Virtual Private Networks and Service-Oriented Architectures has led to the proliferation of security policies that are specified and maintained independent of the application logic. Policy-based approaches enhance flexibility and reduce the application development time. Changes to the security requirements of an application simply entail modifying the policies without requiring changes to the application logic. Particularly important class of such security policies is represented by *access control policies* which determine whether requests to protected resources are permitted or denied. Various types of access control models and mechanisms have emerged, such as PolicyMaker [2], the ISO 10181-3 model [3] and the eXtensible Access Control Mark-up Language (XACML) [4].

A key requirement for the successful large scale deployment of policy-based access control services is represented by tools for managing and analyzing policies. Such tools must not only enable a policy administrator to easily acquire, edit and retrieve policies but must also support powerful analysis that are needed to perform tasks such as consistency checking, comparison and integration that are particularly crucial in the context of distributed collaborative applications.

A key goal for collaborative applications is to share resources, such as services, data and knowledge. Such applications may have different objectives, such as provisioning some complex service to third parties or performing some collaborative data analysis, and may adopt different collaboration mechanisms and tools. However, a common requirement is represented by the need for parties to compare their access control policies in order to decide which resources to share. For example, an important question that a party P may need to answer when deciding whether to share a resource with other parties in a coalition is whether these parties guarantee the same level of security as P. This is a complex question and approaches to it require developing adequate methodologies and processes,

and addressing several issues. A relevant issue is represented by a comparison of access control policies. A party P may decide to release some data to a party P' only if the access control policies of P' are very much the same as its own access control policies. Also as policies are increasingly being deployed at various levels within a distributed system - network devices, firewalls, hosts, applications - an important issue is to determine whether all the deployed policies authorize the same set of requests. Hence an important issue in the development of an analysis environment is devising techniques and tools for assessing *policy similarity*, that we define as the characterization of the relationships among the sets of requests respectively authorized by a set of policies. An important example of such relationship is represented by intersection, according to which one characterizes the set of common requests authorized by a set of given policies.

Another important requirement that arises in collaborative applications due to the need to integrate and share resources is the integration of access control policies. In order to define a common policy for resources jointly owned by multiple parties applications may be required to combine policies from different sources into a single policy. Even in a single organization, there could be multiple policy authoring units. If two different branches of an organization have different or even conflicting access control policies, what policy should the organization as a whole adopt? If one policy allows the access to certain resources, but another policy denies such access, how can they be composed into a coherent whole? Since no single policy integration strategy is known to be suitable for every possible situation it is important to have an effective policy integration mechanism that is able to support a flexible fine-graines policy integration strategy capable of handling complex integration specifications.

To date, no comprehensive environments exist supporting a large variety of query analysis and related management functions. Specialized techniques and tools have been proposed, addressing only limited forms of analysis (detailed discussion is presented in Chapter 5). Common limitations concern: policy conditions, in that only policies with simple conditions can be analyzed [5]; and relationship characterization, in that for example one

can only determine whether two policies authorize some common request, but no characterization of such request is provided.

The objective of this thesis is to address such lack by developing a comprehensive environment supporting a variety of analysis. This thesis focuses on the management and analysis of access control policies expressed in the dialect of XACML (eXtensible Access Control Mark-up Language) [1]. XACML has gained widespread adoption as an industry standard and can express many policies of interest to real world applications.

1.1 Contributions

The main contributions of this thesis is summarized below:

- We have identified and implemented several analysis queries for single and multiple policies.
- We have developed two variants of *policy similarity analysis* :
  - A *policy similarity measure*, which is a lightweight filter based on information retrieval techniques, that quickly computes an *approximate similarity score* for two access control policies. We have incorporated ontology matching techniques to solve the problem of attribute name and value heterogeneity when comparing policies. We have incorporated the notion of *predicate selectivity* to improve the effectiveness of the similarity scores. We have also conducted a pilot study among system administrators and students to validate the practical value of such an approach for policy comparison.
  - A heavyweight *policy similarity analyzer*, based on a combination of model-checking and SAT solving techniques, that gives a precise characterization of the similarity between policies at the granularity of requests that are permitted or denied. We have proposed and implemented a novel multi-level grid based visualization of results of policy similarity analysis.

We have implemented both variants of the policy similarity analysis and reported the experimental results.

- We have proposed a fine-grained integration algebra for language independent 3-valued policies. We introduce a notion of completeness and prove that our algebra is minimal and complete with respect to this notion. We have proposed and implemented a framework that uses the algebra for the fine-grained integration of policies expressed in XACML. The method automatically generates XACML policies as the policy integration result. To the best of our knowledge, none of the existing approaches has generated real policies as policy integration output. We have carried out experimental studies which demonstrate the efficiency and practical value of our policy integration approach.

- We have proposed EXAM (Environment for XACML policy Analysis and Management), a comprehensive environment for the management and analysis of access control policies expressed in XACML that incorporates the policy analysis and integration frameworks. We have developed EXAM as web-based application and made it available for use in a virtual appliance.

1.2 Background

In this section we review basic XACML concepts.

1.2.1 XACML Policies

XACML [4] is the OASIS standard language for the specification of access control policies. It is an XML language able to express a large variety of policies, taking into account properties of subjects and resources as well as context information. In general, a subject can request an action to be executed on a resource and the policy decides whether to deny or allow the execution of that action. Several profiles, such as a role profile, a privacy

profile etc. have been defined for XACML. Commercial implementations of XACML are also available [6, 7]. The main concepts in XACML policies are sketched as follows.

- The *Policy Target* identifies the set of requests that the policy is applicable to. It contains attribute constraints characterizing subjects, resources, actions, and environments.

- Each *Rule* in turn consists of another optional *Target*, a *Condition* and an *Effect* element. The rule *Target* has the same structure as the policy *Target*. It specifies the set of requests that the rule is applicable to. The *Condition* specifies restrictions on the attribute values in a request that must hold in order for the request to be permitted or denied as specified by the *Effect*. The *Effect* specifies whether the requested actions should be allowed (*Permit*) or denied (*Deny*).

  The restrictions specified by the target and condition elements correspond to the notion of attribute-based access control, under which access control policies are expressed as conditions against the properties of subjects and resources. In XACML such restrictions are represented as Boolean functions taking the request attribute values as input, and returning *true* or *false* depending on whether the request attributes satisfy certain conditions. If a request satisfies the policy target, then the policy is applicable to that request. Then, it is checked to see if the request satisfies the targets of any rules in the policy. If the request satisfies a rule target, the rule is applicable to that request and will yield a decision as specified by the *Effect* element if the request further satisfies the rule condition predicates. If the request does not satisfy the policy(rule) target, the policy(rule) is "Not Applicable" and the effect will be ignored.

- The *Rule combining algorithm* is used to resolve conflicts among applicable rules with different effects. For example, if a request is permitted by one rule but denied by another rule in a policy and the permit-overrides combining algorithm is used, the request will be permitted by the policy. If the deny-overrides combining algorithm is used, the request will be denied by the policy.

PolicyId=Pol1
<RuleId=R11 Effect=Permit>
<Target>
<Subject domain $belong\_to$ {".edu"} >
</Target>
<Condition $8:00 \leq$ Time $\leq 22:00$ >
</Rule>

Figure 1.1. Policy Pol1

- The *Policy set* is a set of XACML policies.

An XACML policy may also contain one or more *Obligations*,which represent functions to be executed in conjunction with the enforcement of an authorization decision. However, obligations are outside the scope of this work and we do not further consider them in this thesis.

Figure 1.1 shows the structure of an example policy, with identifier Pol1, which states that *any user from domain ".edu" is authorized to access the data from 8am to 10pm everyday*. This policy contains a single rule, R11, with the *Permit* effect. The rule target element restricts the users to be from domain ".edu", and the condition element restricts the access time. Such restrictions can be represented as the Boolean formulae:$(domain =$ "$.edu''$) $\wedge$ $(8 \leq time \leq 22)$. For an incoming request, if the attribute values of the request make the Boolean formulae true, a *permit* decision is returned.

The rest of this document is structured as follows : Chapter 3 presents an overview of EXAM's architecture and analysis capabilities. Chapter 4 presents in detail the two policy similarity analyses along with experimental results. The policy integration algebra is presented in Chapter 5. Related work is presented in Chapter 6 and finally Chapter 7 discusses conclusions and future work.

2 EXAM – AN OVERVIEW

In this chapter we discuss the various components in EXAM's architecture. We then present an illustrative example ch is used to present an overview of the various functions supported by EXAM. Then, we present some formal definitions of policy analysis queries.

2.1 EXAM Architecture

The EXAM environment, an overview of which is shown in Figure 2.1, includes three levels. The first level is the user interface, which receives policies, requests and queries from users, and returns request replies and query results. The second level is the query dispatcher, which handles various requests received from the user interface, dispatches them to proper analysis module and aggregates obtained results.

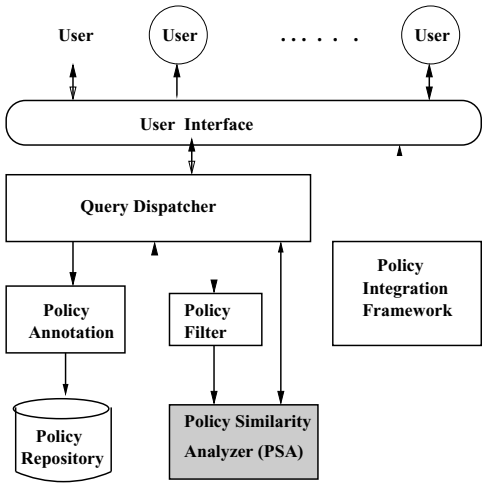


Figure 2.1. EXAM Architecture

The third level is the core level of EXAM and includes four modules supporting different functionalities, namely: *policy annotator*, *policy filter*, *policy similarity analyzer* and *policy integration framework*.

- The **Policy annotator** [8] module preprocesses each newly acquired policy by adding annotations to it. The annotations explicitly represent the behavior or semantics of each function referred in the policy. Such annotations help in automatically translating policies into Boolean formulae that can then be evaluated by the policy analysis modules. The annotated policies are stored in the policy repository together with the policy metadata.

- The **Policy filter** [9] implements a lightweight approach which quickly evaluates similarity between each pair of policies and assigns each pair a similarity score ranging from 0 to 1. The higher the similarity score is, the more similar the two policies are. According to the obtained similarity scores, policies with low similarity scores may be pruned from further analysis, whereas policies with high similarity scores can be further examined. The main goal of the policy filter module is to reduce the number of policies that need to be analyzed more in details, when dealing with large size policy sets. The filtering approach we use is based on techniques from information retrieval and is extremely fast.

  The use of filtering in the policy analysis process is however optional. The query dispatcher can directly send analysis queries to the policy similarity analyzer, to carry out a fine-grained policy analysis, without performing the filtering.

- The **Policy similarity analyzer (PSA)** module is the core component of our approach to policy analysis. It basically implements the strategies for processing the policy analysis queries supported by EXAM, and thus in the subsequent sections we describe in details its main techniques and query processing strategies.

- The **Policy integration framework** [10] module provides the capability for integrating policies using operators defined in the proposed fine grained integration algebra (FIA). It uses the MTBDD based representation of policies to implement the algebraic operators. Operations on policies are mapped onto operations on the corresponding policy MTBDDs and an integrated policy MTBDD is obtained. The inte-

grated policy MTBDD is then traversed to generate a well-formed XACML version of the integrated policy.

It is worth noting that our system is flexible and supports an easy integration of new functions. Even though its current version applies only to XACML policies, it can be easily adapted to deal with other policy languages. The policy repository, policy filter and policy integration modules need to be modified for supporting other policy languages.

2.2 An Illustrative Example

To illustrate the discussion we consider a scenario from a content delivery network (CDN) system built on P2P network in which parties can replicate their data in storage made available by third party resource providers. Real systems adopting this model are for instance Lockss [11] and LionShare [12]. In such scenario, there are usually two types of parties: data owner and resource owner. A data owner owns some data, whereas a resource owner manages some resources for storing data and processing queries on data. A data owner typically needs to determine which resource owners can be more suited for storing its content. Examples of such CDN systems can be found in Grid computing systems and P2P systems. Each such party in a CDN typically has its own access control policies. The policies of a data owner specify which users can access which data, among these owned by the data owner, under which conditions. The access control policies of the resource owners specify conditions for the use of the managed resources. In large dynamic environments, we cannot expect such policies to be integrated and harmonized beforehand, also because policies may dynamically change. Therefore, a subject wishing to run a query has to comply with both the access control policy associated with the queried data and the access control policy of the resource to be used to process the query. Because such parties may not have the same access control policies, in order to maximize the access to the data, it is important to store the data at the resource owner having access control policies similar to the access control policies associated with the data. Furthermore, besides determining the common parts of the access control policies shared by the data owner and resource owner,

the data owner may also be interested in checking if certain key requests can be successfully handled, if the data were to be located at a given resource owner. In other words, the data owner may want to know if the difference among the multiple access control policies has a negative effect on some important tasks.

We now introduce two example policies from the above scenario.

Example 1 Pol1 (Data Owner): Any user from domain "**.edu**" is authorized to access the data from **8am** to **10pm** everyday.

Pol2 (Resource Owner): Any user from domain "**.edu**" or affiliated with "**IBM**" is authorized to access the resource from **6am** to **8pm** everyday.

In order for the data owner to decide whether to store the data at the resource owner, it is crucial to determine which kinds of requests will be permitted by both policies and which will not. Because in this case we are dealing with only two policies, the filtering phase is not required and the policies can be directly transmitted to the PSA. The PSA then returns the following characterization of the similarity for the input policies:

- When *domain* is in the name space of ".edu" and *time* is in the range of [8am, 8pm], such requests are permitted by Pol1 and Pol2.
- When *domain* is in the name space of ".edu" and *time* is in the range of (8pm, 10pm], such requests are permitted by Pol1, denied by Pol2.
- When *affiliation* is "IBM" and *time* is in the range of [6am, 8pm], such requests are denied by Pol1, permitted by Pol2.
- When (*domain* is in the name space of ".edu" or *affiliation* is "IBM"), and *time* is in the range of [6am, 8am), such requests are denied by Pol1, permitted by Pol2.

By using such characterization, the data owner can check if most requests (or some important requests) are satisfied and then decide whether to send his data to such resource owner. More specifically, if the data owner knows that a large percentage of requests are issued during the time interval [8am, 8pm], sending the data to this resource owner would

be a good choice, as both policies yield same effect for the requests during that time period. If, instead, it is crucial that the data be also available at other times, the data owner may determine if there are other resource owners, whose policies are closer to its own, or use some data replication strategies to make sure that at any point of time there is at least one resource owner whose policies allow the data queries to be processed.

The data owner may also investigate additional properties concerning the policies; for example the data owner may also issue queries like *"when are the requests of users from domain ".edu" permitted by both policies?"*. For such query the PSA will return the time interval in which both policies are satisfied.

2.3 Analysis Queries on Policies

In this section, we present formal definitions of policy analysis queries that are supported by EXAM. Because one can analyze policies and sets of policies from different perspectives, it is important to devise a comprehensive categorization of such queries. In our work, we have thus identified three main categories of analysis queries, which differ with respect to the information that they query. These categories are: *policy metadata queries*, *policy content queries*, and *policy effect queries*. Figure 2.2 provides a taxonomy summarizing the various query types.

Policy metadata queries analyze metadata associated with policies, such as policy creation and revision dates, policy author, and policy location. A policy content query, by contrast, extracts and analyzes the actual policy content, such as the number of rules in the policy, the total number of attributes referenced in the policy, the presence of certain attribute values.

A policy effect query analyzes the requests allowed or denied by policies and interactions among policies. The category of the policy effect queries is the most interesting one among the query categories we have identified. The processing of policy effect queries is also far more complex than the processing of queries in the other two categories, and thus we address its processing in details (see next section). The policy effect query category can

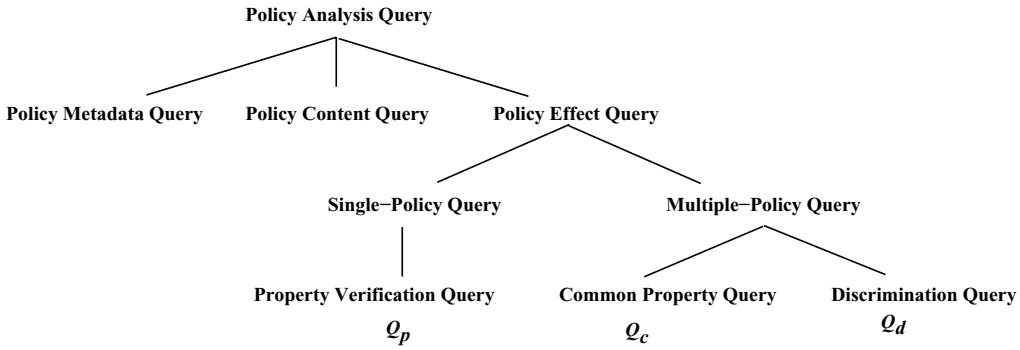


Figure 2.2. Query Categorization

be further divided into two subcategories: (i) queries on single policy; and (ii) queries on multiple policies. The first subcategory contains one type of query, referred to as *property verification query*. The second subcategory contains two main types of queries, namely *common property query* and *discrimination query*.

In the following, we first introduce some preliminary notions, and then present more details for each type of policy effect query (query for short), including their definitions and functionalities.

2.3.1 Preliminary Notions

In our work, we assume the existence of a finite set A of names. Each attribute, characterizing a subject or a resource or an action or the environment, has a name a in A, and a domain, denoted by $dom(a)$, of possible values. The following two definitions introduce the notion of access request and policy semantics.

Definition 1 Let a_1, a_2, ..., a_k be attribute names in policy P, and let $v_i \in dom(a_i)$ ($1 \leq i \leq k$). $r \equiv \{(a_1, v_1), (a_2, v_2), \cdots, (a_k, v_k)\}$ is a request, and e_r^P denotes the effect of this request against P .

Example 2 Consider a policy that permits access to data to users from the ".edu" domain between 8am and 10pm. An example of a request to which this policy applies is that of

a user from domain ".edu" wishing to access the data at 9am. According to Definition 1, such request can be expressed as $r \equiv \{(domain, ".edu"), (time, 9am)\}$.

Definition 2 Let P be an access control policy. We define the semantics of P as a pair $\{B_{permit}, B_{deny}\}$, where B_{permit} and B_{deny} are Boolean expressions corresponding to permit and deny rules respectively. B_{permit} and B_{deny} are defined as follows.

$$\begin{cases} B_{permit} = T_P \wedge ((T_{PR_1} \wedge C_{PR_1}) \vee ... \vee (T_{PR_k} \wedge C_{PR_k})) \\ B_{deny} = T_P \wedge ((T_{DR_1} \wedge C_{DR_1}) \vee ... \vee (T_{DR_j} \wedge C_{DR_j})) \end{cases}$$

where, T_P denotes a Boolean expression on the attributes of the policy target; T_{PR_i} and C_{PR_i} ($i = 1, ..., k$) denote the Boolean expressions on the attributes of the rule target and rule condition of permit rule PR_i; and T_{DR_i} and C_{DR_i} ($i = 1, ..., j$) denote the Boolean expressions on the attributes of the rule target and rule condition of deny rule DR_i.

The Boolean expressions (T and C) that frequently occur in policies can be broadly classified into the following five categories, as identified in [13] :

- Category 1: One variable equality constraints.

  $x = c$, where x is a variable and c is a constant.

- Category 2: One variable inequality constraints.

  $x \rhd c$, where x is a variable, c is a constant, and $\rhd \in \{<, \leq, >, \geq\}$.

- Category 3: Real valued linear constraints.

  $\sum_{i=1}^{n} a_i x_i \rhd c$, where x_i is variable, c_i is a constant, and $\rhd \in \{=, <, \leq, >, \geq\}$.

- Category 4: Regular expression constraints.

  $s \in L(r)$ or $s \notin L(r)$, where s is a string variable, and $L(r)$ is the language generated by regular expression r.

- Category 5: Compound Boolean expression constraints.

This category includes constraints obtained by combining Boolean constraints belonging to the categories listed above. The combination operators are $\vee$, $\wedge$ and $\neg$. By using $\neg$, we can represent the inequality constraint $x \neq c$ as $\neg(x = c)$.

It is worth noting that Boolean expressions on the attributes of policy targets or rule targets (T_P, T_{PR}) usually belong to Category 1.

The domains of the attributes that appear in the above Boolean expressions belong to one of the following categories :

- Integer domain : The attribute domains in Boolean expressions of categories 1,2 and 5 can belong to this domain.

- Real domain : The attribute domains in Boolean expressions of categories 1,2,3 and 5 can belong to this domain.

- String domain : The attribute domains in Boolean expressions of categories 1, 4 and 5 can belong to this domain.

- Tree domain : Each constant of a tree domain is a string, and for any constant in the tree domain, its parent is its prefix (suffix). The X.500 directories, Internet domain names and XML data are in the tree domain. For example, an Internet domain constant ".edu" is the parent of an Internet domain constant "purdue.edu". The attribute domains in Boolean expression of categories 1 and 5 can belong to this domain.

At the end, we define the degree of similarity between two policies as follows.

Definition 3 Let P_1 and P_2 be access control policies. The degree of similarity between P_1 and P_2 is the percentage of requests which are granted the same decisions (Permit or Deny) by the two policies.

According to Definition 4.2.2, we say that P_1 and P_2 are equivalent if P_1 always permits (or denies) a request that P_2 permits (or denies) and vice versus.

2.3.2 Policy Effect Query

Definition 4 *Let P_1, P_2, ..., P_n be n ($n \geq 1$) policies. A policy effect query has the form: $\langle B_q, (e_{q_1}, e_{q_2}, ..., e_{q_n}), f_q \rangle$, where B_q is a Boolean function on a subset of attributes occurring in the n policies, $e_{q_i} \subset \{$Permit, Deny, NotApplicable[1]$\}$ ($1 \leq i \leq n$), and f_q is a Boolean expression on the number of requests.*

To evaluate a policy effect query, we first find the requests that satisfy B_q. For each such request, we obtain the decisions from n ($n \geq 1$) policies and compare the decisions with e_{q_1}, ..., e_{q_n}. If the decision yielded by P_i is an element in e_{q_i} ($1 \leq i \leq n$), insert the request to a result set R. The last step is to check f_q. Currently, our system supports two types of f_q functions (i)$true$, which means there is no constraint; (ii)$|R| \lhd x$ ($\lhd \in \{<, \leq, =, =, >, \geq\}$), where $|R|$ is the number of requests and x is a constant. For example, $|R| > 0$ is a query constraint which checks if the corresponding query returns at least one request. It is worth noting that f_q can be a more complicated function on a particular set of attributes. Such flexibility in the definition on f_q allows our query language to cover various situations. The output of a policy effect query is a value "true" and a set of requests when f_q is satisfied, otherwise the output is "false". In what follows, we show how to represent property verification query, common property query and discrimination query through examples.

Property verification query (Q_p). It checks if a policy can yield specified decisions given a set of attribute values and constraints.

Example 3 Consider a scenario from a content delivery network(CDN) built on P2P network, e.g. Lockss [11] and LionShare [12], in which parties can replicate their data in storage made available by third party resource providers. There are usually two types of parties: data owner and resource owner. The policies of a data owner specify which users can access which data, among these owned by the data owner, under which conditions. The access control policies of the resource owners specify conditions for the use of the managed resources.

[1]We do not distinguish "NotApplicable" and "Non-determinism" in this work.

For example, P_1 is a policy of a data owner who allows any user from domain "**.edu**" to access his data from **8am** to **10pm** everyday. P_2 is a policy of a resource owner who allows any user from domain "**.edu**" or affiliated with "**IBM**" to access his machine from **6am** to **8pm** everyday, and allows his friend Bob to access his machine anytime if the sum of uploading and downloading file sizes is smaller than 1GB. According to Definition 4.2.2, policy P_1 and P_2 are represented as functions (1) and (2) respectively[2].

$$\begin{cases} B_{permit} = (\ (\ (domain = \text{“}.edu\text{”}) \\ \qquad\qquad \wedge (8am \leq time \leq 10pm)\ )\ ) \\ B_{deny} = FALSE \end{cases} \tag{2.1}$$

$$\begin{cases} B_{permit} = (\ (\ (domain = \text{“}.edu\text{”} \vee \\ \qquad\qquad affiliation = \text{“}IBM\text{”}) \\ \qquad\qquad \wedge (6am \leq time \leq 8pm)\ ) \\ \qquad\qquad \vee (\ (user = \text{“}Bob\text{”}) \\ \qquad\qquad \wedge (upload + download < 1GB)\ )\ ) \\ B_{deny} = FALSE \end{cases} \tag{2.2}$$

Suppose that the resource owner would like to carry out system maintenance in the time interval [10pm,12am], and hence he may want to check if policy P_2 will deny any external access to the resource between 10pm and 12am. Such a query can be expressed as follows:

$Q_p \equiv \langle 10pm \leq time \leq 12am, (\{\text{Permit}\}), |R| = 0 \rangle$.

The query first checks if any request with the time attribute in the range of 10pm and 12pm is permitted, and stores such requests in R. Then, the query verifies the constraint f_q. In this example, some requests from "Bob" during [10pm,12am] will be permitted and hence R is not empty which violates f_q. The query will return "false" as the final answer.

Common property query (Q_c). It can be used to find the common properties shared by multiple policies i.e., to find some or all requests which have the *same* effect in all the policies. These type of queries are particularly useful in the context of large dynamic

[2]The predicates on *domain*, *affiliation* and *user* attributes belong to the rule targets and the predicates on *time*, *upload* and *download* attributes belong to rule conditions.

environments where one cannot expect policies to be integrated and harmonized beforehand and one needs to determine if multiple parties actually provide similar level of security for protected resources. For example, a data owner who wishes to host his data on some remote machine and wants to ensure that certain subjects can always access his data can use a common property query that analyses the data owner's access control policy and the host machine's access control policy to check if requests with certain subjects are always *permitted* by both policies.

Example 4 Consider P_1 and P_2 in Example 3, an example common property query is to find all the requests permitted by both policies, which is written as

$Q_c \equiv \langle true$, ({Permit}, {Permit}), $true\rangle$.

In this query, B_q and f_q are $true$, which means there is no constraint on the attributes of a request. "({Permit}, {Permit})" indicate that any request be permitted by both policies will be returned as an answer.

The following example shows a common property query with constraints on the attributes of a request.

Example 5 Determine when the requests of users from domain ".edu" are permitted by policy P_1 and P_2. This query consists of two parts. First, we need to find all requests of users from domain ".edu" that can be permitted by both policies, which is a common property query with the constraint on the *domain* attribute. It can be written as

$Q_c \equiv \langle domain =$ ".edu", ({Permit}, {Permit}), $true\rangle$.

Suppose that Q_c returns the result set R as follows:

R={$\langle domain =$ ".edu", $8am \leq time \leq 8pm$, $Permit$, $P_1\rangle$, $\langle domain =$ ".edu", $8am \leq time \leq 8pm$, $Permit$, $P_2\rangle$, $\langle domain =$ ".edu", $8pm < time \leq 10pm$, $Permit$, $P_1\rangle$, $\langle domain =$ ".edu", $6am \leq time < 8pm$, $Permit$, $P_2\rangle$}.

The second step is to post process R and extract the subsets of R with common *time* attribute, and the final answer is $8am \leq time \leq 8pm$.

Discrimination query (Q_d). Besides determining the common parts of the access control policies shared by multiple parties, one party may also be interested in checking if certain

key requests can be successfully handled by its potential collaborators. In other words, one may want to know if the difference among the multiple access control policies has a negative effect on some important tasks. A discrimination query is thus used to find the difference between one policy and the others.

Example 6 A patient needs to be transferred from a local hospital to a specialistic hospital. He is satisfied with the privacy policies in the local hospital because, for example, the local hospital protects patient data from being used for lab research without the patient agreement. Before the transfer, he wants to make sure that the specialistic hospital will also well protect his medical data. He can then issue a discrimination query like $Q_d \equiv \langle true$, ({Deny}, {Permit}), $true\rangle$ to find out the requests denied by the local hospital's policy but permitted by the specialist hospital's policy.

For example, suppose that the local hospital's policy states that patient medical data will not be accessed by any third party, while the specialist hospital's policy states that patient medical data may be used by research labs. Using Q_d, the patient will be able to find out this difference.

Both the common property query and the discrimination query focus on a partial view of policies. The common property query only considers the intersections of request sets with the same effects, and the discrimination query only considers the mutually exclusive request sets. To obtain an overview of relationships between policies, we combine the common property queries and discrimination queries. Example 7 shows how to check policy equivalence.

Example 7 To determine whether P_1 is the same as P_2, i.e. for any request r, P_1 and P_2 yield the same effect, we can use the following set of discrimination queries.

$$\begin{cases} Q_{d1} \equiv \langle true, (\{Permit\},\{Deny, NotApplicable\}), |R| = 0\rangle \\ Q_{d2} \equiv \langle true, (\{Deny\},\{Permit, NotApplicable\}), |R| = 0\rangle. \\ Q_{d3} \equiv \langle true, (\{NotApplicable\},\{Permit, Deny\}), |R| = 0\rangle. \end{cases}$$

Q_{d1} checks if there exists any request permitted by P_1 but not permitted by P_2. Q_{d2} and Q_{d3} check the other two effects. When all queries return "true", P_1 equals P_2. Note that though there are multiple queries, they can be executed simultaneously (see Section 3.2.4).

Similarly, we can also use combinations of queries to represent other relationships like policy inclusion, policy incompatibility and policy conflict. In particular, policy inclusion means: for any request r that is applicable to P_1, if P_1 and P_2 yield the same effect for r, we say P_1 is included by P_2. Policy incompatibility means: there exists a request r such that P_1 and P_2 yield different effects; also there exists a request r such that P_1 and P_2 yield same effect. Policy conflict means: for every request r that is applicable to P_1 and P_2, P_1 and P_2 yield different effects.

From the previous discussion, we can observe that the execution of each policy query essentially corresponds to the evaluation of a set of requests. For clarity, we would like to distinguish a policy query from general requests in two aspects. First, a policy query usually specifies some constraints on some attributes. A request that only contains the specified attributes is not sufficient for evaluating the policy property, because the policy will consider other attributes as "don't care" and most possibly yields the effect "Not Applicable". Therefore, for a policy query, we need to consider all possible combinations of value assignments for the attributes that are not specified in the query. Second, a policy query often needs to analyze a set of requests. It may not be efficient to treat these requests separately.

2.4 System Demonstration – A tour of EXAM

In this section we present screen shots of the EXAM environment and illustrate EXAM's various functionalities with the help of two policies *policy1* and *policy2* shown in Example 8 and Example 9.

Example 8 *The policy* policy1 *permits any action for users of the* edu *domain between 8am and 10pm and denies any action to the user* bob *if download is greater than 1GB or the upload is greater than 3GB. Also, this policy uses a deny-overrides rule combining algorithm. A succinct representation of this policy is as follows:*

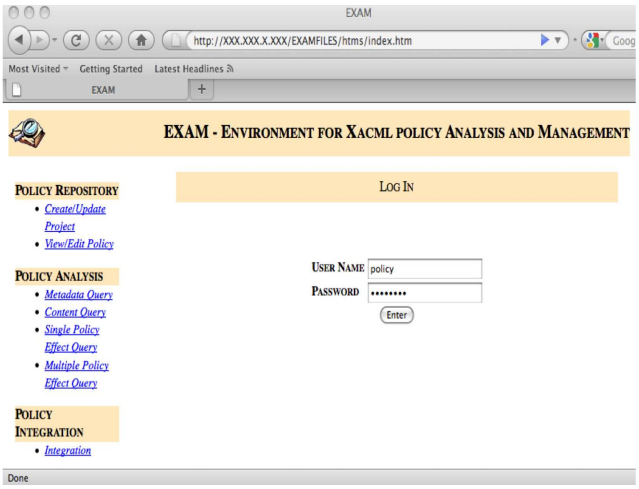


Figure 2.3. EXAM – The login screen.

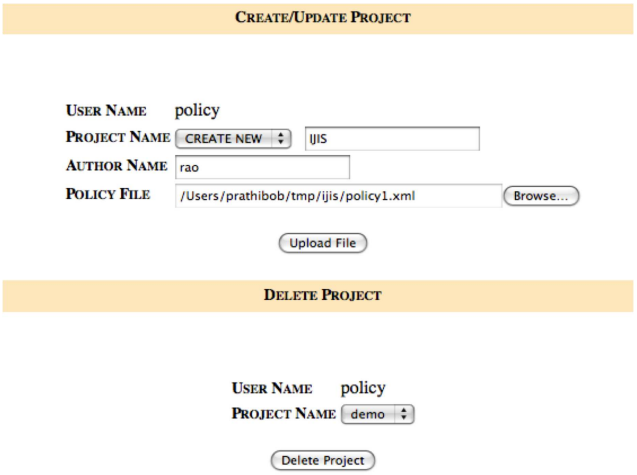


Figure 2.4. EXAM – Create New Project Screen.

$$\begin{cases} B_{permit} = (\,(\,(domain = \text{“}.edu\text{”} \wedge action = \text{“}Any\text{”}) \\ \qquad\qquad \wedge (8 \leq time \leq 22)\,)\,) \\ B_{deny} = (\,(\,(user = \text{“}bob\text{”} \wedge action = \text{“}Any\text{”}) \\ \qquad\qquad \wedge (download > 1 \vee upload > 3)\,)\,) \end{cases}$$

Example 9 *The policy* policy2 *permits any action for users of the* edu *domain between 6am and 8pm and denies any action to the user* bob *if the sum of upload and download is*

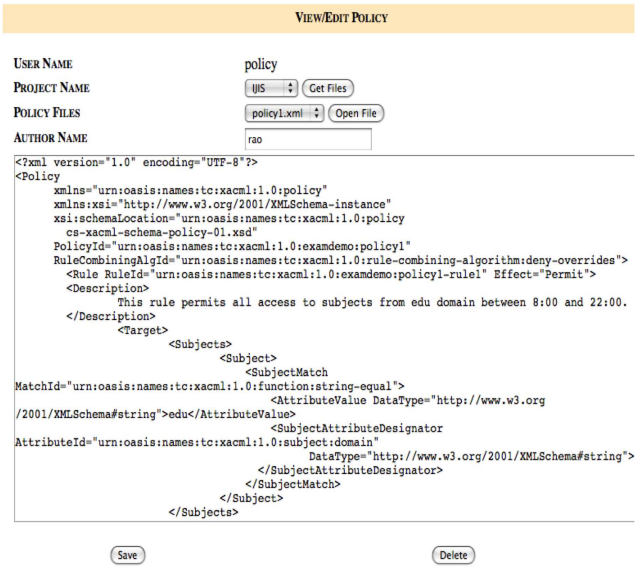


Figure 2.5. EXAM – View/Edit Policy Screen.

greater than 2GB and the upload is less than 1 GB. This policy also uses a deny-overrides rule combining algorithm. A succinct representation of this policy is as follows:

$$\begin{cases} B_{permit} = (\ (\ (domain = \text{“}.edu\text{”}) \wedge (action = \text{“}Any\text{”}) \\ \qquad\qquad \wedge (6 \leq time \leq 20)\ )\ ) \\ B_{deny} = (user = \text{“}bob\text{”}) \wedge (action = \text{“}Any\text{”}) \\ \qquad \wedge (upload + download > 2 \wedge upload < 1)\ )\ ) \end{cases}$$

A policy administrator who wishes to manage and analyze a set of policies must first log on to the environment as shown in Figure 2.3. After log in, the administrator can choose to create, update or delete projects as shown in Figure 2.4. A new project can be created by uploading XACML policy files. The XACML files are parsed and information is stored in various tables in a MySQL database. These tables are used to answer the metadata and content queries. Currently a new project **IJIS** has been created and populated with three files *policy1.xml*, *policy2.xml* and *policy3.xml*. The files *policy1.xml* and *policy2.xml* correspond to the policies discussed in Example 8 and Example 9 respectively which we

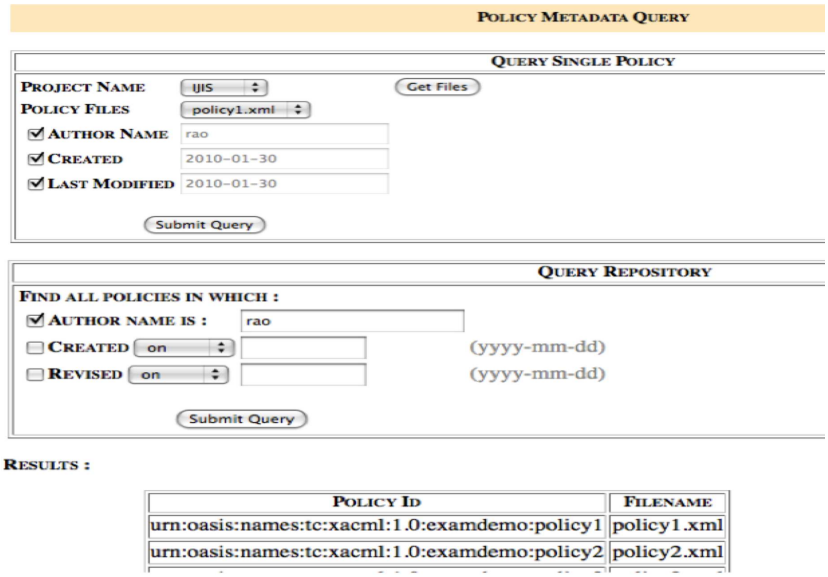


Figure 2.6. EXAM – Metadata Query Screen.

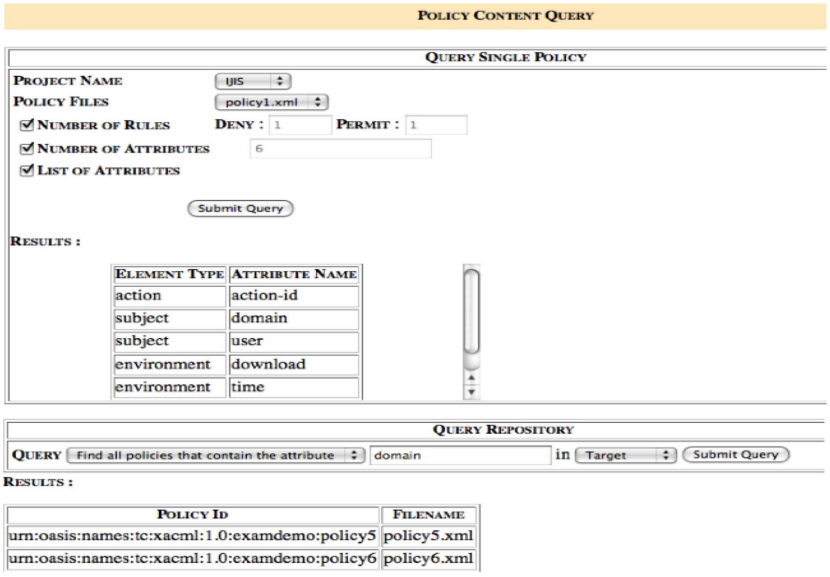


Figure 2.7. EXAM – Content Query Screen.

will be using for further discussion in this section. Policies in a selected project can be viewed and/or modified using the interface shown in Figure 2.5.

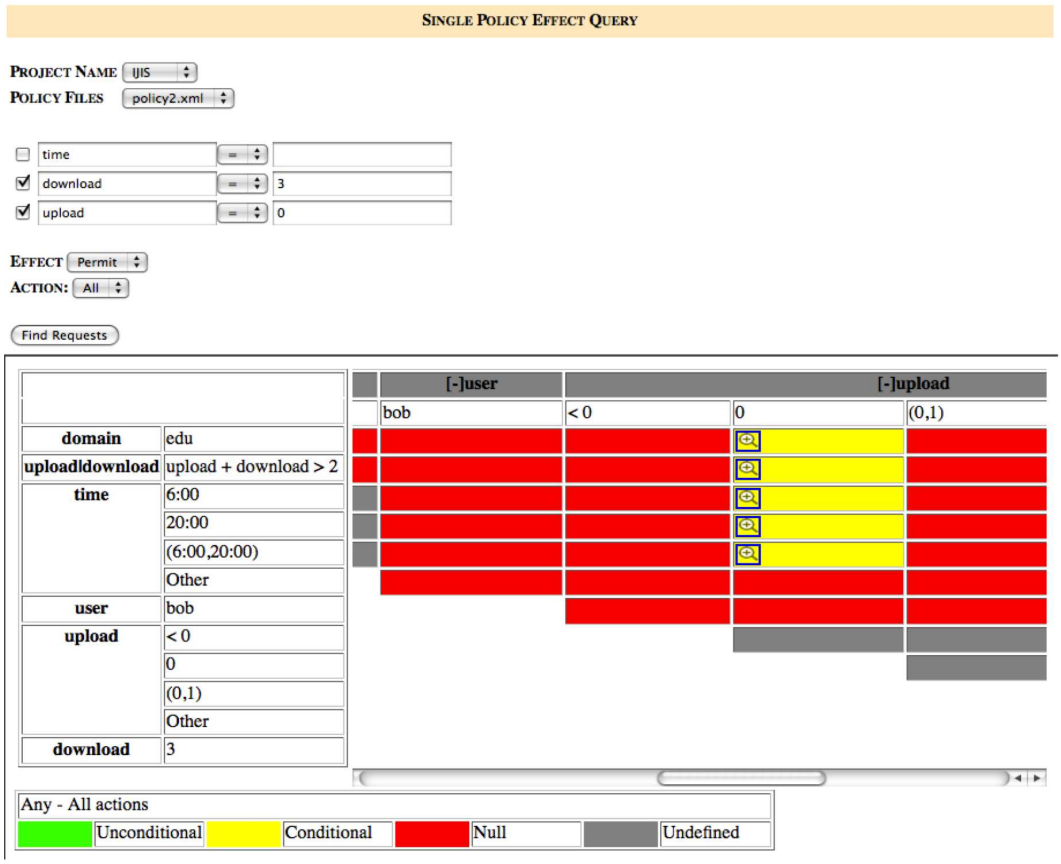


Figure 2.8. EXAM – Single Policy Query Screen.

An administrator can query specific policy files for metadata or query the entire policy repository to find policies with specific values of the various metadata attributes. In Figure 2.6, the author name, date of creation and date of modification for the policy file *policy1.xml* is being queried. In addition, the policy repository is also being queried for all policies with a specific author name.

Figure 2.7 shows the interface that can be used for performing content queries. Here an administrator can query specific policy files for information like the number of deny (permit) rules, the number of different attribute names referenced in the policy and the list of attributes. Figure 2.7 shows the content query results for the policy file *policy1.xml* which has 1 permit rule, 1 deny rule and total of 6 different attribute names referenced in the policy. In addition, an administrator can also query the entire policy repository for

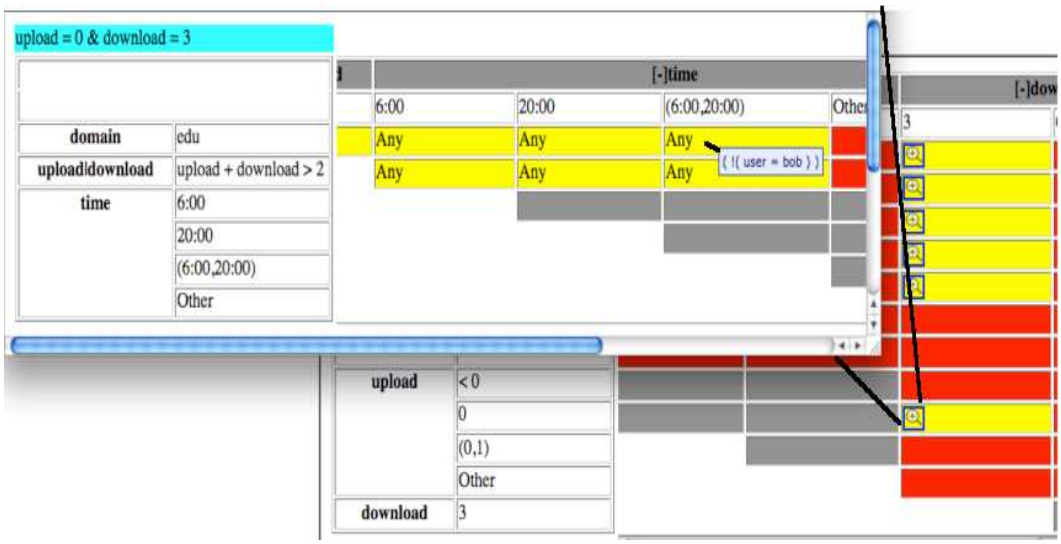


Figure 2.9. EXAM – Result of a Single Policy Query.

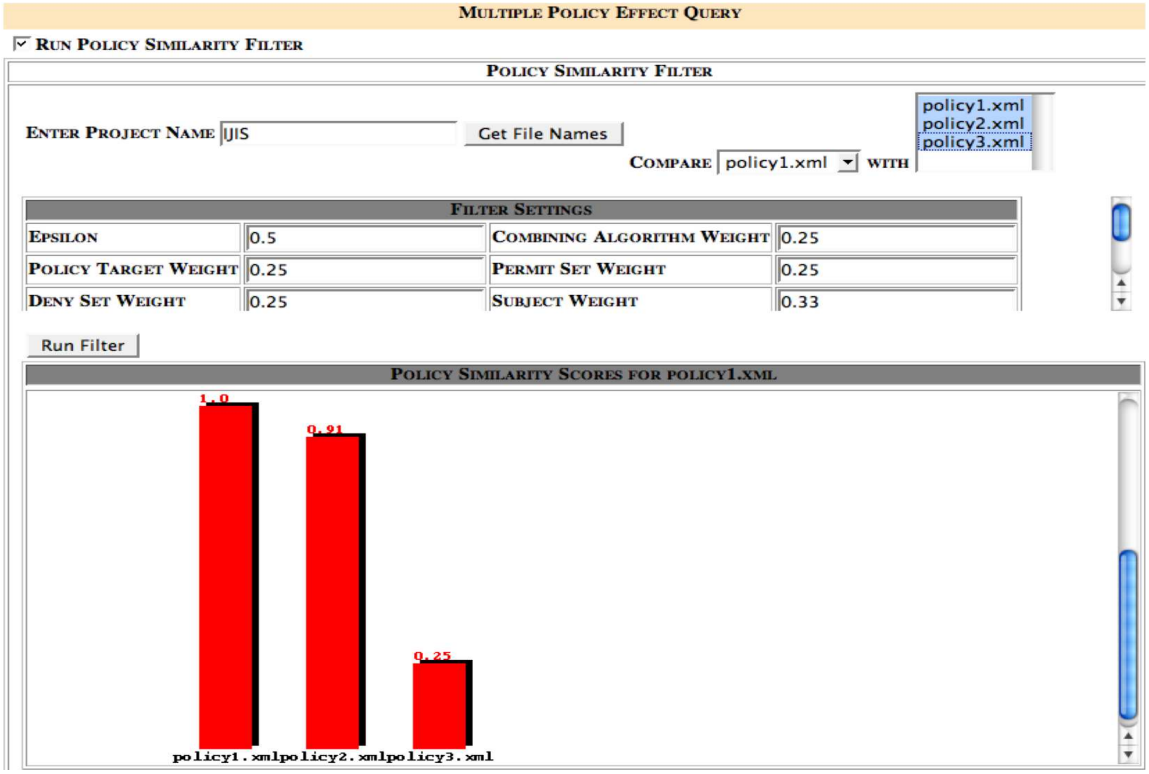


Figure 2.10. EXAM – Policy Similarity Filter Screen.

policy files with specific contents like presence of target elements and presence of certain attribute names in specific elements of the policy. Figure 2.7 shows a list of all policies in the repository which reference that *domain* attribute.

The interface shown in Figure 2.8 can be used to perform single policy property verification queries. An administrator can select the policy file he wishes to query and can select the constraints on the attributes appearing in the policy and the desired effect. The set of requests satisfying the query (if any) can be visualized as a multi-level two dimensional grid whose rows and columns represent predicates on attributes appearing in the policy being queried. For better clarity, the administrator can view the grid for one action at a time. A cell (p_i, p_j) in the grid represents requests that satisfy the predicates p_i and p_j and is associated with a color. A *red* cell indicates that no requests satisfying the corresponding predicates were found. A *grey* cell represents the fact that there can be no requests (regardless of the query) that can satisfy the corresponding predicates. A *green* cell represents requests that are characterized by exactly the two predicates p_i and p_j. A *yellow* cell represents requests which are characterized by more predicates in addition to p_i and p_j. If there are requests which are characterized by just one additional predicate, the additional predicate is displayed in a *tool-tip* that is displayed when the cursor is placed on the cell. Requests that are characterized by more than one additional predicates are represented by yellow cells that contain the *zoom-in* icon hyperlink. On clicking this hyperlink a popup window opens up which shows a smaller 2-D grid (not including p_i and p_j) indicating the other predicates on which the requests depend. In addition, the *green* and *yellow* cell with tool-tip contain an action code corresponding to the action that the user has chosen to view. More details on this type of policy analysis visualization technique is discussed in sub section 3.2.6. Figure 2.8 shows the results for the following query on *policy2* : $Q_p \equiv \langle((upload = 0) \wedge (download = 3)), (\{Permit\}), \textit{true}\rangle$.

The grid corresponds to any action indicated by the selection *All* in the action drop-down list. [3] Since *policy2* uses a deny-overrides algorithm no request must be permitted for user *bob* when $upload = 0$ and $download = 3$. Hence we can observe in Figure 2.8

[3]The keyword *Any* in the figures denotes any type of action like read, write, update, etc.

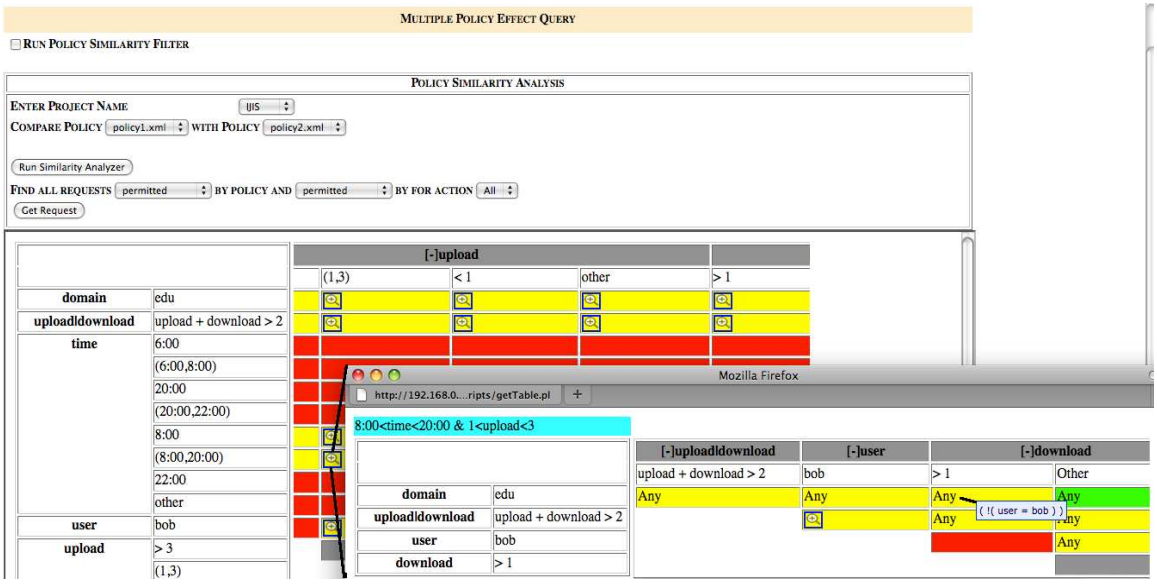


Figure 2.11. EXAM – Multiple Policy Effect Query Screen.

that all cells in the column(row) corresponding to the predicate $user = bob$ are colored *red* indicating that no such requests were found thus not permitting any action. Another example of a request satisfying this query is illustrated in Figure 2.9. Here we observe that any action is allowed during 6am and 8pm for the *edu* domain provided the user is not *bob* (indicated by the predicate $!(user = bob)$ in the tool-tip).

The interface for the policy similarity filter is shown in Figure 2.10. An administrator can choose a specific policy and a set of policies which he wishes to compare it with. The administrator can also specify the weights to be assigned to the different policy elements when calculating the similarity score. A bar graph illustrating the similarity scores (a value between 0 and 1) is displayed. Figure 2.10 shows the results of comparing *policy1* with itself, *policy2* and *policy3*[4]. The corresponding similarity scores obtained are 1.0, 0.91 and 0.25 respectively. The higher similarity with *policy2* as compared to the score obtained with *policy3* can be attributed to the fact that *policy1* and *policy2* have more similar structure

[4]The policy *policy3* has one permit rule that allows a subject from the *domain* edu *write* access to files of *type* source

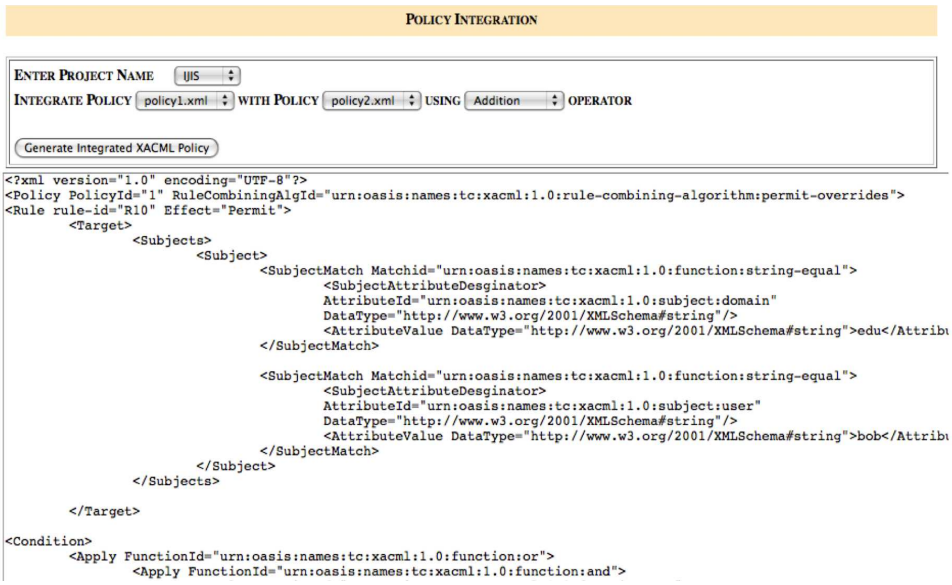


Figure 2.12. EXAM – Policy Integration Screen.

and common attributes. More details of how a similarity score is obtained between two policies can be found in [9].

An administrator can further perform an exhaustive comparison between pairs of policies by using the common property and discrimination queries interface, which is shown in Figure 2.11. Here an administrator can specify the two policies he wishes to query and the effects corresponding to the common property or discrimination queries.

The set of requests satisfying the query are displayed in a multi-level two dimensional grid similar to the results for single policy effect query.

Figure 2.11 shows the results of performing the following common property query between *policy1* and *policy2*:

$Q_c \equiv \langle true, (\{\text{Permit}\}, \{\text{Permit}\}), true \rangle$.

An example of a request (shown in Figure 2.11) that is permitted by both the policies is a request that arrives between 8am and 8pm by a user other than *bob* from the domain edu and the value of the upload attribute is between 1 and 3.

Figure 2.12 shows the interface for performing integration of two policies using operators defined in a fine-grained integration algebra. An administrator can specify the two

policies he wishes to integrate and the operator he wishes to use for integration. An integrated XACML policy is generated and displayed. Figure 2.12 shows the XACML policy generated as a result of integrating *policy1* and *policy2* using the *Addition* operator. More details on policy integration can be found in [10].

3 POLICY SIMILARITY ANALYSIS

A key goal for collaborative applications is to share resources, such as services, data and knowledge. Such applications may have different objectives, such as provisioning some complex service to third parties or performing some collaborative data analysis, and may adopt different collaboration mechanisms and tools. However, a common requirement is represented by the need to assure security for the shared resources. It is important that collaboration does not undermine security of the collaborating parties and their resources. Security however should not drastically reduce the benefits deriving from the collaboration by severely restricting the access to the resources by the collaborators. An important question that a party P thus may need to answer when deciding whether to share a resource with other parties is whether these other parties guarantee the similar level of security as P. This is a complex question and approaches to it require developing adequate methodologies and processes, and addressing several issues. One relevant issue is the comparison of access control policies; access control policies govern access to protected resources by stating which subjects can access which data for which operations and under which circumstances. Access control represents a key component of any security mechanism. A party P may decide to release some data to a party P' only if the access control policies of P' are very much the same as its own access control policies. It is important to notice that an approach under which P just sends its policies together with the data to P' so that P' can directly enforce these policies may not always work. The evaluation of P's policies may require accessing some additional data that may not be available to P' for various reasons, for example for confidentiality, or P may not just be able to share, for confidentiality reasons, its policies with P'.

More complex situations arise when several alternative resources and services, each governed by its own independently-administered access control policies, may have to be selected and combined in a complex service. In order to maximize the number of requests

that can be satisfied by the complex service, and also satisfy the access control policies of each participating resource and service, one would like to select a combination of resources and services characterized by access control policies that are very much similar. As an example consider the case of a grid computing system consisting of data owners and resource owners, each with its own access control policies [14]. For a subject to be able to run a query on the data, this subject must verify both the access control policy associated with the queried data and the access control policy of the resource to be used to process the query. It is often the case that such parties do not have exactly the same access control policies; therefore in order to maximize the access to the data, the data for processing should be stored at the resource having access control policies similar to the access control policies associated with the data.

Policy similarity can be defined as the *characterization of the relationships among the sets of requests respectively authorized by a set of policies*. The goal of policy similarity analysis is to provide such a characterization. The result of a policy similarity analysis between a set of policies can be used to derive for all requests of the form (s, a, r, p) the corresponding effects of the policies in the set and thus to compare the behavior of policies in the set with respect to all possible requests. Such an analysis can be used to answer the *common property* and *discrimination* queries which could be helpful during the harmonization of security and privacy policies. It can also be used for change impact analysis where an administrator may want to verify the effect of changes to current policies [5] or find differences among rules.

A brute force approach to determine policy similarity is to simply evaluate both policies for any request and any assignment, and then compare the results. Obviously, such an approach is very inefficient and even infeasible when the request domain is infinite. Approaches based on model checking [5] and SAT-solving [13] have been proposed. However, these approaches have several shortcomings. The technique in [5] supports only equality conditions on attributes and is restricted to the *string* domain and hence is not capable of handling complex conditions that are prevalent in most real-world application policies. The SAT-solver based technique [13] is able to handle complex conditions but is only able to say

if there is *atleast* one request that is commonly authorized by a set of policies as opposed to enumerating all such requests.

In this thesis, we propose two techniques to determine similarity between policies. The first technique is a *policy similarity filter* that uses the notion of a *policy similarity measure* between policies and is a quick but less precise technique based on principles from the information retrieval field [9]. Second technique is a *policy similarity analyzer* that is precise but computationally intensive and combines the capabilities of model-checking and SAT-solving techniques to provide a precise characterization of the policy similarities.

The rest of this chapter is organized as follows. Section 1 and Section 2 present in the detail the techniques and experimental results for the *policy similarity filter* and *policy similarity analyzer* respectively.

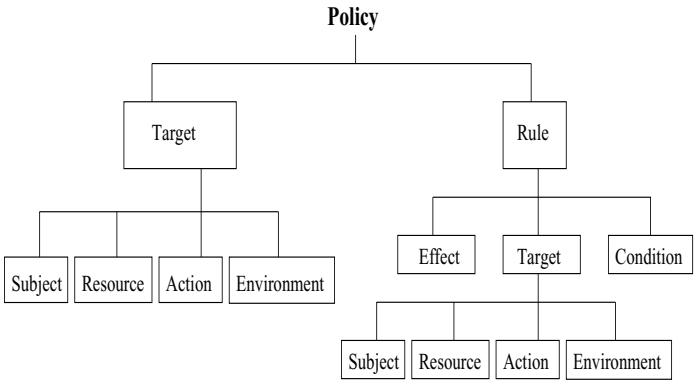


Figure 3.1. Policy Structure

3.1 Policy Similarity Filter

The policy similarity filter technique uses the notion of *policy similarity measure*, based on which a *similarity score*, a value in the interval $[0, 1]$, is computed between two policies to quantify the degree of similarity between them. Specifically, if the similarity score between policies P_1 and P_2 is higher than that between policies P_1 and P_3, it means P_1 and P_2 may yield same decisions to a larger common request set than P_1 and P_3 will do. The

similarity computation is simple and quick and is based on techniques from the information retrieval field.

The similarity measure takes into account the policy structure typical of XACML shown in figure 3.1. Given two polices, our algorithm for computing the similarity score first groups the same components of the two policies and evaluates their similarity. Then the scores obtained for the different elements of the policies are combined according to a weighted (usually linear) combination in order to produce an overall similarity score. Weights are used so that one can emphasize scores obtained due to one or more specific elements when computing the similarity. For example, one may want to find policies most similar with respect to the subjects to which they apply to. Techniques like dictionary lookup and ontology matching are also incorporated when computing the similarity scores to bridge the semantic gap arising due to the use of different vocabularies in expressing access control policies. Such techniques are particularly useful when harmonizing policies across administrative domains where each domain may use a different vocabulary for names and values appearing in their policies. As our case study shows, our approach can successfully identify similar policies.

3.1.1 An Illustrative Example

As an example that we will use throughout this section, we consider three policies P_1, P_2 and P_3, in the context of data and resource management for a grid computing system in a university domain. In particular, P_1 is a data owner policy, whereas P_2 and P_3 are resource owner policies. Specifically, P_1 states that professors, postdocs, students and technical staff in the IBM project group are allowed to read or write source, documentation or executable files of size less than 100MB. P_1 denies the write operations for postdocs, students and technical staff between 19:00 and 21:00 because professors may want to check and make changes to the project files without any distraction. P_2 is an access control policy for a project machine. P_2 allows students, faculty and technical staff in the IBM or Intel project group to read or write files of size less than 120MB. P_2 gives a special

PolicyId=P1
<PolicyTarget>
<Subject GroupName=IBMOpenCollaboration>
</PolicyTarget>
<RuleId=R11 Effect=Permit>
<Target>
<Subject Designation $belong\_to$ {Professor, PostDoc, Student, TechnicalStaff} >
<Resource FileType $belong\_to${Source, Documentation, Executable} >
<Action AccessType $belong\_to${Read, Write} >
</Target>
<Condition FileSize $\leq$ 100MB >
</Rule>
<RuleId=R12 Effect=Deny>
<Target>
<Subject Designation $belong\_to$ {Student, PostDoc, TechnicalStaff} >
<Resource FileType $belong\_to${Source, Documentation, Executable}>
<Action AccessType=Write>
</Target>
<Condition $19:00 \leq$ Time $\leq 21:00$>
</Rule>

Figure 3.2. Data Owner Policy P_1

permission to technical staff between time 19:00 and 22:00 so that technical staff can carry out system maintenance and backup files, and denies students the permission to write any file when technical staff is possibly working on maintenance. Moreover, P_2 does not allow any user to operate on media files on the machine. P_3 is an access control policy for another machine, mainly used by business staff. P_3 states that only business staff in the group named "Payroll" can read or write .xls files of size less than 10MB from 8:00 to 17:00, and it clearly denies students the access to the machine. Figure 3.2, 3.3 and 3.4 report the XACML specification for these policies.

From a user's perspective, P_1 is more similar to P_2 than P_3 because most activities described by P_1 for the data owner are allowed by P_2. Our motivation is to quickly compute similarity scores S_1 between P_1 and P_2, and S_2 between P_1 and P_3, where we would expect that S_1 be larger than S_2 to indicate that the similarity between P_1 and P_2 is much higher than the similarity between P_1 and P_3.

PolicyId=P2
<PolicyTarget>
<Subject GroupName $belong\_to$ {IBMOpenCollaboration, IntelOpenCollaboration}>
</PolicyTarget>
<RuleId=R21 Effect=Permit>
<Target>
<Subject Designation $belong\_to$ {Student, Faculty, TechnicalStaff} >
<Action AccessType $belong\_to${Read, Write}>
</Target>
<Condition FileSize $\leq$ 120MB >
</Rule>
<RuleId=R22 Effect=Permit>
<Target>
<Subject Designation=TechnicalStaff>
<Action AccessType $belong\_to${Read, Write}>
</Target>
<Condition $19:00 \leq$ Time $\leq 22:00$ >
</Rule>
<RuleId=R23 Effect=Deny>
<Target>
<Subject Designation=Student>
<Action AccessType=Write>
</Target>
<Condition {$19:00 \leq$ Time $\leq 22:00$>
</Rule>
<RuleId=R24 Effect=Deny>
<Target>
<Subject Designation $belong\_to$ {Student, Faculty, Staff}>
<Resource FileType=Media>
<Action AccessType $belong\_to$ {Read, Write}>
</Target>
</Rule>

Figure 3.3. Resource Owner Policy P_2

3.1.2 Policy Similarity Measure

Our proposed policy similarity measure is based on the comparison of each corresponding component of the policies being compared. Here, the corresponding component means the policy targets and the same type of elements belonging to the rules with the same effect.

We use a simplified XACML format for defining the policy similarity measure. Each XACML policy must be converted to this format when calculating the similarity score. Figure 3.5 gives the syntax of the simplified format.

PolicyId=P3
<PolicyTarget>
<Subject GroupName = Payroll >
</PolicyTarget>
<RuleId=R31 Effect=Permit>
<Target>
<Subject Designation=BusinessStaff>
<Resource FileType=".xls">
<Action AccessType $belong\_to$ {Read, Write}>
</Target>
<Condition $8:00 \leq$ Time $\leq 17:00$ AND FileSize $\leq$ 10MB >
</Rule>
<RuleId=R32 Effect=Deny>
<Target>
<Subject Designation=Student>
<Action AccessType $belong\_to$ {Read, Write}>
</Target>
</Rule>

Figure 3.4. Resource Owner Policy P_3

We would like the policy similarity measure between any two given policies to assign a *similarity score* S_{policy} that **approximates** the percentage of the requests obtaining the same decisions (permitted or denied) from the two policies. The definition is given in Equation 3.1, where N_{sreq} denotes the number of the requests with the same decisions from P_1 and P_2 and N_{req} is the number of the requests applicable to either P_1 or P_2.

$$S_{policy}(P_1, P_2) \approx N_{sreq}/N_{req} \tag{3.1}$$

The similarity score is a value between 0 and 1. Two equivalent policies are expected to obtain the similarity score close to 1. In a scenario where a set of requests permitted (denied) by a policy P_1 is a subset of requests permitted (denied) by a policy P_2, the similarity score for policies P_1 and P_2 must be higher than the score assigned in a scenario in which the set of requests permitted (denied) by P_1 and P_3 have very few or no request in common.

POLICY :
<policy policy-id = "*policy-id*" combining-algorithm = "*combining-algorithm*">
(TARGET ELEMENT)?
<permitrules>
(RULE ELEMENT)*
</permitrules>
<denyrules>
(RULE ELEMENT)*
</permitrules>
</policy>

RULE ELEMENT :
<rule rule-id="*rule-id*" effect="*rule-effect*">
(TARGET ELEMENT)?
<condition>PREDICATE</condition>
</rule>

TARGET ELEMENT :
<target>
<subject>PREDICATE</subject>
<resource>PREDICATE </resource>
<action>PREDICATE </action>
<environment>PREDICATE </environment>
</target>

PREDICATE :
(*attr_name* $\oplus$ (*attr_value*)+)*

attr_name denotes attribute name, *attr_value* denotes attribute value
and $\oplus$ denotes any operator supported by the XACML standard.

Figure 3.5. Simplified XACML format

3.1.2.1 Policy Normalization

Before calculating the similarity scores, a *policy normalization* process will be executed, which aims to capture more equivalent policies. Policy normalization consists of two operations: (i) upgrading rule targets; and (ii) decomposing condition component. The first operation extracts the common part of rule targets in all rules and treat it as the policy target during the similarity score calculation. The second operation is applied only to policies with the permit-override or deny-override combining algorithm. In particular, consider a rule R with a condition (c_1 OR c_2 OR ... OR c_k), where c_i ($1 \leq i \leq k$) can be any arbi-

trary Boolean expression. The policy normalization process will replace rule R with k new rules, each corresponding to a c_i. Note that, both operations will not affect the semantic meaning of the policy.

3.1.2.2 Overview of Policy Similarity Measure

We now introduce how to obtain the similarity score of two policies. Given two policies P_1 and P_2, the rules in these policies are first grouped according to their effects, which results in a set of Permit Rules (denoted as PR) and a set Deny Rules (denoted as DR). Note that we do not change the rule ordering. The grouping is just done logically. Each single rule in P_1 is then compared with a rule in P_2 that has the same effect, and a similarity score of two rules is obtained. The similarity score obtained between the rules is then used to find one-many mappings (denoted as Φ) for each rule in the two policies. For clarity, we choose to use four separate Φ mappings Φ_1^P, Φ_1^D, Φ_2^P and Φ_2^D. The mapping $\Phi_1^P(\Phi_1^D)$ maps each PR(DR) rule r_{1i} in P_1 with one or more PR(DR) rules r_{2j} in P_2. Similarly the mapping $\Phi_2^P(\Phi_2^D)$ maps each PR(DR) rule r_{2j} in P_2 with one or more PR(DR) rules r_{1i} in P_1. For each rule in a policy $P_1(P_2)$, the Φ mappings give similar rules in $P_2(P_1)$ which satisfy certain similarity threshold. The computation of the Φ mapping will be addressed in the Section 3.1.2.3.

By using the Φ mappings, we compute the similarity score between a rule and a policy. We aim to find out how similar a rule is with respect to the entire policy by comparing the single rule in one policy with a set of similar rules in the other policy. The notation $rs_{1i}(rs_{2j})$ denotes the similarity score for a rule $r_{1i}(r_{2j})$ in policy $P_1(P_2)$. The rule similarity score $rs_{1i}(rs_{2j})$ is the average of the similarity scores between a rule $r_{1i}(r_{2j})$ and the rules similar to it given by the Φ mapping. rs_{1i} and rs_{2j} are computed according to Equations 3.2 and 3.3, where S_{rule} is a function that assigns a similarity score between two rules.

Next, we compute the similarity score between the permit(deny) rule sets PR_1 (DR_1) and $PR_2(DR_2)$ of policies P_1 and P_2 respectively. We use the notations $S^P_{rule-set}$ and $S^D_{rule-set}$ to denote the similarity scores for permit and deny rule sets respectively. The

similarity score for a permit(deny) rule set is obtained by averaging the rule similarity scores (Equations 3.2 and 3.3) for all rules in the set. The permit and deny rule set similarity scores are formulated by Equation 3.4 and 3.5, where N_{PR_1} and N_{PR_2} are the numbers of rules in PR_1 and PR_2 respectively, N_{DR_1} and N_{DR_2} are the numbers of rules in DR_1 and DR_2 respectively, and p_x reflects the importance of rule r_x. In particular, p_x is determined by the number of requests applicable to the rule r_x. It is set to 1 by default. It comes into effect when the statistical information about the request distribution is available. Suppose that policy P_1 contains k rules and the numbers of applicable requests for the rules are n_1,..., n_k respectively. Correspondingly, p_1, ..., p_k can be calculated as : $p_x = n_x/(n_1 + ...+n_k)$.

$$rs_{1i} = \begin{cases} \dfrac{\sum\limits_{r_j \in \Phi_1^P(r_{1i})} S_{rule}(r_{1i}, r_j)}{|\Phi_1^P(r_{1i})|}, r_{1i} \in PR_1 \\ \dfrac{\sum\limits_{r_j \in \Phi_1^D(r_{1i})} S_{rule}(r_{1i}, r_j)}{|\Phi_1^D(r_{1i})|}, r_{1i} \in DR_1 \end{cases} \tag{3.2}$$

$$rs_{2j} = \begin{cases} \dfrac{\sum\limits_{r_i \in \Phi_2^P(r_{2j})} S_{rule}(r_{2j}, r_i)}{|\Phi_2^P(r_{2j})|}, r_{2j} \in PR_2 \\ \dfrac{\sum\limits_{r_i \in \Phi_2^D(r_{2j})} S_{rule}(r_{2j}, r_i)}{|\Phi_2^D(r_{2j})|}, r_{2j} \in DR_2 \end{cases} \tag{3.3}$$

$$S^P_{rule-set} = \frac{\sum\limits_{i=1}^{N_{PR_1}} (rs_{1i} \cdot p_{1i}) + \sum\limits_{i=1}^{N_{PR_2}} (rs_{2j} \cdot p_{2j})}{N_{PR_1} + N_{PR_2}} \tag{3.4}$$

$$S^D_{rule-set} = \frac{\sum\limits_{i=1}^{N_{DR_1}} (rs_{1i} \cdot p_{1i}) + \sum\limits_{i=1}^{N_{DR_2}} (rs_{2j} \cdot p_{2j})}{N_{DR_1} + N_{DR_2}} \tag{3.5}$$

Finally, we combine the similarity scores for permit and deny rule sets between the two policies along with a similarity score between the *Target* elements of the two policies,

to develop an overall similarity score, S_{policy}. The formulation of S_{policy} is given by the following equation:

$$S_{policy}(P_1, P_2) = w_T S_T(P_1, P_2) + w_p S^P_{rule-set} + w_d S^D_{rule-set} \tag{3.6}$$

where S_T is a function that computes a similarity score between the *Target* elements of any two given policies; w_T, w_p and w_d are weights that can be chosen to reflect the relative importance to be given to the similarity of the policy target, permit and deny rule sets respectively. For normalization purpose, the weight values should satisfy the constraint: $w_T + w_p + w_d = 1$. The weights introduced here and in the later part aims to provide more flexibility for users to locate desired policies. For example, if a user would like to find policies applicable to similar targets regardless of policy decision, he can assign a larger value to w_T to emphasize on the target similarity. Without any preference, equal weight values are assumed by default. An example is given in Section 3.1.2.12 (refer to step 5–9).

The intuition behind the similarity score assigned to any two policies is derived from the fact that two policies are similar to one another when the corresponding policy elements are similar.

In the following sections, we introduce the detailed algorithms for the computation of Φ mappings and rule similarity score S_{rule}. Table 3.1 lists main notations used in this section.

3.1.2.3 Computation of Φ Mappings

The one-many Φ mappings determine for each PR(DR) rule in $P_1(P_2)$ which PR(DR) rules in $P_2(P_1)$ are *very similar*. Intuitively, two rules are *similar* when their targets and the conditions they specify are similar. Thus we define a *general* Φ *mapping* as follows:

$$\Phi(r_i) = \{r_j | S_{rule}(r_i, r_j) \geq \epsilon\} \tag{3.7}$$

where S_{rule} is computed by Equation 3.8 and ϵ is a threshold. The threshold term allows us to calibrate the quality of the similarity approximation.

Table 3.1
Notations

Notation	Meaning
P	Policy
PR	Permit rule set
DR	Deny rule set
r	Rule
a	Attribute
v	Attribute value
H	Height of a hierarchy
S_{policy}	Similarity score of two policies
S_{rule}	Similarity score of two rules
$S^{P}_{rule-set}$	Similarity score of two sets of permit rules
$S^{D}_{rule-set}$	Similarity score of two sets of deny rules
$S_{\langle Element\rangle}$	Similarity score of elements, $\langle Element\rangle \in \{'T','t','c','s','r','a','e'\}$
s_{cat}	Similarity score of two categorical values
S_{cat}	Similarity score of two categorical predicates
s_{num}	Similarity score of two numerical values
S_{num}	Similarity score of two numerical predicates
rs	Similarity score between a rule and a policy
Φ	Rule mapping
$\mathcal{M}_a$	Set of pairs of matching attribute names
$\mathcal{M}_v$	Set of pairs of matching attribute values
N_{PR}	Number of permit rules in a policy
N_{DR}	Number of deny rules in a policy
N_a	Number of attributes in an element
N_v	Number of values of an attribute
$SPath$	Length of shortest path of two categorical values
$w_{\langle Element\rangle}$	Weight of similarity scores of elements, $\langle Element\rangle \in \{'T','t','c','s','r','a','e'\}$
ϵ	Rule similarity threshold
δ	Compensating score for unmatched values
p_r	Percentage of requests applicable to rule r

An example of the computation of a Φ mapping is shown by steps 3 and 4 in Section 3.1.2.12.

The general Φ mapping is further refined to a one-one mapping when the first-one-applicable or only-one-applicable rule combining algorithm is employed. Without loss of generality, we consider the calculation of the mappings for rules in P_1. Let r_i and r_j denote the rule with the same effect in P_1 and P_2 respectively. If P_2 has the first-one-applicable

Procedure ComputePhiMapping(R'**,** R''**,** ϵ**)**
Input : R' and R'' are sets of rules and ϵ
is a *threshold* value.
1. **foreach** rule $r' \in R'$
2. $\Phi(r') = \emptyset$
3. **foreach** rule $r'' \in R''$
4. **if** $S_{rule}(r', r'') \geq \epsilon$ **then**
5. $\Phi(r') = \Phi(\ r') \cup \{r''\}$
6. **return** Φ
end ComputePhiMapping.

Figure 3.6. Procedure for computing a Φ mapping

rule combining algorithm, the Φ mapping for r_i will contain the only r_j which is the first rule with the similarity score $S_{rule}(r_i, r_j)$ above the threshold ϵ. This method first ensures that the two rules are applicable to the similar set of requests with the aid of the threshold, and then reflects the definition of the first-one-applicable rule combining algorithm which specifies that the first applicable rule makes the final decision. We can deal with the P_2 with the only-one-applicable rule combining algorithm in a similar way. The Φ mapping for r_i will contain the only r_j which has the maximum similarity score and the score should also be above the threshold.

Figure 3.6 summarizes the procedure for calculating a Φ mapping. This procedure takes two rule sets R' and R'' as input and computes a mapping for each rule in R' based on Equation 3.7.

3.1.2.4 Similarity Score between Rules

Since our similarity measure is targeted as a lightweight approach, we do not want to involve complicated analysis of Boolean expressions. Our similarity measure is developed based on the intuition that rules r_i and r_j are similar when both apply to similar targets

and both specify similar conditions on request attributes. Specifically, we compute the rule similarity function S_{rule} between two rules r_i and r_j as follows:

$$S_{rule}(r_i, r_j) = w_t S_t(r_i, r_j) + w_c S_c(r_i, r_j) \tag{3.8}$$

w_t and w_c are weights that can be used for emphasizing the importance of the rule target or condition similarity respectively. For example, if users are more interested in finding policies applied to similar targets, they can increase w_t to achieve this goal. The weights satisfy the constraint $w_t + w_c = 1$. S_t and S_c are functions that compute a similarity score between two rules based on the comparison of their *Target* and *Condition* elements respectively.

As the *Target* element in each rule contains the *Subject*, *Resource* and *Action* elements, each of these elements in turn contains predicates on the respective category of attributes. Thus, the *Target* similarity function S_t is computed as follows:

$$S_t(r_i, r_j) = w_s S_s(r_i, r_j) + w_r S_r(r_i, r_j) + w_a S_a(r_i, r_j) + w_e S_e(r_i, r_j) \tag{3.9}$$

In Equation 3.9, w_s, w_r, w_a, w_e represent weights that are assigned to the corresponding similarity scores. Like in the previous equations, weight values need to satisfy the constraint $w_s + w_r + w_a + w_e = 1$. S_s, S_r, S_a and $S_e$are functions that return a similarity score based on the *Subject*, *Resource*, *Action* and *Environment* attribute predicates respectively in the *Target* elements of the two given rules.

The computation of functions S_c, S_s, S_r, S_a and S_e involves the comparison of pairs of predicates in the given pair of rule elements, which we discuss in detail in the next subsection.

3.1.2.5 Similarity Score of Rule Elements

Each of the rule elements *Subject*, *Resource*, *Action*, *Environment* and *Condition* is represented as a set of predicates in the form of $\{attr\_name_1 \oplus_1 attr\_value_1, attr\_name_2$

$\oplus_2 attr\_value_2, ...\}$, where $attr\_name$ denotes the attribute name, $\oplus$ denotes a comparison operator and $attr\_value$ represents an attribute value.

We support syntactic and semantic variations, with respect to attribute names and values, that can occur in different policies. An example of syntactic variation is the use of "emp-name" and "EmpName" to refer to the employee name attribute. An example of semantic variation is a case in which the synonym "pay" is used to refer to an employee's "salary" attribute. Syntactic variations are addressed by using a user-defined lookup table typically set up by a policy administrator. For detecting semantic variations we use the WordNet [15], a lexical database for English language, which is used to derive all the synonyms in the context of English language for an attribute name. Semantic variations can also occur when attribute names or values are associated with different domain ontologies. For such cases we use a semantic score obtained from running an *ontology matching* algorithm [16] on the specified ontologies.

Based on the type of attribute values, predicates are divided into two categories, namely *categorical predicate* and *numerical predicate*.

- *Categorical predicate*: The attribute values of this type of predicate belong to the *string* data type. Such values may or may not be associated with a domain-specific ontology. They may also be associated with more than one ontology. Predicates like "Designation = Professor" and "FileType = Documentation" belong to the categorical type.

- *Numerical predicate*: The attribute values of this type of predicate belong to *integer*, *real*, or *date/time* data types. For example, predicates "FileSize $<$ 10MB", "Time=12:00" are of numerical type.

The similarity score between two rules r_i and r_j regarding the same element is denoted as $S_{\langle Element \rangle}$, where $\langle Element \rangle$ refers to condition, subject, resource or action. The $S_{\langle Element \rangle}$ is computed by comparing the corresponding predicate sets in two rules. There are three steps. First, we cluster the predicates for each rule element according to the attribute names. It is worth noting that one attribute name may be associated with multiple

values. Second, we find the predicates in the two rules whose attribute names match exactly and then proceed to compute a similarity score for their attribute values. The way we compute similarity score between attribute values differs, depending on whether the attribute value is of categorical type or numerical type (details about the computation are covered in the following subsection). Finally, we summarize the scores of each pair of matching predicates and obtain the similarity score of the rule element. Since not all attributes in one rule can find a matching in the other, we include a penalty for this case by dividing the sum of similarity scores of matching pairs by the maximum number of attributes in a rule. In addition, there is a special case when the element set is empty in one rule, which means no constraint exists for this element. For this case, we consider the similarity of the elements of the two rules to be 0.5 due to the consideration that one rule is a restriction of the other and the 0.5 is the estimation of the average similarity. The formal definition of $S_{\langle Element \rangle}$ is given by Equation 3.10.

$$S_{\langle Element \rangle}(r_i, r_j) = \begin{cases} \dfrac{\sum_{(a_{1k}, a_{2l}) \in \mathcal{M}_a} S_{\langle attr\_typ \rangle}(a_{1k}, a_{2l})}{max(N_{a_1}, N_{a_2})}, & N_{a_1} > 0 \text{ and } N_{a_2} > 0; \\ 1, & \text{otherwise.} \end{cases} \tag{3.10}$$

In Equation 3.10, $\mathcal{M}_a$ is a set of pairs of matching predicates with the same attribute names; a_{1k} and a_{2l} are attributes of rules r_{1i} and r_{2j} respectively; $S_{\langle attr\_typ \rangle}$ is the similarity score of attribute values of the type $attr\_typ$; and N_{a_1} and N_{a_2} are the numbers of distinct predicates in the two rules respectively.

In addition, the computation of the similarity score of two policy targets S_T is the same as that for the rule targets i.e. S_t.

3.1.2.6 Similarity Score for Categorical Predicates

For the categorical values, we not only consider the exact match of two values, but also consider their semantic similarity. For example, policy P_1 is talking about the priority

of professors, policy P_2 is talking about faculty members, and policy P_3 is talking about business staff. In some sense, policy P_1 is more similar to policy P_2 than to policy P_3 because "professors" is a subset of "faculty members" which means that policy P_1 could be a restriction of policy P_2. Based on this observation, our approach assumes that a hierarchy relationship exists for the categorical values. The similarity between two categorical values (denoted as S_{cat}) is then defined according to the shortest path of these two values in the hierarchy. The formal definition is shown below:

$$s_{cat}(v_1, v_2) = 1 - \frac{SPath(v_1, v_2)}{2H} \tag{3.11}$$

where $SPath(v_1, v_2)$ denotes the length of the shortest path between two values v_1 and v_2, and H is the height of the hierarchy. In Equation 3.11, the length of the shortest path of two values is normalized by the possible maximum path length which is $2H$. The closer the two values are located in the hierarchy, the more similar the two values will be, and hence a higher similarity score s_{cat} will be obtained.

Figure 3.7 gives an example hierarchy, where each node represents a categorical value (specific values are given in Figure 3.12). The height of the hierarchy is 3, and the length

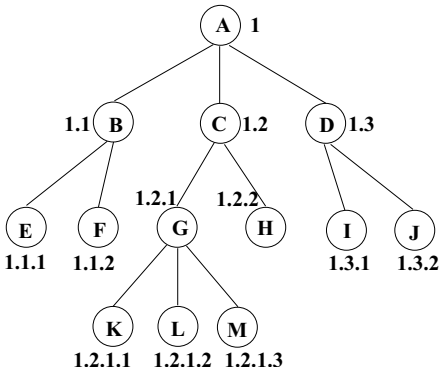


Figure 3.7. An Example Hierarchy

of maximum path of two values is estimated as $2 \times 3 = 6$ (the actual maximum path in the figure is 5 due to the imbalance of the hierarchy). $SPath(E, B)$ is 1 and $SPath(E, F)$ is 2. According to Equation 3.11, the similarity score of nodes E and B is $1 - 1/6 = 0.83$,

and the similarity score of nodes E and F is $1 - 2/6 = 0.67$. From the obtained scores, we can observe that E is more similar to B than to F. The underlying idea is that the parent-child relationship (B and E) implies that one rule could be a restriction of the other and this would be more helpful than the sibling relationship (E and F) especially in rule integration.

To avoid repeatedly searching the hierarchy tree for the same value during the shortest path computation, we assign to each node a *hierarchy code* (Hcode), indicating the position of each node. In particular, the root node is assigned an Hcode equal to '1', and its children nodes are named in the order from left to right by appending their position to the parent's Hcode with a separator '.', where we will have Hcodes like '1.1' and '1.2'. Then the process continues till the leaf level. The number of elements separated by '.' is equal to the level at which a node is located. From such Hcodes we can easily compute the length of shortest path between two nodes. We compare two Hcodes element by element until we reach the end of one Hcode or there is a difference. The common elements correspond to the same parent nodes they share, and the number of different elements correspond to the levels that they need to be generalized to their common parent node. Therefore, the shortest path is the total number of different elements in two Hcodes. For example, the length of the shortest path from node '1.1' to '1.2' is 2, as there are two different elements in the Hcodes.

Note that our definition of s_{cat} can also be applied to categorical values which do not lie in a hierarchy. In that case, if two values are matched, their shortest path $SPath$ is 0 and their similarity score will be 1; otherwise, $SPath$ is infinity and their similarity score becomes 0.

Having introduced our approach to compare two single values, we now extend the discussion to two sets of values. Suppose there are two attributes $a_1 : \{v_{11}, v_{12}, v_{13}, v_{14}\}$ and $a_2 : \{v_{21}, v_{22}, v_{23}\}$, where a_1 and a_2 are the attribute names belonging to policy P_1 and P_2 respectively, and the values in the brackets are corresponding attribute values. Note that the values associated with the same attribute are different from one another. The similarity score of the two attribute value sets is the sum of similarity scores of pairs $\langle v_{1k}, v_{2l} \rangle$ and a compensating score δ (for non-matching attribute values). Obviously, there could be many

combinations of pairs. Our task is to find a set of pairs (denoted as $\mathcal{M}_v$) which have the following properties:

1. If $v_{1k} = v_{2l}$, then $(v_{1k}, v_{2l}) \in \mathcal{M}_v$.
2. For pairs $v_{1k} = v_{2l}$, pairs contributing to the maximum sum of similarity scores belong to $\mathcal{M}_v$.
3. Each attribute value v_{1k} or v_{2l} occurs at most once in $\mathcal{M}_v$.

The process of finding the pair set $\mathcal{M}_v$ is the following. First, we obtain the hierarchy code for each attribute value. See Figure 3.8 for an example of these values for the example hierarchy from Figure 3.7. Then we compute the similarity between pairs of attribute values with the help of the hierarchy code. Figure 3.9 shows the resulting scores for the example. Next, we pick up exactly matched pairs, which are $\langle v_{11},\ v_{21}\rangle$ and $\langle v_{14},\ v_{23}\rangle$ in the example. For the remaining attribute values, we find pairs that maximize the sum of similarity scores of pairs. In this example, $\langle v_{12},\ v_{22}\rangle$ has the same similarity score as $\langle v_{13},\ v_{22}\rangle$, and hence we need to further consider which choice can lead to a bigger compensating score. The compensating score δ is for attribute values which do not have matchings when two attributes have different number of values. δ is computed as average similarity scores between unmatched values with all the values of the other attribute. For this example, no matter which pair we choose, the compensating score is the same. Suppose we choose the pair $\langle v_{12},\ v_{22}\rangle$, and then one value v_{13} is left whose compensating score δ is $(0.33 + 0.67 + 0.17)/3 = 0.39$. Finally, the similarity score for the two attribute a_1 and a_2

Policy P_1

Attr	*Hcode*
v_{11}	1.1
v_{12}	1.2.1.1
v_{13}	1.2.1.2
v_{14}	1.3.2

Policy P_2

Attr	*Hcode*
v_{21}	1.1
v_{22}	1.2
v_{23}	1.3.2

Figure 3.8. Hierarchy Code

P_2 \ P_1	v_{11} (1.1)	v_{12} (1.2.1.1)	v_{13} (1.2.1.2)	v_{14} (1.3.2)
v_{21} (1.1)	*1*	*0.33*	*0.33*	*0.5*
v_{22} (1.2)	*0.67*	***0.67***	*0.67*	*0.5*
v_{23} (1.3.2)	*0.5*	*0.17*	*0.17*	*1*

Figure 3.9. Similarity Score of Two Sets of Attributes

takes into account both the similarity of attribute names and attribute values. Specifically, the similarity score for attribute names is 1 as the exact matching of names is used. The similarity score for attribute values is the average scores of pairs and the compensating score. The final score is $\frac{1}{2}[1 + (1 + 1 + 0.67 + 0.39)/4] = 0.88$.

The similarity score of two categorical predicates is finally defined as below:

$$S_{cat}(a_1, a_2) = \frac{1}{2}\Big[1 + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} s_{cat}(v_{1k}, v_{2l}) + \delta}{max(N_{v_1}, N_{v_2})}\Big] \tag{3.12}$$

$$\delta = \begin{cases} \dfrac{\sum\limits_{(v_{1k}, \_) \notin \mathcal{M}_v} \sum\limits_{l=1}^{N_{v_2}} s_{cat}(v_{1k}, v_{2l})}{N_{v_2}}, & N_{v_1} > N_{v_2}; \\ \dfrac{\sum\limits_{(\_, v_{2l}) \notin \mathcal{M}_v} \sum\limits_{k=1}^{N_{v1}} s_{cat}(v_{1k}, v_{2l})}{N_{v_1}}, & N_{v_2} > N_{v_1}. \end{cases} \tag{3.13}$$

where N_{v_1} and N_{v_2} are the total numbers of values associated with attributes a_1 and a_2 respectively.

3.1.2.7 Similarity Score for Numerical Predicates

Unlike categorical values, numerical values do not have any hierarchical relationship. For computation efficiency, the similarity of two numerical values v_1 and v_2 is defined based on their difference as shown in Equation 3.14.

$$s_{num}(v_1, v_2) = 1 - \frac{|v_1 - v_2|}{range(v_1, v_2)} \tag{3.14}$$

s_{num} tends to be large when the difference between two values is small.

The computation of the similarity score of two numerical value sets is similar to that for two categorical value sets; we thus have the following similarity definition for numerical predicates:

$$S_{num}(a_1, a_2) = \frac{1}{2}\ 1 + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} s_{num}(v_{1k}, v_{2l}) + \delta}{max(N_{v1}, N_{v2})} \tag{3.15}$$

$$\delta = \begin{cases} \dfrac{\sum\limits_{(v_{1k}, \_) \notin \mathcal{M}_v} \sum\limits_{l=1}^{N_{v_2}} s_{num}(v_{1k}, v_{2l})}{N_{v_2}}, & N_{v_1} > N_{v_2}; \\ \dfrac{\sum\limits_{(\_, v_{2l}) \notin \mathcal{M}_v} \sum\limits_{k=1}^{N_{v1}} s_{num}(v_{1k}, v_{2l})}{N_{v_1}}, & N_{v_2} > N_{v_1}. \end{cases} \tag{3.16}$$

3.1.2.8 Similarity Score of Rule Elements - A Variation

As aforementioned, we do not want to complicate the policy comparison by introducing Boolean expression analysis, and hence we ignore operators (denoted by $\oplus$) in the predicates. This would mean that two predicates like *salary* $<$ *10000* and *salary* $>$ *10000* would be given a similarity score of 1. In order to distinguish these two cases to some extent while still keeping our approach a lightweight approach, we propose the following variation for computing rule element similarity scores.

The basic idea is to first cluster the predicates for each rule element according to the attribute names and then further cluster values of the same attributes according to the operators. After that, we use the same procedure as described in Section 3.1.2.5 to compute the similarity scores between clusters with matching attribute names. The new similarity score between rule elements is computed by Equation 3.17.

$$S^{op}_{\langle Element \rangle}(r_i, r_j) = \begin{cases} \dfrac{\sum\limits_{(a_{1k}, a_{2l}) \in \mathcal{M}_a} S^{op}_{\langle attr\_typ \rangle}(a_{1k}, a_{2l})}{max(N_{a_1}, N_{a_2})}, & N_{a_1} > 0 \text{ and } N_{a_2} > 0; \\ 1, & \text{otherwise.} \end{cases} \tag{3.17}$$

Note that Equation 3.17 now uses a new similarity score given by $S^{op}_{\langle attr\_typ \rangle}$ which is defined in Equation 3.18.

$$S^{op}_{\langle attr\_typ \rangle}(a_1, a_2) = \frac{1}{2}\ 1 + \frac{\sum_{(a_{1k}, a_{2l}) \in \mathcal{M}_{op}} S_{\langle attr\_typ \rangle}(a_{1k}, a_{2l})}{max(N_{op_1}, N_{op_2})} \tag{3.18}$$

where N_{op_1} and N_{op_2} are the number of unique operators in the predicates corresponding to attributes a_1 and a_2 respectively. $\mathcal{M}_{op}$ is a set of pairs of predicates with the same operator and attribute name. Depending on the types of the attributes being compared $S_{\langle attr\_typ \rangle}$ in Equation 3.18 is substituted with S_{cat} or S_{num} which can be calculated using equations 3.12 and 3.15 respectively.

3.1.2.9 Selectivity

The operator-based variant of policy similarity measure can be further extended to incorporate the notion of *predicate selectivity*. The concept of predicate selectivity is often used in relational database management systems for query cost estimation. Typically, selectivity of a predicate p refers to the percentage of records in a relation that satisfy p. Extending this concept to the current context,we can regard selectivity as a measure of the fraction of requests that satisfy predicates in a policy rule. Selectivity is very useful in improving the accuracy of similarity scores n some cases. Below is a simple example which contains three policies P_1, P_2 and P_3 with the permit effect and the same target component. They differ only in their condition components as shown below:

P_1: (sex=female) $\wedge$ (salary = 5000)

P_2: (sex=female $\vee$ sex=male) $\wedge$ (salary = 5000)

P_3: salary = 5000

We can observe that P_2 and P_3 are much more similar than P_2 and P_1 in terms of number of requests that can be permitted. However, when using previous introduced similarity measure, we may obtain a lower $S_{policy}(P_2, P_3)$ than $S_{policy}(P_1, P_2)$ because P_1 and P_2 have more common attributes. Such problem can be relieved if selectivity is considered and

carefully integrated into similarity score calculation, as we can see that the selectivity of the conditions in P_2 and P_3 is higher than that of the condition in P_1.

We proceed to formally define the selectivity used in our similarity measure. Selectivity of a predicate $p_i : a_i \oplus v_i$ is computed as follows:

$$Sel_{predicate}(p_i) = \frac{\text{\# of values in domain}(a_i) \text{ that satisfy } p_i}{\text{total \# of distinct values in domain}(a_i)} \tag{3.19}$$

For example, selectivity of the predicate "sex $=$ female" is $1/2$ considering that the domain of the *sex* attribute has two distinct values {male, female}. Using the independence assumption, selectivity of a conjunctive predicate of the form $p_i \wedge p_j$ can be computed as follows:

$$Sel_{predicate}(p_i \wedge p_j) = Sel_{predicate}(p_i) \wedge Sel_{predicate}(p_j) \tag{3.20}$$

Similarly, the selectivity of a disjunctive predicate $p_i \vee p_j$ can be computed using the formula:

$$Sel_{predicate}(p_i \vee p_j) = Sel_{predicate}(p_i) + Sel_{predicate}(p_j) - Sel_{predicate}(p_i)Sel_{predicate}(p_j) \tag{3.21}$$

The basic idea here is to adjust the policy similarity score obtained for a policy pair, $S_{policy}(P_1, P_2)$, based on the selectivities of the policy target and rule elements of P_1 and P_2. In order to do this, we compute a *policy target selectivity*, denoted by $Sel_{T1}(Sel_{T2})$, and a *rule selectivity*, denoted by $Sel_{r1i}(Sel_{r2j})$ for each policy target and rule respectively in P_1(P_2), where $1 \leq i \leq (N_{PR_1} + N_{DR_1})(1 \leq j \leq (N_{PR_2} + N_{DR_2}))$.

The *policy target selectivity* is derived by combining the selectivities of the elements within the target. The target is essentially a conjunction of the subject, action, resource and environment element predicates. Thus, the policy target selectivity Sel_T can be computed as follows:

$$Sel_T = Sel_{\langle s \rangle} Sel_{\langle a \rangle} Sel_{\langle r \rangle} Sel_{\langle e \rangle} \tag{3.22}$$

where $Sel_{\langle s \rangle}, Sel_{\langle r \rangle}, Sel_{\langle a \rangle}$ and $Sel_{\langle e \rangle}$ denote the *subject* , *action*, *resource* and *environment* element selectivities respectively. Each of these element selectivities is in turn computed based on the selectivities of the corresponding element predicates. Each of these elements is essentially a *disjunctive normal form* of predicates ($(p_1 \wedge p_2 \wedge \cdots p_i) \vee (p_{i+1} \wedge \cdots \wedge p_j) \vee \cdots \vee (p_k \wedge \ cdots \wedge p_n)$) [1] and thus the element selectivity can computed using equations 3.20 and 3.21. The policy target selectivities Sel_{T1} and Sel_{T2} obtained for policies P_1 and P_2 respectively are used to derive a new policy target similarity score S'_T using the following steps:

1. The difference in selectivities, $SD_T = |Sel_{T1} - Sel_{T2}|$ is computed.

2. The SD_T(if greater than 0) is used to derive a *selectivity difference weight*, SW_T, the computation for which is given in equation 3.23.

$$SW_T = 1 + \log SD_T \tag{3.23}$$

3. If $0 \leq SW_T \leq 1$, the original policy target similarity score S_T is adjusted to derive the new policy target score S'_T using equation 3.24.

$$S'_T(P_1, P_2) = S_T(P_1, P_2)(1 - SW_T * \xi) \tag{3.24}$$

The intuition behind steps (1)-(3) is the following. Higher difference in selectivities SD_T implies that the policy elements are different and hence must be assigned lower similarity scores. However, lower selectivity difference does not guarantee that two policy elements are similar since it is possible that two different policy elements containing different sets of attributes have similarly high selectivity values. Therefore, we integrate only a partial effect of selectivity difference by using the parameter ξ which is set to 0.2 as default. In addition, since we use selectivity to mainly reduce the rate of false positives, we ignore any selectivity difference (SD) less than 0.1 and adjust the original similarity score only when SW_T lies between 0 and 1.

Similarly, the *rule selectivity* is derived by combining the *rule target* and *condition* selectivities as follows:

$$Sel_{ri} = Sel_{ti}Sel_{ci} \tag{3.25}$$

where Sel_{ti}, the rule target selectivity and Sel_{ci}, the condition selectivity are computed from the corresponding element predicates using equations 3.20 and 3.21. Further, the Sel_{r1i} and Sel_{r2j} computed for each rule in P_1 and P_2 respectively is used to compute a new rule similarity score $S'_{rule}(r_{1i}, r_{2j})$ for each pair of permit and deny rules in the two policies using computations similar to steps (1)-(3) described above. Finally, the new rule similarity scores are used to derive new permit and deny set similarity scores $S'^{P}_{rule-set}$ and $S'^{D}_{rule-set}$ as outlined in section 3.1.2.2.

3.1.2.10 Similarity Score with Dictionary Lookup and Ontology Matching

So far, our similarity measure is defined under an implicit assumption that every party uses the same vocabulary to write its policies. However, policies being compared for similarity may use different vocabularies and hence have *syntactic* and/or *semantic* variations of attribute names and categorical values. Therefore, we extend our policy similarity measure with dictionary lookup and ontology matching techniques to incorporate such variations..

An example of syntactic variation is the use of ”emp-name” and ”EmpName” to refer to the employee name attribute. For such cases, we use a user-defined lookup table typically set up by a policy administrator.

An example of semantic variation is a case in which the synonym “pay” is used to refer to an employee’s salary attribute. For such cases, we use the WordNet [15], a lexical database for English language, which is used to derive all the synonyms in the context of English language.

Semantic variations can also occur when attribute names or categorical values are associated with different ontologies. For such cases, we use a semantic score obtained from running an *ontology matching* algorithm [16] on the different ontologies. We will now dis-

cuss how to obtain such a semantic score between two attribute names or values belonging to different ontologies.

Let O_1 and O_2 be different ontologies to which the values v_1 and v_2 that are being compared belong to respectively. This means that v_1 and v_2 represent concepts (nodes) in the ontologies O_1 and O_2 respectively. An ontology matching algorithm A_O [16] takes two ontologies O_1 and O_2 as input and returns a mapping $M_{O_1 \rightarrow O_2}$ between the two ontologies. The mapping $M_{O_1 \rightarrow O_2}$ contains for each concept (node) C_i in the ontology $O_1(O_2)$ a matching concept C_j in $O_2(O_1)$ along with a *confidence measure* m, a value between 0 and 1, indicating the similarity between the matched concepts. Thus the matching $M_{O_1 \rightarrow O_2}$ is a list of triples of the form $\langle C_i, C_j, m \rangle$. Second, we incorporate the scores obtained from the ontology mapping $M_{O_1 \rightarrow O_2}$ to calculate the similarity score between two categorical values as follows :

1. Let C_{12} be the concept in O_2 that matches v_1 and let m_{12} be the corresponding matching score. Similarly let C_{21} be the concept in O_1 that matches v_2 and let m_{21} be the corresponding matching score. If no matching concept is found in either case a 0 score is returned.

2. We now have two pairs of values $P_1 : \{v_1, C_{21}\}$ and $P_2 : \{v_2, C_{12}\}$ that belong to the ontologies O_1 and O_2 respectively. We then apply the techniques presented in Section 3.1.2.6 and use Equation 3.11 to calculate the scores s_1 and s_2 for pairs P_1 and P_2 respectively. Note that the score s_1 (s_2) is calculated using Equation 3.11 only if the matching score $m_{21}(m_{12})$ is greater than a *matching threshold* t_m(a value between 0 and 1). Otherwise they are set to 0.

3. An average of the scores s_1 and s_2 is returned as the semantic score between the values v_1 and v_2.

Let s_{cat}^{ONTO} denote the function that computes the semantic score. Equation 3.26 summarizes the computation of the semantic score.

$$s_{cat}^{ONTO}(v_1, v_2) = \begin{cases} \frac{s_{cat}(v_1, C_{21}) + s_{cat}(v_2, C_{12})}{2}, C_{21}, C_{12} = \emptyset \text{ and } m_{21}, m_{12} \geq t_m \\ 0, \quad C_{21} = \emptyset \text{ or } C_{12} = \emptyset \\ \frac{s_{cat}(v_1, C_{21})}{2}, \quad m_{21} < t_m \\ \frac{s_{cat}(v_2, C_{12})}{2}, \quad m_{12} < t_m \end{cases} \tag{3.26}$$

We then revise Equations 3.12 and 3.15 for computing predicate similarity scores to Equations 3.27 and 3.28 respectively, by incorporating syntactic and semantic variations.

$$S_{cat}(a_1, a_2) = \frac{1}{2}\ s_{aname}(n_{a_1}, n_{a_2}) + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} s_{cat}^{ONTO}(v_{1k}, v_{2l}) + \delta}{max(N_{v_1}, N_{v_2})} \tag{3.27}$$

$$S_{num}(a_1, a_2) = \frac{1}{2}\ s_{aname}(n_{a_1}, n_{a_2}) + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} s_{num}(v_{1k}, v_{2l}) + \delta}{max(N_{v1}, N_{v2})} \tag{3.28}$$

where n_{a_1} and n_{a_2} denote the attribute names associated with a_1 and a_2 respectively and $s_{aname}(n_{a_1}, n_{a_2})$ is a function that returns 1 if n_{a_1} and n_{a_2} are syntactic variation or synonym of one another and returns a value equal to $S_{cat}^{ONTO}(n_{a_1}, n_{a_2})$ if n_{a_1} and n_{a_2} are associated with different ontologies and in all other cases returns a value 0.

Note that as a result of this change, we no longer only compare predicates whose attribute names match exactly. Instead, we compare all pairs of predicates.

3.1.2.11 Overall Algorithm

In this section, we summarize the steps involved in the computation of a similarity score between two policies P_1 and P_2. Figure 3.10 presents the pseudo-code of the complete algorithm, which consists of five phases. First, we categorize rules in P_1 and P_2 based on

their *effects* (line 1). Second, we compute the similarity score S_{rule} for each pair of rules in P_1 and P_2 (line 2-7). Third, based on S_{rule}, we compute the Φ mappings (line 8-11).

Algorithm PolicySimilarityMeasure(P_1, P_2)
Input : P_1 is a policy with n rules $\{r_{11}, r_{12}, ..., r_{1n}\}$
and P_2 is a policy with m rules $\{r_{21}, r_{22}, ..., r_{2m}\}$

1. Categorize rules in P_1 and P_2 based on their *effects*.
Let $PR_1(PR_2)$ and $DR_1(DR_2)$ denote the set of permit
and deny rules respectively in $P_1(P_2)$.

/* Compute similarity scores for each rule in P_1 and P_2 */
2. **foreach** rule $r_{1i} \in PR_1$
3. **foreach** rule $r_{2j} \in PR_2$
4. $S_{rule}(r_{1i}, r_{2j})$ //compute similarity score of rules
5. **foreach** rule $r_{1i} \in DR_1$
6. **foreach** rule $r_{2j} \in DR_2$
7. $S_{rule}(r_{1i}, r_{2j})$ //compute similarity score of rules

/* Compute Φ mappings */
8. $\Phi_1^P \leftarrow$ComputePhiMapping(PR_1, PR_2, ϵ)
9. $\Phi_2^P \leftarrow$ComputePhiMapping(PR_2, PR_1, ϵ)
10. $\Phi_1^D \leftarrow$ComputePhiMapping(DR_1, DR_2, ϵ)
11. $\Phi_2^D \leftarrow$ComputePhiMapping(DR_2, DR_1, ϵ)

/* Compute the rule set similarity scores */
12. **foreach** rule $r_{1i} \in P_1$
13. **if** $r_{1i} \in PR_1$ **then**
14. $rs_{1i} \leftarrow$ ComputeRuleSimilarity(r_{1i}, Φ_1^P)
15. **elsif** $r_{1i} \in DR_1$ **then**
16. $rs_{1i} \leftarrow$ ComputeRuleSimilarity(r_{1i}, Φ_1^D)
17. **foreach** rule $r_{2j} \in P_2$
18. **if** $r_{2j} \in PR_2$ **then**
19. $rs_{2j} \leftarrow$ ComputeRuleSimilarity(r_{2j}, Φ_2^P)
20. **elsif** $r_{1i} \in DR_1$ **then**
21. $rs_{2j} \leftarrow$ ComputeRuleSimilarity(r_{2j}, Φ_2^D)
22. $S_{rule-set}^P \leftarrow$ average of rs of permit rules
23. $S_{rule-set}^D \leftarrow$ average of rs of deny rules

/* Compute the overall similarity score */
24. $S_{policy}(P_1, P_2) = S_T(P_1, P_2) + w_p S_{rule-set}^P + w_d S_{rule-set}^D$

end PolicySimilarityMeasure.

Figure 3.10. Algorithm for Policy Similarity Measure

Fourth, we use the Φ mappings to calculate the rule set similarity scores (line 12-23). Finally, the overall similarity score is obtained (line 24).

The most computationally expensive part of the algorithm is to compute S_{rule}. We analyze its complexity as follows. S_{rule} is the sum of similarity scores of corresponding elements. Suppose that the average number of attributes in one element is n_a. To find matching attributes with the same name, it takes O($n_a$logn_a) to sort and compare the list of attribute names. For each pair of matching attributes, we further compute the similarity scores of attribute values. Generally speaking, one attribute name is associated with one or very few number of values (e.g. ≤ 10). Therefore, we estimate the time for the attribute value computation to be a constant time c. Then the complexity of computing a similarity score of two elements is O($n_a$logn_a+$n_a c$). For each rule, there are at most 5 elements, and the computation complexity of S_{rule} is still O($n_a$logn_a).

Procedure ComputeRuleSimilarity(r', Φ)
Input : r' is a rule and Φ is a mapping between rules
1. **foreach** rule $r'' \in \Phi$
2. sum = sum $+S_{rule}(r', r'')$
3. rs = $\frac{sum}{|\Phi|}$
4. **return** rs

end ComputeRuleSimilarity.

Figure 3.11. Procedure for Computing Rule Similarity

It is worth noting that n_a is usually not a big value. For an entire policy, the total number of attribute-value pairs tested in [5] is 50. The maximum number of attribute-value pairs in one policy we have seen so far is about 500 [17]. Considering that the average number of attributes in one policy component is even smaller, our similarity score computation is very efficient.

3.1.2.12 Case Study

In this section we present a detailed example to illustrate how our policy similarity measure algorithm works. Continuing with the policy examples P_1, P_2 and P_3 introduced in Section 2, we show how our policy similarity algorithm assigns a similarity score to these policies. We further show that our similarity algorithm assigns a higher similarity score

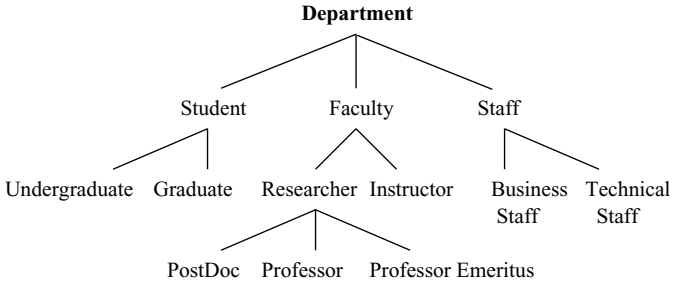


Figure 3.12. User Hierarchy in the University Domain

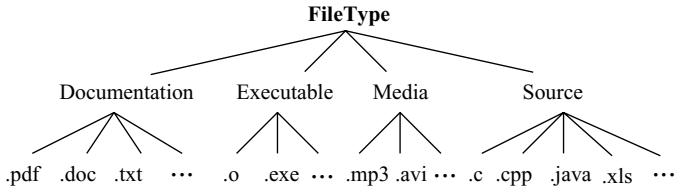


Figure 3.13. File Hierarchy in the University Domain

between the data owner policy P_1 and resource owner policy P_2 than between the data owner policy P_1 and resource owner policy P_3, adequately representing the relationship between the sets of requests permitted(denied) by the corresponding policies. Thus using the similarity score computed by our algorithm, the data owner is notified that P_2 is more compatible to its own policy. The data owner only need to further check one policy P_2 instead of testing two policies before sending its data to the resource owner.

In the following discussion we refer to the policies shown in Figures 3.2, 3.3 and 3.4. We also refer to two attribute hierarchies in the domain, namely the user hierarchy (Figure 3.12) and file type hierarchy (Figure 3.13). Without having any additional knowledge of the application, we assume that each rule component is equally important and hence assign the same weight to all computations.

The similarity score between P_1 and P_2 is calculated as follows:

1. We categorize rules in P_1 and P_2 based on their effects and find the permit and deny rule sets, $PR_1(PR_2)$ and $DR_1(DR_2)$. These sets are

$$PR_1 = \{R11\}$$
$$PR_2 = \{R21, R22\}$$
$$DR_1 = \{R12\}$$
$$DR_2 = \{R23, R24\}$$

2. We compute the rule similarity scores between pairs of rules with the same effect in both policies.

$$S(R11, R21) = 0.81$$
$$S(R11, R22) = 0.56$$
$$S(R12, R23) = 0.81$$
$$S(R12, R24) = 0.76$$

3. For policy P_1, we find the Φ mappings Φ_1^P and Φ_1^D using the **ComputePhiMapping** procedure. We use 0.7 as the value of the threshold for this example when computing the mappings. The Φ mappings obtained for policy P_1 are as follows:

$$\Phi_1^P = \{R11 \rightarrow \{R21\}\}$$
$$\Phi_1^D = \{R12 \rightarrow \{R23, R24\}\}$$

4. The Φ mappings Φ_2^P and Φ_2^D are calculated similarly for policy P_2.

$$\Phi_2^P = \{R21 \rightarrow \{R11\}, R22 \rightarrow \{\}\}$$
$$\Phi_2^D = \{R23 \rightarrow \{R12\}, R24 \rightarrow \{R12\}\}$$

5. For each rule r_{1i} in P_1, the corresponding rule similarity score rs_{1i} is computed:

$$rs_{11} = S_{rule}(R11, R21) = 0.81$$
$$rs_{12} = \frac{1}{2}\big[S_{rule}(R12, R23) + S_{rule}(R12, R24)\big] = 0.79$$

6. For each rule r_{2j} in P_2, the corresponding rule similarity score rs_{2j} is computed:

$$rs_{21} = S_{rule}(R11, R21) = 0.81$$
$$rs_{22} = 0$$
$$rs_{23} = S_{rule}(R12, R23) = 0.81$$
$$rs_{24} = S_{rule}(R12, R24) = 0.76$$

7. Then, the similarity between the permit rule sets of P_1 and P_2, given by $S^P_{rule-set}$ is computed:

$$\begin{aligned} S^P_{rule-set} &= \frac{rs_{11} + rs_{21} + rs_{22}}{3} \\ &= \frac{0.81 + 0.81 + 0}{3} \\ &= 0.54 \end{aligned}$$

8. The similarity between the deny rule sets of P_1 and P_2, given by $S^D_{rule-set}$, is computed:

$$\begin{aligned} S^D_{rule-set} &= \frac{rs_{12} + rs_{23} + rs_{24}}{3} \\ &= \frac{0.79 + 0.81 + 0.76}{3} \\ &= 0.79 \end{aligned}$$

9. Finally the permit and deny rule set similarities and policy target similarities are combined to obtain the overall policy similarity score between policies P_1 and P_2:

$$\begin{aligned} S_{policy}(P_1, P_2) &= \frac{1}{3}S_T + \frac{1}{3}S^P_{rule-set} + \frac{1}{3}S^D_{rule-set} \\ &= \frac{1}{3}\cdot 0.75 + \frac{1}{3}\cdot 0.54 + \frac{1}{3}\cdot 0.79 \\ &= \mathbf{0.71} \end{aligned}$$

We then calculate the policy similarity score for policies P_1 and P_3. The policy target similarity score $S_T = 0.5$. The rule similarity scores for policies P_1 and P_3 are:

$$S(R11, R21) = 0.7$$
$$S(R12, R23) = 0.66$$

By using the threshold 0.7, we obtain the following Φ mappings:

$$\Phi^P_1 = \{R11 \rightarrow \{R31\}\}$$
$$\Phi^D_1 = \{R12 \rightarrow \{\}\}$$

Following the same steps as described for policies P_1 and P_2, we have the following similarity score between P_1 and P_3.

$$\begin{aligned} S_{policy}(P_1, P_3) &= \frac{1}{3}S_T + \frac{1}{3}S^P_{rule-set} + \frac{1}{3}S^D_{rule-set} \\ &= \frac{1}{3}\cdot 0.5 + \frac{1}{3}\cdot 0.7 + \frac{1}{3}\cdot 0 \\ &= \mathbf{0.4} \end{aligned}$$

We observe that policy P_1 is clearly more similar to policy P_2 than to policy P_3. Hence, the data owner will be suggested to carry out the fine-grained policy analysis with P_2 first.

Next we briefly discuss an example of two semantically equivalent but syntactically different policies. Policies P_8 and P_9 share the same policy targets. P_8 has only two permit rules R_{81} and R_{82}. R_{81} contains only one condition component which is ($5am < t < 8am$). R_{82} also contains only one condition component which is ($3am < t < 6am$). P_9 has

only one permit rule which is ($3am < t < 8am$). It is clear that the two policies are equivalent. Their similarity score is close to 1 as calculated in the following, which also satisfies Property 1.

$$S(R81, R91) = 0.98$$
$$S(R82, R91) = 0.98$$

$$\begin{aligned} S^{P}_{rule-set} &= \frac{rs_{81} + rs_{82} + rs_{91}}{3} \\ &= \frac{S(R81, R91) + S(R82, R91) + S(R82, R91)}{3} \\ &= \frac{0.98 + 0.98 + 0.98}{3} \\ &= 0.98 \end{aligned}$$

$$\begin{aligned} S_{policy}(P_8, P_9) &= \frac{1}{3}S_T + \frac{1}{3}S^{P}_{rule-set} + \frac{1}{3}S^{D}_{rule-set} \\ &= \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 0.98 + \frac{1}{3} \cdot 1 \\ &= \mathbf{0.99} \end{aligned}$$

3.1.3 Experimental Evaluation

We have implemented a prototype of the proposed similarity measure techniques using Java. We have performed extensive testing of the implementation on randomly generated access control policies. We evaluated both the effectiveness and efficiency of our lightweight policy similarity measure in contrast to exhaustive policy comparison techniques which involve Boolean expression analysis.

We have used the Falcon-AO v0.7 [18] ontology mapping implementation for performing ontology matching. The 2007 Ontology Alignment Evaluation Initiative(OAEI 07) results indicate Falcon to be the best performing ontology matcher available. We have used the WordNet2.1 Java API corresponding to the WordNet [15] English language lexi-

cal database for implementing the functions dealing with finding the semantic variations in names.

All experiments were conducted on 3Gz Pentium III processor machine with 500MB RAM.

3.1.3.1 Random generation of access control policies

We implemented a random attribute based access control policy generator (*RACPG*). *RACPG* generates policies in two formats : (i) simple XACML format which serves as input for the policy similarity measure and (ii) Boolean expression format which serves as input for the Boolean policy comparison. Each generated policy contained conditions on attributes randomly chosen from a list of 21 attributes. Out of the 21 attributes, there were 10 categorical, 7 numerical, 2 date and 2 time attributes. Out of the 10 categorical attributes 4 of them were associated with hierarchies. A maximum of 12 attribute-conditions were included in each policy element. Each policy or a rule in a policy could have a target with probability 0.5.

In case of policies generated for testing the policy similarity measure with ontology matching we used concepts randomly chosen from the swportal([19]) and swrc_updated([20]) ontologies that are available online. In addition we also introduced semantic variations (synonyms) in the attribute list.

We first evaluated the effectiveness and efficiency of the policy similarity measure. These set of experiments were conducted without considering the ontology matching and dictionary lookup techniques. We then measured the scalability of the implementation for both versions with and without ontology. Finally we looked in detail the differences obtained with respect to the similarity scores when using ontology matching and dictionary lookup techniques.

3.1.3.2 Effectiveness

Since our policy similarity measure is an approximation of the similarity between two policies, in order to demonstrate the effectiveness of the similarity measure, we compared our results with those obtained by an exact policy similarity analyzer. The exact policy similarity analyzer represents each policy as a Boolean expression and constructs a corresponding MTBDD. When comparing two policies, the MTBDD of the two policies are combined to determine the differences between the two policies. More information on such technique can be found in [21]. The output of the exact policy similarity analyzer is a list of requests and effects of the two policies for these requests. Based on this information, we can quantify the differences between two policies using the percentage of the requests for which the two policies have different effects. The higher the percentage of such requests the less similar the policies are.

Each policy pair in *set-4* and *set-8* was input to both the policy similarity measure and the exact policy similarity analyzer. For each policy pair a policy similarity score and a policy difference percentage was recorded. The test sets *set-4* and *set-8* each contained 100 pairs of policies. In *set-4* each policy had 4 rules each and in *set-8* each policy had 8 rules each. The maximum number of attribute predicates in any given policy was 68 for *set-4* and 124 for *set-8*. Considering that for typical policies we have encountered in real world applications the average number of atomic Boolean expressions lies between 10 and 50, our test sets covered a much bigger range.

Figure 3.14 shows the policy similarity score and policy difference percentage for policy pairs in *set-4* and *set-8* with the threshold ϵ set to 0.75. We observe that policy similarity scores decrease when the differences between two policies increase. This indicates that our policy similarity measure provides a good approximation of the similarity between policies.

3.1.3.3 Effect of varying ϵ

We evaluated the effect of the threshold ϵ by varying ϵ from 0.2 to 0.75 for the two test tests. The result is shown in Figure 3.15. Observe that for both test sets *set-4* and

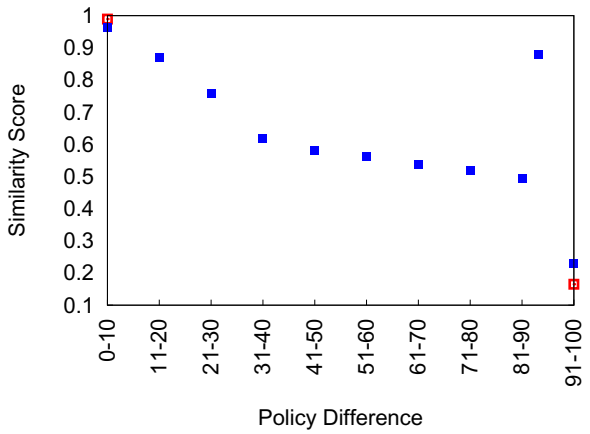
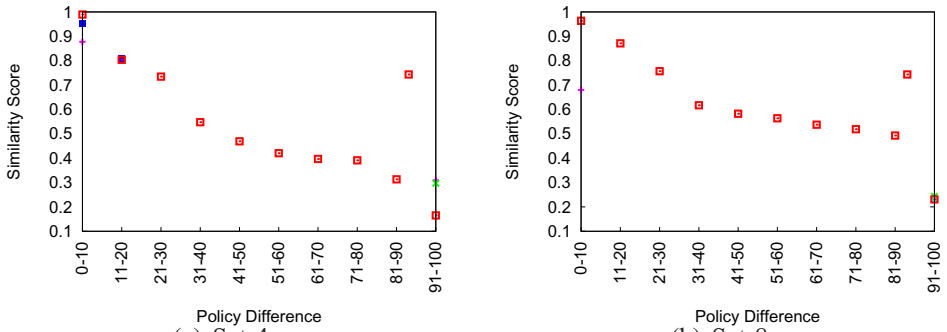


Figure 3.14. Policy Similarity Scores versus Policy Difference



(a) Set-4 (b) Set-8

Figure 3.15. Effect of Varying ϵ

set-8 higher values of ϵ tend to provide a better approximation. This is because the overall similarity score is the average of the rule similarity scores above ϵ and using higher values of ϵ prunes more rules which are less similar to one another.

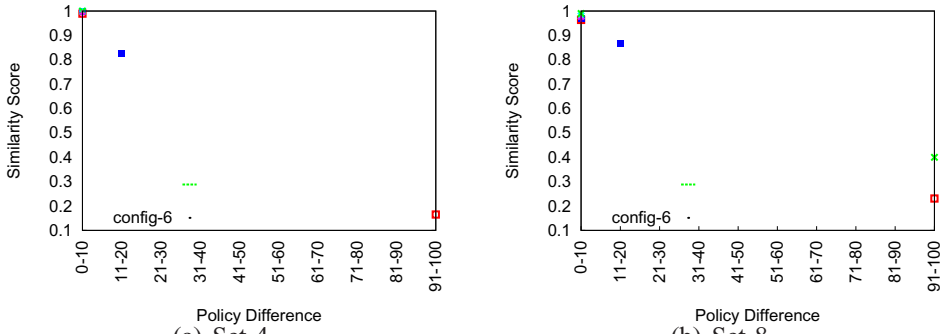
3.1.3.4 Effect of varying policy element weights

The computation of policy similarity score is associated with five policy element weights namely w_T:*policy target weight*, w_p:*permit set weight*, w_d:*deny set weight*, w_t:*rule target weight*, w_c:*condition weight*. In this set of experiments we evaluated the effect of varying these element weights on the similarity score computation.

Table 3.2
Configurations with varying policy element weights

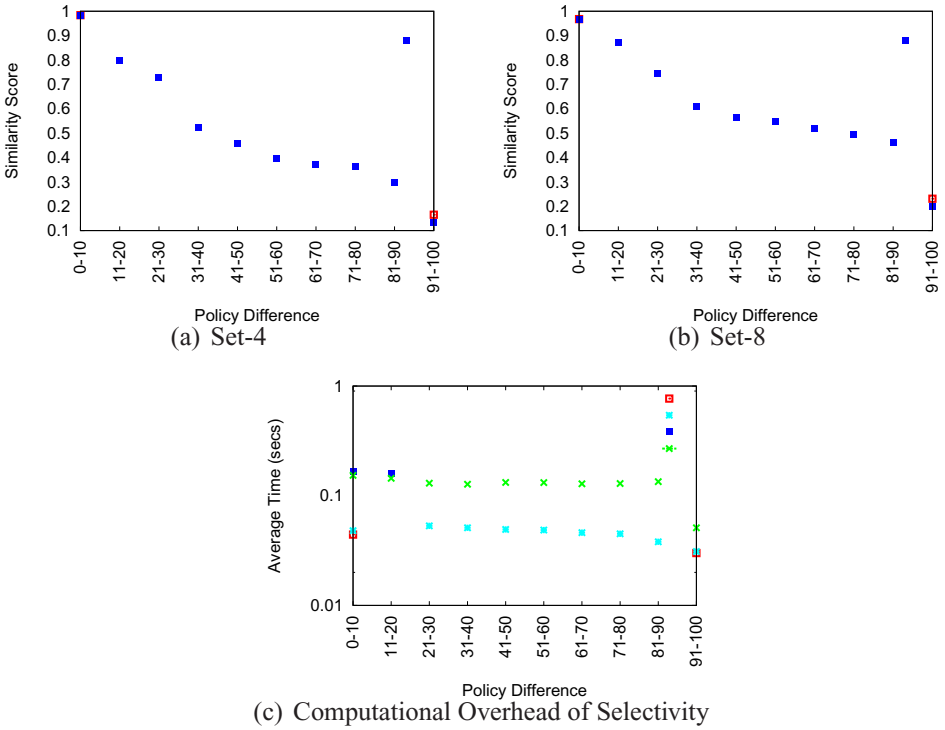
Configuration	**Policy Target Weight** (w_T)	**Permit Set Weight** (w_p)	**Deny Set Weight** (w_d)	**Rule Target Weight** (w_t)	**Rule Condition Weight** (w_c)
config-1	0.33	0.33	0.33	0.5	0.5
config-2	0.25	0.375	0.375	0.5	0.5
config-3	0.5	0.25	0.25	0.5	0.5
config-4	0.75	0.125	0.125	0.5	0.5
config-5	0.33	0.33	0.33	0.25	0.75
config-6	0.33	0.33	0.33	0.75	0.25

We obtained the policy similarity scores for policy pairs in *set-4* and *set-8* for six different configurations of the policy similarity computation. Each configuration was associated with different values of policy element weights as shown in Table 3.2. The threshold(*epsilon*) value was set at 0.75 for all the configurations. Figure 3.16 shows the results obtained for *set-4* and *set-8*. We observe for both policy sets that *config-1*, *config-2* and *config-5* yielded similar results with respect to effectiveness of the similarity scores. However, *config-3* and *config-4* in which the policy target weight w_T was given higher weight compared to *permit-* and *deny*-set weights produced lot of false positives resulting in policy pairs with higher difference percentage being assigned higher similarity scores. Similarly, *config-6* which assigned higher weight to w_t, the rule target weight compared to the condi-



(a) Set-4 (b) Set-8

Figure 3.16. Effect of Varying Policy Element Weights



(a) Set-4 (b) Set-8

(c) Computational Overhead of Selectivity

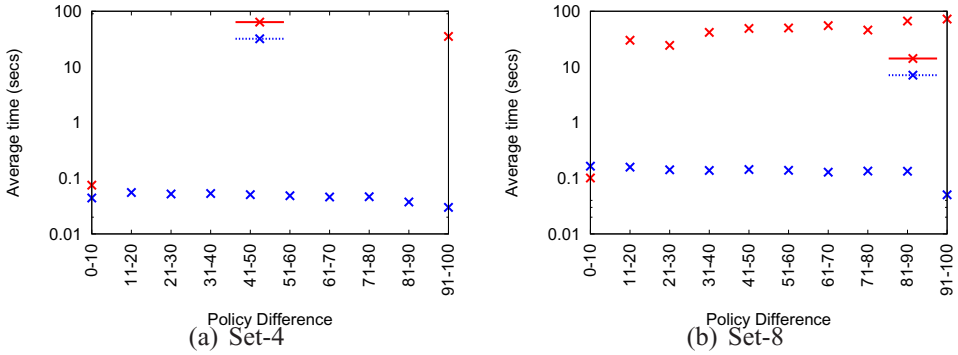
Figure 3.17. Effect of Selectivity

tion weight w_c did not perform well. The results obtained from this experiment re-iterates our general guideline to assign equal values to the various policy element weights. However, a user may want to assign unequal values if he wants to find policies which are similar with respect to one particular element.

3.1.3.5 Effect of Operators and Selectivity

Similar experiments for policy pairs in *set-4* and *set-8* are conducted by using the variant of the policy similarity measure that considers operators in predicates (in Section 3.1.2.8). We observe that the trends of scores obtained from the variant are similar to that of the original version which did not consider operators. Here, we did not see significant improvement [1] over the score accuracy after considering operators. This is because that policies with

[1] By improvement we mean that there is a decrease in the value of the scores obtained for policy pairs with higher policy difference percentage



(a) Set-4 (b) Set-8

Figure 3.18. Execution time

similar attributes but totally different operators do not occur very often in general. In other words, the consideration of operators mainly help adjust similarity scores in the special cases when two policies contain same attributes but differ in operators.

We also evaluate the effect of incorporating selectivity (discussed in Section 3.1.2.9) computation while computing the scores. The results for *set-4* and *set-8* are shown in Figure 3.17. We observe that by adjusting the similarity score based on selectivity there is a slight improvement in the effectiveness of the scores especially for policy pairs that are not very similar. In effect, incorporating selectivity decreases the similarity scores for policy pairs that are not very similar thereby improving the effectiveness. It is also worth noting that this slight improvement does not come at the cost of efficiency as can be observed from Figure 3.17(c).

3.1.3.6 Efficiency

The previous set of experiments demonstrate the effectiveness of the policy similarity measure. In order for our technique to be useful as a light weight approach that can quickly rank policies, it must also be efficient. We compared the execution time of the policy similarity measure with that of the exact policy similarity analyzer. The same data sets *set-4* and *set-8* were used. The results for *set-4* and *set-8* are shown in Figures 3.18. Each

point in the graphs corresponds to the average execution time for 10 different policy pairs. The value of ϵ was set to 0.5.

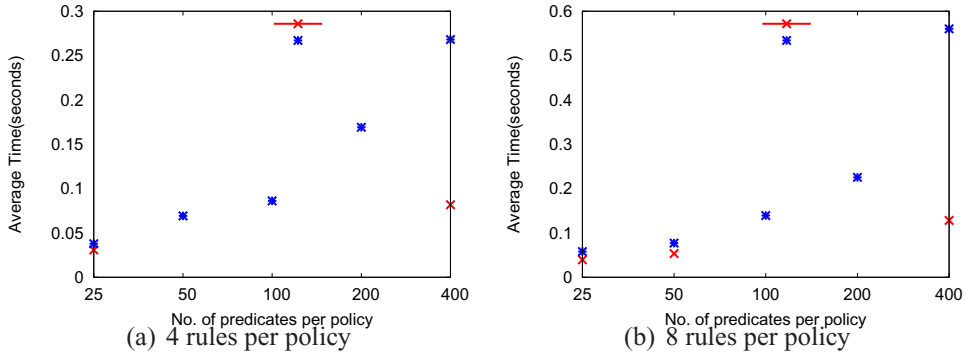
From the figures, we observe that the policy similarity measure almost remains constant for both *set-4* and *set-8*. This is because the time taken by the policy similarity measure depends on the number of rules and attribute predicates in the policies being compared which is constant for policies in both sets. In contrast, the time taken by the exact policy similarity analyzer is a function of the size of the resulting comparison MTBDD which increases with increase in policy difference. Observe that the average execution time taken by the policy similarity measure is two to three orders of magnitude less than the time taken by the exact similarity analyzer. Such difference can be attributed to the quick comparison techniques which avoids computationally intensive Boolean expression analysis. This also indicates that considerable gain in time can be achieved by using the similarity measure before invoking the more computationally expensive similarity analysis.

3.1.3.7 Scalability

In this set of experiments, we evaluated the scalability of the policy similarity measure implementation considering both versions with and without ontology matching[2]. We used test cases that considered attribute names and values from the WordNet synonym set and ontologies as described in Section 3.1.3.1. We varied the number of attribute predicates across the policies and plotted the average time taken to compute the similarity score. For these experiments the value of the threshold ϵ was set to 0.5.

Figure 3.19 reports the average time taken to compute similarity scores for 10 different policy pairs in data sets containing 7 and 8 rules per policy, when varying the number of predicates in each policy from 25 to 400. We can observe that both versions scale reasonably well as the number of predicates per policy increases. Though the average time taken by the ontology matching version is marginally higher than that taken by the version without ontology matching, it is still two to three orders of magnitude lower than the exact

[2]We use the term "ontology mapping" to refer also to dictionary lookup



(a) 4 rules per policy (b) 8 rules per policy

Figure 3.19. Scalability as number of attribute predicates per policy is increased

similarity analyzer indicating that incorporating ontology matching as part of a filtering phase can still be feasible.

3.1.3.8 Effect of Ontology Matching on Similarity Scores

In the final set of experiments, we explored the benefit from introducing ontology matching. For these experiments we generated a random policy with 4 rules and 68 attribute predicates considering concepts from a single ontology. We then generated five different variants of this policy by replacing attribute names with synonyms and equivalent concepts belonging to another ontology. In effect, all the policies were semantically the same but only the vocabulary was different, and hence similarity score 1 is expected. We then compared the similarity scores for both versions with and without ontology. The results are shown in table 3.3. We observe that as the number of changes increases the policy similarity score without ontology indicates decreasing scores although all the policies were semantically similar while the version with ontology matching consistently gives the highest score indicating the underlying semantic similarity of the policy pairs considered. In general, for all policy pairs used in the scalability experiments, we observed that the ontology matching version gave higher similarity scores.

Table 3.3
Similarity Scores for the With and Without Ontology Versions

Policy Variant	**# of Changes**	**Score Without Ontology Matching**	**Score With Ontology Matching**
Variant1	5	0.58	0.99
Variant2	8	0.58	0.98
Variant3	12	0.55	0.99
Variant4	17	0.43	0.98
Variant5	23	0.43	0.96

3.1.4 Applications

Our similarity measure can be applied in many scenarios. In what follows, we discuss some of the potential applications.

3.1.4.1 Policy integration

With the advent of Web 2.0, collaborative applications (web services) that share and protect resources are becoming important. Such applications are associated with complex security policies. To support a secure collaboration, it is necessary to consolidate (integrate) the security policies of the parties involved in the collaboration. Techniques to determine the similarity between security policies are vital to perform such integration. Mazzoleni et al. [Mazzoleni et al. 2006] discuss integration algorithms that require the knowledge of the similarity between access control policies in terms of the relationship between the set of requests permitted (denied) by a given set of policies.

When a large number of policies are to be integrated in a time efficient manner, integrating more similar policies first can reduce the overall integration time since less conflict needs to be resolved during each round of integration. Our similarity measure can be used to quickly identify those similar policies without invoking time-consuming Boolean similarity analysis. Thus the time overall time needed to perform the integration can be reduced.

3.1.4.2 Policy clustering

Clustering is an important technique for discovering interesting data patterns. Policy clustering can help in understanding the most commonly occurring entities in policies, in finding meta-policies and policy writing guidelines for different types of organizations. Our similarity measure can serve as a good distance function among policies so that it can be used with existing clustering algorithms.

3.1.4.3 Language independent policy comparison

Although the proposed policy similarity measure is defined for XACML policies, we should notice that the underlying schema of XACML policies is XML. It is not surprising that our policy similarity measure can be easily adapted to compare any attribute-based access control policies expressed in a XML based language like P3P (Platform for Privacy Preferences) policies.

P3P policy comparison

The P3P policy is a World Wide Web Consortium (W3C) standard for expressing privacy policies of a website. A similarity measure for comparing two P3P policies can be useful when a website visitor would like to ensure that privacy policy of a website he wishes to conduct business with is similar to an *ideal* privacy policy reflecting his privacy preferences. It can also be used to rank websites based on similarity between an ideal privacy policy and the privacy policies of the websites. In the following, we briefly discuss how to adapt our current similarity measure to this case.

A P3P policy is mainly composed of statements that describe the data and category of information collected along with how the information may be used, how the information may be shared and the associated data retention policies. Thus a P3P policy can be abstracted into a list of tuples each containing the data, category, purpose, recipient and retention elements. Each of the elements can contain values belonging to a pre-defined set specified in the P3P specification. A similarity score between two P3P policies is derived by computing the similarity score between pairs of tuples corresponding to the two policies.

A score between a pair of tuples is calculated by comparing the individual components of the two tuples.

Further, since our policy similarity measure is defined by comparing the corresponding components in two policies, the basic idea can be used for comparing policies not even written in XML. We consider the web server configuration files in Apache and SELinux policy as examples.

Web server configuration file comparison

Web server configuration files also contain access control information and are used to direct traffic from browsers to applications running at the server. In particular, such files specify whether a requested operation can be performed by certain applications. For collaboration purpose, different web servers will need to check if their configuration files allow the same set of requests to the same applications. Again, they can use a similarity measure to quickly obtain a basic idea on the similarity of their configuration files before the collaboration.

In the Apache web-server, the main configuration file "httpd.conf" contains the configuration directives that give the server its instructions. The configuration directives are grouped into three basic sections: 1) Directives that control the operation of the Apache server process as a whole; 2) Directives that define the server type; 3) Settings for virtual hosts. To obtain a similarity measure for such files, we can follow the basic idea of the XACML policy comparison and summarize the similarities between each corresponding sections.

Security-Enhanced Linux (SELinux) policy comparison

The SELinux policy [22] is a set of rules that guide the SELinux security engine. It defines types for file objects and domains for processes. Rules (referred to as access vector rules) in the policy determine how each domain may access each type. Only what is specifically allowed by the rules is permitted. By default, every operation is denied.

A SELinux policy consists of many components like commons, object classes, types, attributes, access vector rules, type rules, users, roles, role allow rules, role and range transition rules. Typically a policy consists of thousands of access vector rules which makes

the search of similar policies using tools like sediff and sediffx [23] a time-consuming task. A quick similarity computation among pairs of access vectors rules, using techniques proposed in this work, can be useful for pruning dissimilar policies. Another scenario where a similarity score between SELinux policies can be useful is when an administrator who has to manage a large cluster of servers each with its own policy configuration would like to ensure that every server's policy has similar level of security as specified in an *ideal* server policy configuration.

An access vector rule in a SELinux policy is made up of four components: (i) an *access vector* which could be one of the values in the set {`allow`, `neverallow`, `auditallow`, `dontaudit`},(ii) the *source* type, (iii) the *target* type and (iv) *classes* or list of *permissions*. The *access vector* component can be regarded as analogous to the *Effect* in XACML rules and the access vector rules can be first grouped based on the value of this component. Now for each pair of rules that have the same access vector value we can compare the corresponding source, target and permission components and compute a score based on each of these and aggregate the obtained results to get a similarity score between two access vector rules.

Similar techniques can be used to derive a similarity score not only for the different kinds of rules in a SELinux policy like the type rules, role allow rules and role transition rules but also for other components like the types, commons and users. Considering that SELinux policies support role-based access control, we can also utilize the hierarchy distance based measures proposed here to find similarity between roles in the SELinux policies. The scores obtained for individual components can be combined as a weighted aggregrate to determine a similarity score for two SELinux policies.

3.1.5 Usability Survey

In order to assess the practical value of the proposed policy similarity measure, we conducted a pilot survey involving system administrators and graduate students. We developed a questionnaire (see Appendix) which contained 30 pairs of security policies in natural lan-

guage. Each participant was asked to rate the similarity of policies in each of the policy pairs on a scale of 1-7 where 1 corresponds to *Not similar at all* and 7 corresponds to *Very similar*. We also asked them to give a rating for the *appropriateness* [3] of each of the policies on a scale of 1-7 where 1 corresponds to *Not appropriate at all* and 7 corresponds to *Very appropriate*. 8 system administrators and 22 students participated in this pilot study.

We converted each of the policies to an intermediate form as required for the policy similarity measure and computed a similarity score for each policy pair using the techniques presented in subsection 3.1.2. We converted the similarity score which is a value between 0 and 1 to a corresponding value between 1 and 7 in order to compare the scores with the similarity ratings given by participants. We plotted the similarity score assigned by our policy similarity measure and the average of similarity ratings assigned by all the participants for each policy pair. The results are shown in Figure 3.20. We observed that the policy similarity measure conforms well to the similarity ratings assigned by the users and is a good predictor of which policies users may evaluate as similar.

Encouraged by the promising results of the pilot study, we conducted another survey with larger number of participants involving information security executives, system administrators and students. This larger study involved 82 students and 30 professionals. Results of this larger survey(Figure 3.21) showed trends similar to that of the pilot study thus further establishing that our proposed similarity measure can be a useful tool to assess the similarity of policies.

3.2 Policy Similarity Analyzer

PSA is the key component of EXAM in that it implements the analysis queries. In what follows, we describe its architecture, detailed construction algorithms and the query processing strategies.

[3] Although appropriateness is not directly related to our work on policy similarity we included this as part of another study to understand different stakeholders perspectives toward security policies.

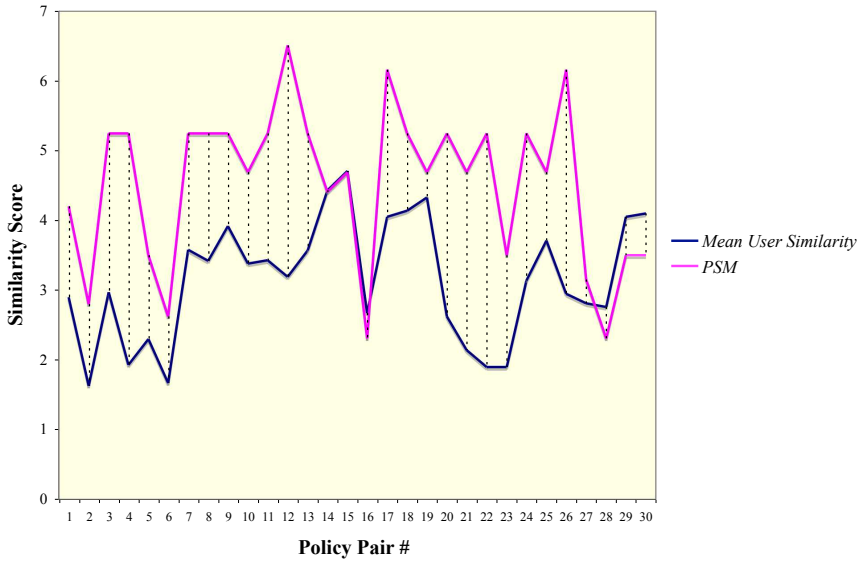


Figure 3.20. Results of Similarity Survey - A Pilot Study

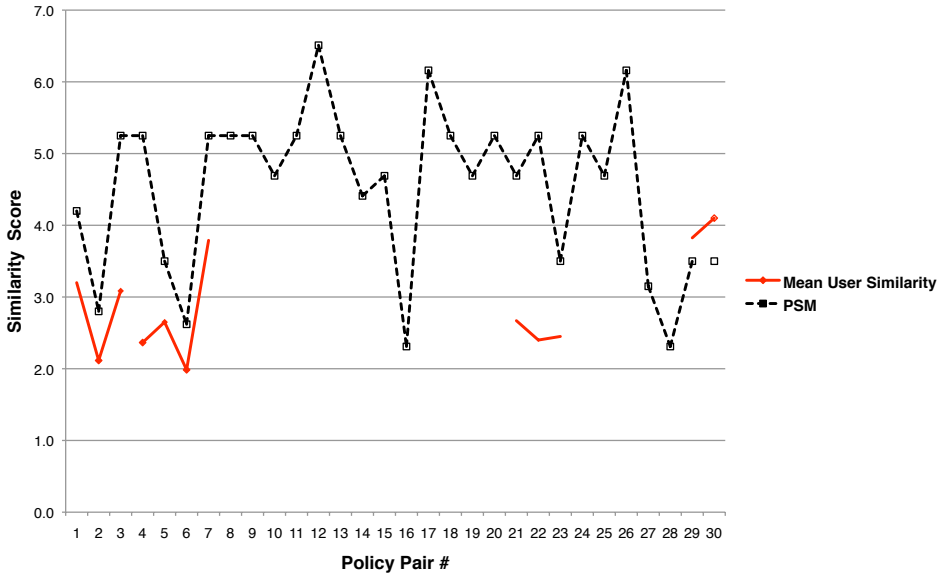


Figure 3.21. Results of Policy Similarity Survey

3.2.1 Architecture of PSA

The problem of analyzing policies can be translated into the problem of analyzing Boolean formulae. The main task of the PSA module is to determine all variable assign-

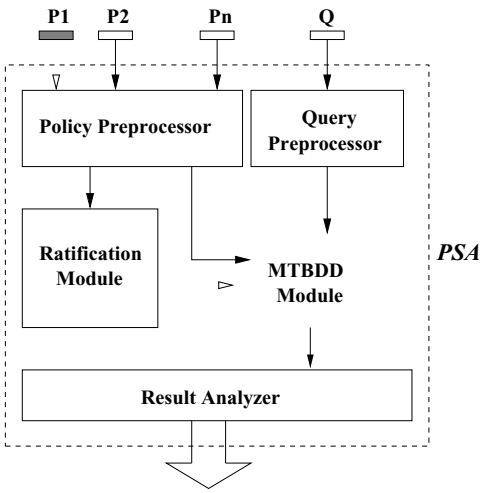


Figure 3.22. Architecture of the Policy Similarity Analyzer (PSA)

ments that can satisfy the Boolean formulae corresponding to one or more policies, and also variable assignments that lead to different decisions for different policies. The basic idea is to combine functionalities of the policy ratification technique [13] and MTBDD technique [5] by using a divide-and-conquer strategy.

Figure 3.22 shows the architecture of PSA. Policies are first passed to a *preprocessor* which identifies parts to be processed by the *ratification module* and parts to be directly transmitted to the *MTBDD module*. The ratification module then generates unified nodes and a set of auxiliary rules that are transmitted to the MTBDD module. The MTBDD module then creates a combined MTBDD that includes policies and additional rules. By using the combined MTBDD, the PSA module can thus process the queries that we introduced in Section 2.3. Specifically, queries on a single policy are carried out on the MTBDD of the policy being queried, whereas queries on multiple policies are carried out on the CMTBDD of corresponding policies. Finally, the result analyzer reformats the output of the MTBDD module and reports it to the users.

In the following sections, we first introduce how to represent a policy using a MTBDD and then present the details of policy analysis based on such representation. Finally, we discuss the policy query processing.

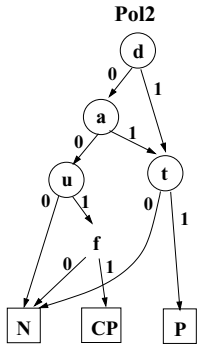


Figure 3.23. The MTBDD for policy P_2

3.2.2 Policy Representation

Given an input policy, the policy preprocessor translates it into at most two compound Boolean expressions (category 5) which correspond to the permit and deny effects respectively. The compound Boolean expressions are composed of atomic Boolean expressions which usually belong to the first four categories, i.e., one variable equality, one variable inequality, real valued linear and regular expression constraints as presented in Section 2.3.1. Example 3 shows the Boolean expressions of policy P_1 and P_2.

The compound Boolean expressions of a policy are represented as a MTBDD. The structure of a MTBDD is a rooted acyclic directed graph. The internal nodes represent atomic Boolean expressions and the terminals represent policy effects, i.e., Permit(P), Deny(D) and NotApplicable (NA). Each non-terminal node has two edges labeled 0 and 1 which means that the atomic Boolean expression associated with this node is unsatisfied or satisfied respectively. Nodes along the same path have "$\wedge$"(AND) relationship and nodes in the different paths have "$\vee$" (OR) relationship. Each path in the MTBDD represents a set of requests that satisfy the atomic Boolean expressions in the nodes with 1-edge along the path, and the terminal at the end of the path represents the effect of the policy for the set of requests. While in the worst case the number of nodes in an MTBDD is exponential in the number of variables, in practice the number of nodes is often polynomial or even linear [5].

Example 10 Figure 3.23 shows the MTBDD for policy P_2. The MTBDD has five nodes and three terminals. Nodes d, a, t, u and f stand for atomic Boolean expressions (domain = ".edu"), ($affliation$= "IBM"), (6am $\leq$t$\leq$8pm), (user="Bob") and (upload + download $<$ 1GB), respectively. Terminals "N", "CP" and "P" stand for "NotApplicable", "Conditional Permit" and "Permit" respectively. Take the right most path as an example. Such path indicates that if a request satisfies Boolean expressions in nodes d and t, the request will be permitted by policy P_2.

From Figure 3.23, we notice a new terminal "CP" which means *conditional permit*. Such a terminal indicates that there exist some requests satisfying the Boolean expressions along the paths ending at this terminal but the variable assignments cannot be directly derived from the internal nodes due to the existence of linear constraints or regular expressions. Checking whether the Boolean expressions along that path is satisfiable is the task of the ratification module which will be detailed in the next subsubsection. Similarly, we can define another terminal "CD" (*conditional deny*).

3.2.3 Policy Comparison

To compare policies, their corresponding MTBDDs are combined to form a combined MTBDD (CMTBDD) by a binary operation called `Apply` [24]. MTBDDs to be combined need to follow the same variable ordering, i.e. the ordering that determines which node precedes another. We first consider the CMTBDD constructed from two policies. The `Apply` operation is a recursive operation that traverses two MTBDDs simultaneously starting from the root node. If the currently retrieved nodes of the two MTBDDs are the same, the node will be kept and the `Apply` operation is applied to the left children of both nodes, and the right children of both nodes separately. If node N_1 of MTBDD$_1$ precedes N_2 of MTBDD$_2$, N_1 will be kept in the CMTBDD and the `Apply` operation continues to compare N_2 with both left and right children of N_1. When the terminals of both MTBDDs are reached, the terminal of the CMTBDD is obtained by combining the effects of the two terminals. Since each MTBDD has five terminals: P(Permit), D(Deny), CP(Conditional Permit), CD

(Conditional Deny) and N(NotApplicable), a CMTBDD has twenty-five terminals, one for each ordered pair of results from the policies being compared (such as P-P, P-D). A high level description of the `Apply` operation is shown in Figure 4.3. For multiple policies, we can construct CMTBDD for each pair of policies to be compared and then aggregate the analysis results.

The construction of the CMTBDD is for the purpose of supporting policy analysis queries. However, if we construct the CMTBDD without analyzing the Boolean expressions represented by nodes in MTBDDs, the resulting CMTBDD may contain useless information as shown by Example 11.

Procedure Apply(N_1, N_2)
Input : N_1, N_2 are MTBDD nodes

1. initiate N_c // N_c is the node in the CMTBDD
2. **if** N_1 and N_2 are terminals **then**
3. $N_c \leftarrow (N_1.var \ + \ N_2.var, null, null)$
4. **else**
5. **if** $N_1.var = N_2.var$ **then**
6. $N_c.var \leftarrow N_1.var$
7. $N_c.left \leftarrow$ Apply($N_1.left$, $N_2.left$, OP)
8. $N_c.right \leftarrow$ Apply($N_1.right$, $N_2.right$, OP)
9. **if** $N_1.var$ precedes $N_2.var$ **then**
10. $N_c.var \leftarrow N_1.var$
11. $N_c.left \leftarrow$ Apply($N_1.left$, N_2, OP)
12. $N_c.right \leftarrow$ Apply($N_1.right$, N_2, OP)
13. **if** $N_2.var$ precedes N_1.var **then**
14. $N_c.var \leftarrow N_2.var$
15. $N_c.left \leftarrow$ Apply($N_2.left$, N_1, OP)
16. $N_c.right \leftarrow$ Apply($N_2.right$, N_1, OP)
17. return N_c

Figure 3.24. Description of the `Apply` operation

Example 11 The left part of Figure 3.25 shows the MTBDDs of policies P3 and P4 and their CMTBDD P34 constructed by the `Apply` operation. Policy P3 allows access during time 6am to 8am while policy P4 allows access during time 2pm to 4pm. Since these two time ranges are disjoint, the path shown as a dashed line in their CMTBDD should not exist, i.e., no request can satisfy this path.

The right part of Figure 3.25 shows the MTBDDs of policies P5 and P6 and their CMTBDD P56. Policy P5 allows access when the condition "$x < 0 \wedge x + y > 10$" is satisfied. Policy P6 allows access when the condition "$y < 0$" is satisfied. Without considering the relationship between Boolean expressions of each node, the constructed CMTBDD P56 contains one path (shown by the broken line) which can never be satisfied.

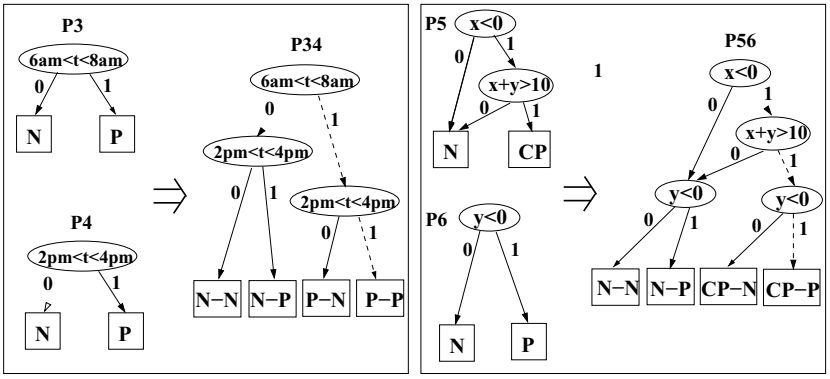


Figure 3.25. Examples of CMTBDDs

The problem in the above examples is mainly due to the existence of complex Boolean expressions of category 2, 3 and 4. To solve the problem, we propose two important operations termed as *node unification* and *auxiliary rule generation*, which are carried out in the ratification module before the MTBDD construction. An auxiliary rule has the same format of a policy rule. We proceed to present how to apply the two operations to each type of Boolean expression. Note that we do not need to take special care of Boolean expressions of category 5 since they are just combinations of previous types of Boolean expressions and such combinations are naturally reflected by the MTBDD structure.

Boolean expressions of category 1. For one variable equality constraints, we need to be careful about variables in the tree domain. For values along the same path in the tree, an auxiliary rule is needed to guarantee that if a variable cannot be assigned a certain value, then none of its children value can be satisfied. For example, suppose there are two constraints, "domain = .edu" and "domain = purdue.edu". The auxiliary rule will specify

that if the node of "domain=.edu" is *false*, the node of "domain=purdue.edu" should also be *false*. We will present how to generate such an auxiliary rule.

An auxiliary rule is represented as a Boolean expression. Let x be a variable in a tree domain and f_1, ..., f_k be a set of equality constraints on x occurring in policies to be compared. Suppose that f_1, .., f_k are in an ascending order of values of x, i.e., value in f_i is the ancestor of the value in f_j in the tree domain when $i < j$. Then we have the following auxiliary rule which specifies that f_j can be true only when every f_i ($i < j$) is satisfied. The effect of the rule is permit.

$$(f_1 \wedge \neg f_2 \cdots \wedge \neg f_k) \vee (f_1 \wedge f_2 \wedge \neg f_3 \cdots \wedge \neg f_k) \vee \cdots \vee (f_1 \wedge f_2 \cdots \wedge f_{k-1} \wedge f_k)$$

Boolean expressions of category 2. To generate unified nodes containing the Boolean expressions of category 2, i.e. one variable inequality constraints, we need to first find the disjoint domain ranges of the same variable occurring in different policies. Assume that the original domains of a variable x are $\langle d_1^-, d_1^+\rangle$, $\langle d_2^-, d_2^+\rangle$, ..., $\langle d_n^-, d_n^+\rangle$, where the superscript '-' and '+' denote lower and upper bound respectively, d_i^- can be $-\infty$, and d_i^+ can be $+\infty$ ($1 \leq i \leq n$), $\langle$ can be '[' or '(' depending on whether the lower bound is included or not and $\rangle$ can be ']' or ')' depending on whether the upper bound is included or not. We sort the domain bounds in an ascending order, and then employ a plane sweeping technique which scans the sorted domain bounds from left to right and keeps the ranges of two neighbor bounds if the ranges are covered in the original domain. The obtained disjoint ranges: $\langle d_1'^-, d_1'^+\rangle$, $\langle d_2'^-, d_2'^+\rangle$, $\cdots$, $\langle d_m'^-, d_m'^+\rangle$, satisfy the following three conditions. It is easy to prove that m is at most $4n - 2$.

(1) $d_i'^-, d_i'^+ \in D$, $D = \{d_1^-, d_1^+, ..., d_n^-, d_n^+\}$.

(2) $\cup_{i=1}^{m} \langle d_i'^-, d_i'^+\rangle = \cup_{j=1}^{n} \langle d_j^-, d_j^+\rangle$.

(3) For every k, l where $1 \leq k, l \leq m$ and $k = l$,
$\langle d_k'^-, d_k'^+\rangle \cap \langle d_l'^-, d_l'^+\rangle = \emptyset$.

After having obtained disjoint domain ranges, all related Boolean functions are rewritten by using new domain ranges. Specifically, an original Boolean function $d_j^- \lhd x \lhd d_j^+$ ($1 \leq j \leq n$, $\lhd \in \{<, \leq\}$) is reformatted as $\vee_{i=1}^{k} d_i'^- \lhd x \lhd d_i'^+$, where $\cup_{i=1}^{k} \langle d_i'^-, d_i'^+\rangle =$

$\langle d_j^-, d_j^+ \rangle$. Then, the ratification module generates unified nodes of the form of $N(f(x))$, where $f(x)$ is an inequality function in the form of $d_i'^- \lhd x \lhd d_i'^+$.

Next, we construct auxiliary rules to indicate that each time only one node of x can be assigned the value *true*. In other words, this rule tells the MTBDD module that each variable can only have one value or belong to one disjoint range during each round of the assessment. In particular, given a set of constraints on x: f_1, ..., f_k, we have the following auxiliary rule with the permit effect.

$(f_1 \wedge \neg f_2 \cdots \wedge \neg f_k) \vee (\neg f_1 \wedge f_2 \wedge \neg f_3 \cdots \wedge \neg f_k) \vee \cdots \vee (\neg f_1 \wedge \neg f_2 \cdots \wedge \neg f_{k-1} \wedge f_k)$

An example of such auxiliary rule will be given in Example 12 at the end of this section.

Boolean expressions of category 3. This type of Boolean expressions is handled during the combination of two MTBDDs. Given any one path in $MTBDD_1$ and any one path in $MTBDD_2$, the path in the CMTBDD is obtained by merging the two paths using the `Apply` operation. The SAT solver is invoked when the merged path contains atleast one linear constraint along with other constraints on common attributes. For example, (i) when both paths contain nodes[4] of linear constraints, we need to use the SAT solver to check the satisfiability of the merged path and (ii) when only one of the two paths contains nodes of linear constraints and the other path contains other constraints (e.g. equality constraint) on the variables occurring in the linear constraints, we also need to use the SAT solver to check the satisfiability of the merged path. If the Boolean expression corresponding to the merged path is satisfiable, the terminal in the CMTBDD is the combination of the terminals of MTBDD_1 and MTBDD_2. Otherwise, the terminal in the CMTBDD is "NA-NA" which means the variable assignment along the merged path does not satisfy policies corresponding to MTBDD_1 and MTBDD_2. The above steps are integrated into the `Apply` operation, specifically line 3 in Figure 4.3 which is revised to take into account the types of Boolean expressions, satisfiability check and terminal changes.

To exemplify, consider policies $P5$ and $P6$ in Figure 3.25. When the path "$x < 0 \wedge x + y > 0$" is merged with path "$y < 0$", we need to check the satisfiability of "$x < 0 \wedge x + y >$

[4]Here, we only need to consider nodes with 1-edge

$0 \wedge y < 0$". Since it is unsatisfiable, the terminal of this path should be "N-N" instead of "CP-P" shown in Figure 3.25.

Boolean expressions of category 4. For the Boolean functions of category 4, we use finite automata techniques to determine satisfiability [25]. In particular, when combining two MTBDDs, we check whether the regular expression constraints along the same path in the CMTBDD can be satisfied simultaneously. For example, consider two constraints "$x \in L(\text{"}A*\text{"})$" and "$x \in L(\text{"}B*\text{"})$" which require x to be a string with starting letter A and B respectively. Obviously, there is no assignment of x that can satisfy both constraints at the same time and we call these two constraints *conflicting constraints*. More generally, for all regular expression constraints, we first find all pairs of *conflicting constraints*. Then for each pair f_i and f_j, we construct an auxiliary rule with permit effect: $(f_i \wedge \neg f_j) \vee (\neg f_i \wedge f_j)$, which specifies that each time only one constraints can be satisfied.

The unified nodes and auxiliary rules are fed into the MTBDD module. The MTBDD module constructs a MTBDD for each policy and each auxiliary rule. Then the MTBDDs are combined and auxiliary rules are applied to the CMTBDD. When the effect of the auxiliary rule is *Permit*, the terminal function follows the original CMTBDD. When the effect of the rule is *NotApplicable*, the corresponding terminal function changes to "NA-NA". Figure 3.26 summarizes the CMTBDD construction procedure followed by the PSA module.

To illustrate the above steps, let us consider again policy P_1 and P_2 in Example 3.

Example 12 Policy P_1 and P_2 are first translated into Boolean formulae as shown in function (1) and (2) in Example 3. There are six variables occurring in these policies, namely "domain", "time", "affiliation", "user", "upload" and "download". For variables "domain", "affiliation" and "user", whose Boolean expressions belong to the first category, the preprocessor generates the following nodes:

$d(domain=\text{".}edu\text{"}), a(affiliation=\text{"}IBM\text{"}), u(user = \text{"}Bob\text{"})$.

These nodes are sent to the MTBDD module. For the Boolean formulae of variable "time" which are inequality constraints, the preprocessor sends them to the ratification module. The ratification module computes the disjoint range of the variables and obtain

Procedure CMTBDD_Construction(P_1**,** P_2**, ...,** P_n**)**
Input: P_i is a policy, $1 \leq i \leq n$

```
    /* Policy Preprocessor */
1.  translate policies into Boolean formulae BF1 and BF2
2.  for each variable x in BF1 and BF2
3.      Cx ← [f1(x), ..., fn(x)]
        // a cluster of atomic Boolean expressions with x
    /* Ratification Module */
4.      if Cx contains only Boolean expressions of category 1
5.          construct node N(fi(x)) for every fi(x)(1 ≤ i ≤ n)
6.          construct auxiliary rules for the domain constraint
7.      if Cx contains Boolean expressions of category 2
8.          compute disjoint domains of x
9.          convert every fi(x) to fi'(x) by using new domains
10.         construct auxiliary rules for the domain constraint
11.         construct node N(fi'(x)) for every fi'(x)
12.     if Cx contains Boolean expressions of category 4
13.         construct node N(fi(x)) for every fi(x)
14.         find conflicting constraints
15.         construct auxiliary rules for each conflicting constraint
    /* MTBDD Module */
16. construct an MTBDD for each policy
17. construct an MTBDD for each auxiliary rule
18. combine MTBDDs and create the CMTBDD,
    invoke the ratification module when Boolean expressions
        of category 3 are encountered
19. combine the CMTBDD with auxiliary rules
```

Figure 3.26. Procedure of CMTBDD Construction

three nodes: $t_1(6 \leq time < 8)$, $t_2(8 \leq time \leq 20)$, $t_3(20 < time \leq 22)$. Correspondingly, P_1 and P_2 are rewritten as:

$$P_1 \begin{cases} B_{permit} = (\ (\ (domain = \text{“}.edu\text{”}) \\ \qquad\qquad \wedge\ (\ (8 \leq time \leq 20 \vee 20 < time \leq 22)\ )\ ) \\ B_{deny} = FALSE \end{cases}$$

$$P_2 \begin{cases} B_{permit} = (\ (\ (domain = \text{“}.edu\text{”} \\ \qquad\qquad \vee affiliation = \text{“}IBM\text{”}) \\ \qquad\qquad \wedge (\ (6 \leq time < 8 \vee 8 \leq time \leq 20)\ )\ ) \\ \qquad\qquad \vee (\ (user = \text{“}Bob\text{”}) \\ \qquad\qquad \wedge (upload + download < 1)\ ) \\ B_{deny} = FALSE \end{cases}$$

An auxiliary rule is associated with the variable “time”, which is expressed as follows.

$(t_1 \wedge \neg t_2 \wedge \neg t_3) \vee (\neg t_1 \wedge t_2 \wedge \neg t_3) \vee (\neg t_1 \wedge \neg t_2 \wedge t_3)$

Variables “upload” and “download” appear in a linear function. The ratification module checks its satisfiability and then inform the MTBDD module to construct the terminal “CP” for it.

By taking the unified nodes and new Boolean formulae as inputs, the MTBDD module first constructs the MTBDD for each policy and auxiliary rules as shown in Figure 3.23. Notice the difference between the MTBDDs of P_2 in Figure 3.27 and Figure 3.23 where node t in Figure 3.23 is split into nodes $t1$ and $t2$ in Figure 3.27. Then these MTBDDs are combined into one CMTBDD. In the following subsection, we show how the CMTBDD is used to execute policy analysis queries.

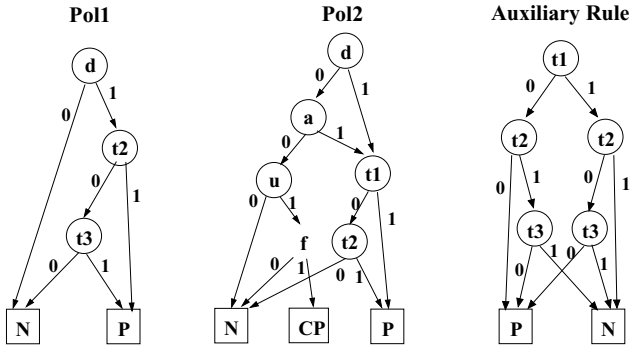


Figure 3.27. MTBDD of policies P_1 and P_2 and the auxiliary rule

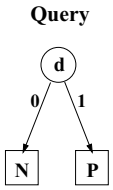


Figure 3.28. Query MTBDD

3.2.4 Query Processing Strategy

Policy analysis queries are carried out based on the MTBDDs and CMTBDDs. For the same set of policies, we only need to construct their MTBDDs and CMTBDDs once and store them in the policy repository for the query processing. In what follows, we propose a generic query processing algorithm that applies to all types of queries on both single and multiple policies. Note that the technique used for queries on a single policy is a special case of the technique used for queries on multiple policies; thus we only discuss queries on multiple policies in the following.

Recall that each query has three types of constraints, $\mathcal{B}_q$, e_q and f_q, where $\mathcal{B}_q$ is a Boolean expression on $Attr_q$, e_q is the desired effect and f_q is a constraint on a set of requests. The query algorithm consists of three steps. The first step preprocesses the query, the second step constructs the query MTBDD and performs model checking, and the final step performs some post-processing.

In particular, for a given query, first we normalize its $\mathcal{B}_q$, map the specified ranges of attributes to the existing unified nodes, and represent the specified ranges as corresponding unified nodes. Then, we construct the query MTBDD. Here, we can treat the normalized $\mathcal{B}_q$ and effect e_q in a query as a rule, and then construct the MTBDD for it. With reference to Example 1, a query like *find the time interval when the user from domain ".edu" can access the data* can be translated as "given Domain = ".edu", Decision = *permit*, find all possible requests". Figure 3.28 shows the query MTBDD.

After we obtain the query MTBDD, we combine it with the MTBDD or CMTBDD of the policies being queried, where we obtain a temporary structure – Query CMTBDD.

By using model checking on the Query CMTBDD, we are now able to find the requests satisfying the $\mathcal{A}_q$ and e_q. As for the example query, we just need to find all paths in the Query CMTBDD which leads to the terminal named "P-P". For conditional decisions, the nodes along the path may need to be examined by plugging the specific variable values.

As for the policy queries with an empty set of $\mathcal{B}_q$, such as the policy relationship evaluation queries, the processing is even simpler. We only need to check the terminals of the CMTBDD. For example, to check if two policies are equivalent, we check whether there exist only three terminals containing "P-P", "D-D" and "N-N", which means two policies always yield same effects for incoming requests. In contrast, if we want to know if two policies totally conflict with one another, we only need to check there does not exist any terminal containing same decisions from two policies, e.g., terminals of "P-P", "D-D", "CP-P". Yet another case is that when terminals like "P-P" and "P-D" both exist which indicates that two policies may yield same decisions sometimes but not always.

Finally, a post-processing may be required if there are constraints specified by f_q. This step is straightforward since we only need to execute some simple examinations on the requests obtained from the previous step. The results will then be collected and organized by the result analyzer before being presented to the user.

3.2.5 Experimental Evaluation

The PSA module has been implemented in Java. An implementation of the modified simplex algorithm [13] has been used for processing Boolean expressions with real value linear constraints. The modified CUDD library developed in [5] has been used for the MTBDD module. In order to test our implementation, we generated XACML policies in which each rule contained randomly generated atomic Boolean expressions of the first three types introduced in Section 4 [5], and then concatenated them with the operator *and* or *or*. The atomic Boolean expression (ABE for short) usually contains a pair of attribute name and value except for the atomic linear inequality function which has multiple attributes.

[5]We have not fully tested and incorporated the automata technique for processing regular expression constraints in the current prototype.

The attributes in each atomic Boolean expression were randomly selected from a predefined attribute set. We performed policy similarity analysis between pairs of generated XACML policies with varying number of rules and ABEs. The experiments were conducted on a Intel Pentium4 CPU 3.00GHz machine with 512 MB RAM.

The performance of our policy similarity analyzer is determined by the three main modules: the ratification module which preprocesses the policies, the MTBDD module which constructs the policy MTBDDs and the CMTBDD and the result analyzer which queries the CMTBDD. In what follows, we first evaluate the preprocessing time taken by ratification module which is followed by evaluation of the MTBDD and the result analyzer modules.

We report results for two sets *set-1* and *set-2* of policy pairs. In *set-1*, the number of rules in each policy was fixed to 10 and the number of atomic Boolean expressions in each rule was varied between 2 and 10 in increments of 2. In *set-2*, the number of atomic Boolean expressions per rule was fixed to 10 and the number of rules was varied between 2 and 10 in increments of 2. Each of the sets contained 4 different types of policy pairs. The policy pairs denoted by *25*, *50*, *75* and *100* on the x-axis of the graphs, the two policies had 25, 50, 75 and 100 percent of *related* atomic Boolean expressions respectively. We define two atomic Boolean expressions to be *related* if they have the same attribute name but different attribute value and possibly different operator.

10 different pairs of policies of each of the four types of policy pairs were generated for the two sets. Thus each point in the following graphs represents an average of these 10 different pairs of policies.

3.2.5.1 Preprocessing Time

Compared to Margrave [5], our policy similarity analyzer supports more rich classes of policies. The performance difference between the two approaches mainly lies in the preprocessing time taken by the ratification module which analyzes the relationships between atomic Boolean expressions before sending them to the MTBDD module. Therefore, in the first round of experiments, we examined the time consumed in the ratification module.

Figures 3.29 and 3.30 show preprocessing time for *set-1* and *set-2* respectively. Time on y-axis is shown in log scale. The amount of work done by the preprocessing module is directly proportional to the number of related atomic Boolean expressions among the policies since every group of related expressions is analyzed and corresponding auxiliary expressions are generated. Thus, as expected we observe that as the number of related Boolean expressions is increased from 25% to 100% the preprocessing time increases.

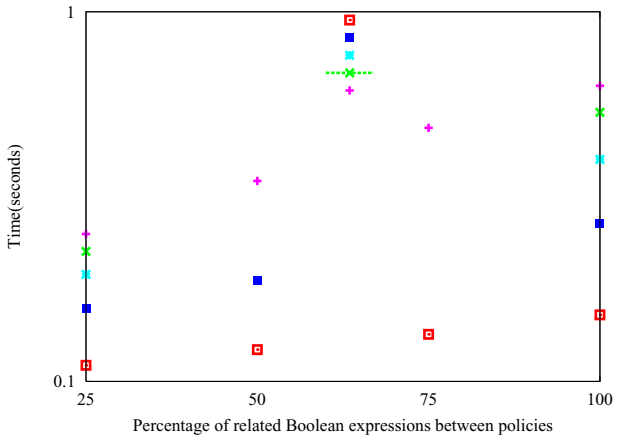


Figure 3.29. Preprocessing Time for *set-1*.

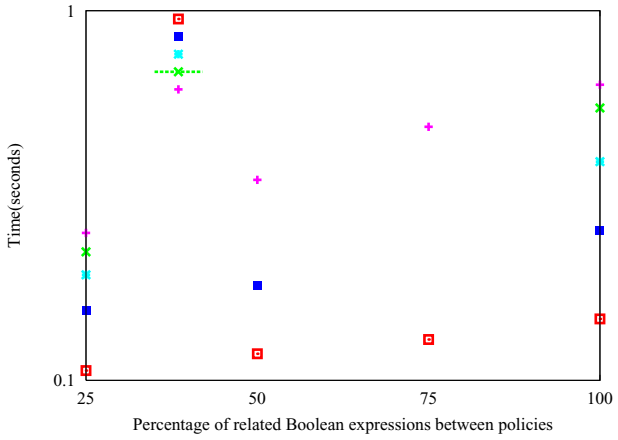


Figure 3.30. Preprocessing Time for *set-2*.

3.2.5.2 MTBDD and CMTBDD Construction Time

Figures 3.31 and 3.32 show the time taken for constructing the MTBDDs of the two policies that are being compared and the CMTBDD in *set-1* and *set-2* respectively. As the number of related Boolean expressions increases more predicates are introduced as a result of preprocessing the two policies being compared. And since the size of the MTBDD constructed for a policy depends on the number of unique predicates in the policy, the increase in number of predicates results in an increase in the size of the policy MTBDD to be constructed which in turn increases the time required to construct the MTBDDs and the CMTBDD. Thus we observe that the time for constructing the MTBDDs and CMTBDDs is larger for policy pairs that have higher percentage of related atomic Boolean expressions.

Although the increase in preprocessing and construction times is not linear with increase in number of rules and number of atomic Boolean expressions, the actual preprocessing and construction times for policies with the largest number of atomic Boolean expressions, i.e., policies with 100 atomic Boolean expressions were 0.63 seconds and 4.3 seconds respectively. Considering that the number of atomic Boolean expressions in policies occurring in practice tends to lie in the range of 20 to 100 as reported in [5] and the CMTBDD once constructed can be used for answering multiple queries, we believe that the performance is acceptable.

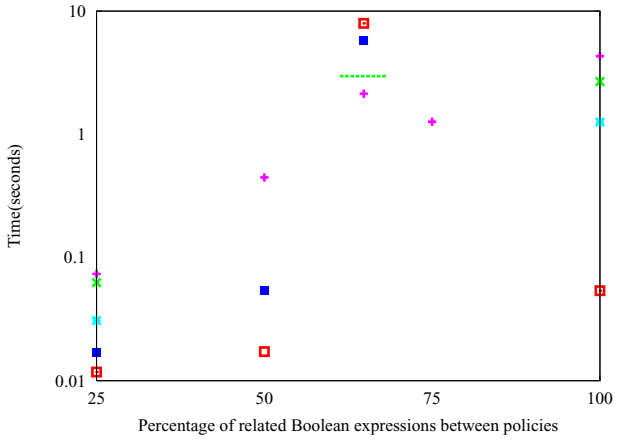


Figure 3.31. MTBDD Construction Time for *set-1*.

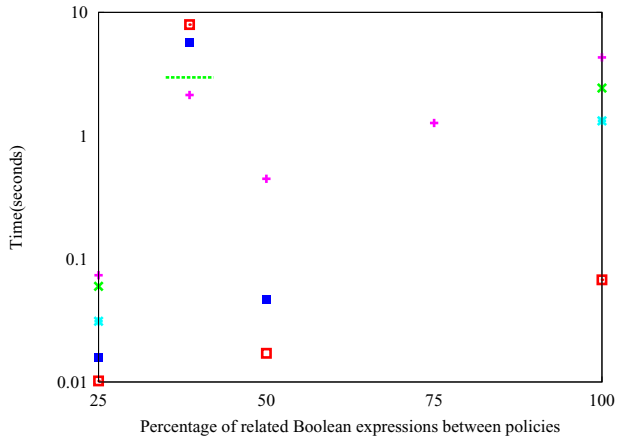


Figure 3.32. MTBDD Construction Time for *set-2*.

3.2.5.3 Query Processing Time

Figures 3.33 and 3.34 show the average time taken for an effect query for policy pairs in *set-1* and *set-2* respectively. When the number of related atomic Boolean expressions between the policies being compared increases there are more number of common nodes when the policy MTBDDs are combined resulting in CMTBDDS with fewer nodes and paths. Thus we observe that for policy pairs with higher percentage of related Boolean expressions the average query time decreases. Notice that the decrease in average query time becomes more pronounced in policy pairs with higher number of rules and atomic Boolean expressions. This means that the result analyzer is efficient for larger policies with larger number of related Boolean expressions. Also, as the number of rules and the number of atomic Boolean expressions are increased the average query time scales well.

3.2.6 Multi-level Grid Visualization

Policy analyses like policy similarity and policy conflict return large sets of requests, characterized by many attributes. The issue of representing such requests has received little attention so far, undermining the usability of the analysis. To facilitate the correct

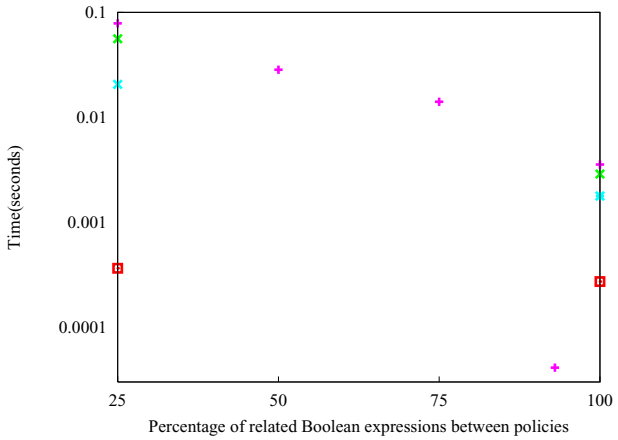


Figure 3.33. Query Processing Time for *set-1*.

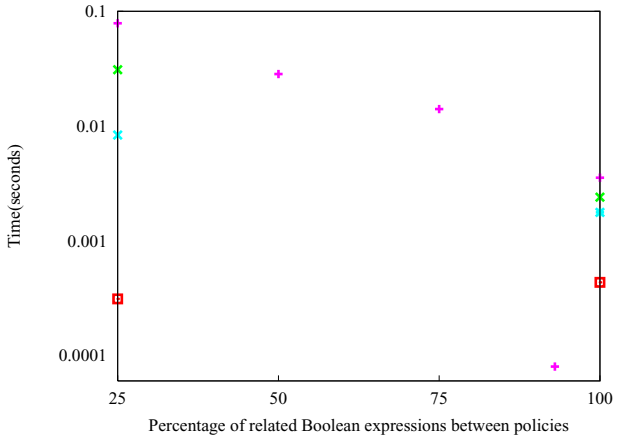


Figure 3.34. Query Processing Time for *set-2*.

interpretation of such results, a concise and intuitive representation method is necessary. An effective visualization tool must address several *design goals*:

- Eliminate redundant information, by representing requests in a simple and concise manner.
- Extract semantic information from policy attribute domains to simplify visualization.
- Provide a generic visual interface that can represent in a uniform fashion the results of several analysis types.

In this thesis, we propose a *multi-level grid* visualization method to represent sets of requests resulting from policy similarity analysis. Access control policies consist of rules that specify *subjects* which can perform particular *actions* on *resources*. Rules are expressed using predicates defined on the domains of attributes that describe subjects, actions and resources. A request authorized by such policies is thus an assignment of values to attributes. Each cell in the grid represents a single request or a group of requests. By looking at the different visual aspects like color, content, and the column and row headers of a cell in the grid, a user can quickly visualize the requests covered by the cell. The grid is also useful for visualizing a policy by itself, and thus can also help in the process of policy authoring.

3.2.6.1 Preliminary Definitions

We consider policies expressed in an *attribute-based* access control language, such as XACML [4]. Denote by $\{A_i\}_{1 \leq i \leq m}$ the set of attributes, and by $\{D_i\}_{1 \leq i \leq m}$ their domains.

Definition 5 (Domain Space Partition) Let each domain D_i be partitioned into a set of n_i disjoint sub-sets $D_{i\,(1 \leq j \leq n_i)}^{j}$, i.e., $D_i = \bigcup_{j \leftarrow 1, n_i} D_i^j$ and $\forall 1 \leq j_1, j_2 \leq n_i | j_1 = j_2$ it holds that $D_i^{j_1} \bigcap D_i^{j_2} = \emptyset$. A *domain space partition* $\mathcal{D}$ is defined as the Cartesian product

$$\mathcal{D} = D_1^1 \times \cdots \times D_1^{n_1} \times \cdots \times D_m^1 \times \cdots \times D_m^{n_m}$$

We consider *minimal* partitions of the attribute domain space, in the sense that all access control policy rules are specified with respect to sub-domains of equal or coarser granularity than the partition granularity.

Example 13 An on-line live-streaming content provider specifies access control policies with respect to two attributes: $A_1 = ConType$ specifies the content type, and $A_2 = Time$ specifies the time of request. The corresponding domains are $D_1 = \{video, audio\}$, partitioned into $D_1^1 = video, D_1^2 = audio$, and $D_2 =$ *[00:00-23:59]* split into time intervals $D_2^1 =$ *[00:00-12:00]*, $D_2^2 =$ *[12:01-18:00]* and $D_2^3 =$ *[18:01-23:59]*.

Definition 6 (Aggregate Request) A *request* is a m-dimensional tuple $\langle v_1, \ldots, v_m \rangle$, which specifies the assignment of value[6] v_i for each attribute A_i. An *aggregate request* is a m-tuple $r \equiv \langle D_1^{j_1}, \ldots, D_m^{j_m} \rangle$, which represents a generalization of all requests in a particular sub-set of attribute domain space partition $\mathcal{D}$.

Informally, an aggregate request allows ranges of attribute values to be specified instead of exact values, subject to the granularity restriction mentioned earlier. The advantage of aggregate requests is that they allow concise representation of a larger (possibly infinite, if attribute domains are continuous) set of requests. Note that, each sub-set D_i^j can be associated with a Boolean predicate, characterizing whether a certain value belongs to D_i^j or not. For instance, we can associate D_1^1 with predicate $p_1 \equiv$ "$ConType = video$" and D_2^3 with $p_2 \equiv$ "$18:01 \leq time \leq 23:59$". Then, an aggregate request for downloading video in the evening hours can be expressed as conjunction $p_1 \wedge p_2$. Based on domain space partition $\mathcal{D}$, we obtain a set of predicates $\mathcal{P} = \{p_1, \ldots, p_n\}$ (one for each D_i^j), where $n = \sum_{i=1}^{m} n_i$. We further consider only aggregate requests, specified as conjunctions of predicates in $\mathcal{P}$.

3.2.6.2 Multi-Level Grid

Consider set R of aggregate requests, which is the result of one of the policy analysis types discussed earlier (e.g., the set of all requests permitted by a certain policy). We aim to represent R in the n-dimensional space of predicate set $\mathcal{P}$. Since the number of dimensions that can be visually represented is limited, we construct projections on pairs of predicates. Specifically, we use a two-dimensional grid (or matrix) representation, where rows and columns represent predicates. The entry (p_i, p_j) corresponds to the projection of $\mathcal{P}$ on predicates p_i and p_j. Since projection is order-independent, the resulting matrix is symmetric, and we only represent entries above the main diagonal. Furthermore, we require the predicates in the projection to be distinct, hence we omit the first column (i.e., p_1) and the last row (i.e., p_n) resulting in a grid with $(n-1) \times (n-1)$ cells.

[6]We consider that each value is a singleton. However, our technique can accommodate multiple values for an attribute, if the partition $\mathcal{D}$ is defined on the power-sets of attribute domains

The visualization grid is built in two stages: *request hashing* and *dependence list minimization*.

Request Hashing. Each request $r \in R$ is hashed into all grid cells (p_i, p_j) such that p_i, p_j belong to the predicate list of r. For each such cell, a *dependence list* is maintained, which is the conjunction of all predicates contained in r, other than p_i and p_j. The dependence list is non-empty for all requests that are expressed as a conjunction of three or more predicates.

Example 14 Consider requests r_1, r_2, r_3 shown in Figure 3.35(a). Request r_1, which is the conjunction of predicates p_1, p_2 and p_3, is hashed into cell (p_1, p_2) with dependence list $\{p_3\}$ (Figure 3.35(b)). Note that, r_1 also hashes into cells (p_1, p_3) and (p_2, p_3) (i.e., the other combinations of two predicates out of set $\{p_1, p_2, p_3\}$). However, for clarity of presentation, we omit the contents of other cells. Similarly, request r_2 hashes into cell (p_1, p_2) with dependence list $\{p_4\}$. Request r_2 also hashes into cell (p_2, p_4) with dependence list $\{p_1\}$, etc.

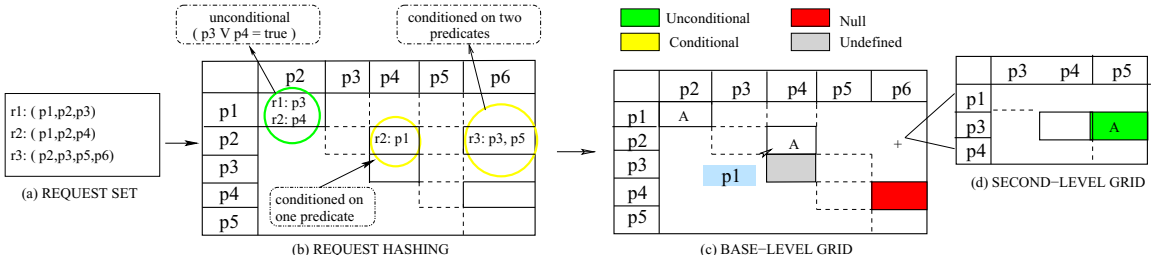


Figure 3.35. Multi-level Grid Visualization of Policy Analysis Results.

When the hashing is complete (Figure 3.35(b)), the dependence list of each cell is a disjunctive normal form (DNF), i.e., a disjunction of dependence lists of all hashed requests.

Dependence List Minimization. To obtain a concise representation of requests, the DNF of each cell is further subjected to binary minimization, with the aim of eliminating redundant dependencies. The binary minimization can be performed based on two criteria: *(i)* factoring common predicates of distinct requests and *(ii)* using semantic relationships among predicates. For the former case, we use the *ESPRESSO* Boolean minimization

tool [26], which minimizes the DNF of each cell. The following example illustrates the latter case of using attribute semantics.

Example 15 Let p_3 and p_4 represent predicates corresponding to the *time* attribute with $p_3 \equiv$ "00:00 <= time < 12:00" and $p_4 \equiv$ "12:00 <= time < 23:59". Let $p_1 \equiv$ "$user = Bob$" and $p_2 \equiv$ "$ConType = video$". The dependence lists of r_1 and r_2 in cell (p_1, p_2) (Figure 3.35(b)) can be minimized, since the two time intervals are disjoint, and cover the entire time range, Hence, $p_3 \vee p_4 = True$. By using such semantic information, we eliminate the dependencies in the cell (p_1, p_2).

Following the minimization phase, the grid representation shown in Figure 3.35(c) is obtained. Note that, also due to attribute semantics, not all cells correspond to valid predicate combinations. For instance, predicates p_3 and p_4 in the previous example correspond to disjoint time intervals, and no request can be characterized by both predicates. We refer to such cells as *undefined*, and we represent them in grey color.

One of the main challenges of concise visualization is dealing with the large number of predicates. Since high-dimensional functions are difficult to represent graphically, we employ alternative means to specify certain predicates. Specifically, *action* is a type of attribute that appears in every request. The set of actions is typically restricted to a few values (e.g., *read*, *write*, etc). Similar to [27], we represent *action* as a special predicate, which is not associated with grid row/columns. Instead, each grid cell is split into a number of sub-cells, one for each action type, and an abbreviation of the action name (e.g., W for *write*, D for *download*, etc) identifies each sub-cell . The hashing and minimization phases are performed independently with respect to each action type. In Figure 3.35(c) there is a single action type, abbreviated as *"A"*.

The content of each grid cell (p_i, p_j) is decided as follows:

Case 1. No requests were hashed into cell (p_i, p_j). This means that the attribute domain sub-space corresponding to p_i and p_j does not contain any requests in R. We refer to such cells as *Null* cells, and represent them with *red* background. For example, in Figure 3.35(c), none of the requests depend on the predicates p_4 and p_6, therefore (p_4, p_6) is a $Null$ cell.

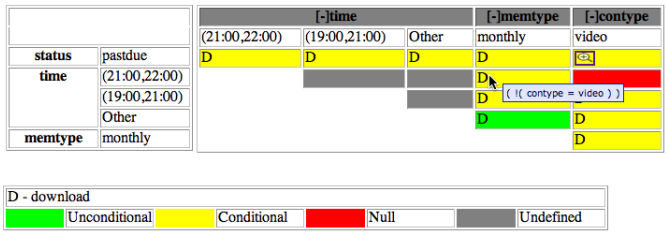
Case 2. At least one request was hashed into cell (p_i, p_j), and after minimization the cell's dependence list is empty. This means that at least one request in R belongs to the domain sub-space characterized by p_i and p_j, regardless of whether the request contains other predicates or not. Such cells are called *unconditional*, and are represented with *green* background. Cell (p_1, p_2) in Figure 3.35(c) is an unconditional cell.

Case 3. At least one request was hashed into cell (p_i, p_j), and after the minimization phase the dependence list contains exactly one predicate p_k. This case signifies that at least one request belongs to the domain sub-space characterized by p_i and p_j, subject to restrictions on the value of p_k. We call such cells *conditional*, and we represent them using *yellow* background. In addition, a *tool-tip* displays the additional restriction on p_k, when a user moves the cursor over the cell. In Figure 3.35(c), request r_2 that is hashed into cell (p_2, p_4) is conditioned by predicate p_1. The tool-tip displays the additional predicate p_1.

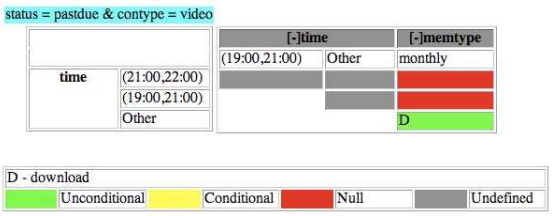
Case 4. This case is similar to Case 3, except that the dependence list includes at least two predicates. Such cells are also called conditional, and represented with yellow background. However, a tool-tip cannot completely express the dependence condition. Instead, such cells contain a *zoom-in hyperlink*. Clicking this hyperlink opens a new grid with all predicates except p_i and p_j. The hashing and minimization phases are applied recursively on the reduced grid, but only with respect to requests hashed in cell (p_i, p_j). In the worst case, requests specified by all n predicates require at most $\lfloor n/2 \rfloor$ grid levels. In practice, however, considerably fewer grid levels are necessary. In Figure 3.35(b), request r_3 is hashed into cell (p_2, p_6), with dependence on p_3 and p_5. This cell is expanded into a second-level grid in Figure 3.35(d). In this case, at the second level there is no conditional cell.

3.2.6.3 Case Study

Consider the example of a data owner who wishes to outsource the data to a storage service provider (SP). Both the owner and the SP have their own access policies, and a data request will only be allowed if it is permitted by both policies. Therefore, the data owner wants to find an SP with a similar access control policy.



(a) Base-level Grid



(b) Second-level Grid

Figure 3.36. Visualizing Results of Policy Similarity Analysis.

The policy P_1 of the data owner *permits* download access to monthly subscribers but *denies* downloading of video files between $19:00$ and $22:00$ to subscribers whose status is past due. The policy P_2 of the SP *permits* all downloads, but also *denies* downloading of video files between $21:00$ and $22:00$, due to maintenance. Both P_1 and P_2 use the *deny-override* rule combining algorithm. The full specification of P_1 and P_2 is the following:

$$\mathbf{P_1}:$$
$$B_{permit} = (\ (memtype = \text{“}monthly\text{”} \wedge$$
$$(action = \text{“}download\text{”})\ )$$
$$B_{deny} = (\ (status = \text{“}pastdue\text{”}) \wedge (action = \text{“}download\text{”})$$
$$\wedge (contype = \text{“}video\text{”} \wedge 19:00 < time < 22:00)\ )$$

$$\mathbf{P_2}:$$
$$B_{permit} = (action = \text{“}download\text{”})$$
$$B_{deny} = (\ (action = \text{“}download\text{”})$$
$$\wedge (contype = \text{“}video\text{”} \wedge 21:00 < time < 22:00)\ )$$

The objective is to find the set of requests R which are the results of the PSA query *"Find all requests that have effect* **Permit** *in policy* P_1 *and effect* **Permit** *in policy* P_2*."* Figure 3.36(a) shows the grid visualization of set R. The abbreviation *"D"* in each cell stands for *download* action. The cells corresponding to combinations of disjoint time intervals are not valid predicate combinations, and are shown in *grey*. The *red* cell ("21:00 $<$ time $<$ 22:00", "contype $=$ video") indicates that no requests issued between $21:00$ and $22:00$ are permitted by both policies.

Both policies permit download for monthly subscribers (*"memtype = monthly"*) if the request is in time interval *"Other"*, i.e., outside time interval $19:00$ to $22:00$. The corresponding cell is *green*, meaning that the effect is independent of any other predicates. All other cells are conditional, meaning that they do contain some requests that are permitted by both policies, but only subject to predicates other than the ones corresponding to the cells. For instance, a monthly subscriber is permitted download access between $21:00$ and $22:00$ only if the content type is not video. This single condition is represented by the tooltip in the cell ("21:00 $<$ time $<$ 22:00", "memtype $=$ monthly").

Finally, the cell ("status $=$ pastdue", "contype $=$ video") has a dependence on more than two additional predicates. Note that, a zoom-in hyperlink is represented inside the cell, to mark the fact that the dependence cannot be represented using a tool-tip. When a user clicks on the hyperlink, a second-level grid is opened in a new window, shown in Figure 3.36(b). The highlighted blue banner at the top left of the window displays the fixed predicates ("status $=$ pastdue", "contype $=$ video") from the previous grid level. Within this fixed sub-space of the attribute domain, both policies permit download access to monthly subscribers at all times outside the $19:00$ to $21:00$ interval. This is indicated by the *green* cell ("time $=$ Other", "memtype $=$ monthly"). The two *red* cells ("21:00 $<$ time $<$ 22:00", "memtype $=$ monthly") and ("19:00 $<$ time $<$ 21:00", "memtype $=$ monthly") signify that access is never permitted by either policy within the $19:00$ to $22:00$ time interval.

4 POLICY INTEGRATION FRAMEWORK

Many distributed applications such as dynamic coalitions and virtual organizations need to integrate and share resources, and these integration and sharing will require the integration of access control policies. In order to define a common policy for resources jointly owned by multiple parties applications may be required to integrate policies from different sources into a single policy. Even in a single organization, there could be multiple policy authoring units. If two different branches of an organization have different or even conflicting access control policies, what policy should the organization as a whole adopt? If one policy allows the access to certain resources, but another policy denies such access, how can they be composed into a coherent whole? Approaches to policy integration are also crucial when dealing with large information systems. In such cases, the development of integrated policies may be the product of a bottom-up process under which policy requirements are elicited from different sectors of the organization, formalized in some access control language, and then integrated into a global access control policy.

When dealing with policy integration, it is well known that no single integration strategy works for every possible situation, and the exact strategy to adopt depends on the requirements by the applications and the involved parties. An effective policy integration mechanism should thus be able to support a flexible fine-grained policy integration strategy capable of handling complex integration specifications. Some relevant characteristics of such an integration strategy are as follows. First, it should be able to support 3-valued policies. A 3-valued policy may allow a request, deny a request, or not make a decision about the request. In this case we say the policy is not applicable to the request. Three-valued policies are necessary for combining partially specified policies, which are very likely to occur in scenarios that need policy integration. When two organizations are merging and need policy integration, it is very likely that the organizations are unaware or might not have jurisdiction over each other resources, and thus a policy in one organization may be

"NotApplicable" to requests about resources in the other organization. Second, it should allow one to specify the behavior of the integrated policy at the granularity of requests and effects. In other words, one should be able to explicitly characterize a set of requests that need to be permitted or denied by the integrated policy. For example, users may require the integrated policy to satisfy the condition that for accesses to an object O_i policy P_1 has the precedence, whereas for accesses to an object O_j, policy P_2 has precedence. Third, it should be able to handle *domain constraints* requiring the integrated policy to be applied to a restricted domain instead of the original domain. And fourth, it should be able to support policies expressed in rich policy languages, such as XACML with features like policy combining algorithms.

The problem of policy integration has been investigated in previous works. The concept of policy composition under constraints was first introduced by Bonatti et al. [28]. They proposed an algebra for composing access control policies and use logic programming and partial evaluation techniques for evaluating algebra expressions. Another relevant approach is by Wijesekera et al. [29] who proposed a propositional framework for composing access control policies. Those approaches have however a number of shortcomings. They support only limited forms of compositions. For example, they are unable to support compositions that take into account policy effects or policy jumps (i.e., if P_1 permits, let P_2 makes decision, otherwise P_3 makes decision). They only model policies with two decision values, either "Permit" or "Deny". It is not clear the scope or expressive power of their languages since they do not have any notion of completeness. They do not provide an actual methodology or an implementation for generating the integrated policies. Neither work relates their formalisms to any language used in practice.

In this thesis we propose a framework for the integration of access control policies that addresses the above shortcomings. The overall organization of our integration framework is outlined in Figure 4.1. The core of our framework is the *Fine-grained Integration Algebra* (FIA). Given a set of input policies P_1, P_2, $\cdots$, P_n, one is able to specify the integration requirements for these input policies through a FIA expression, denoted as $f(P_1, P_2, \cdots, P_n)$ in Figure 4.1. The FIA expression is then processed by the other components of the frame-

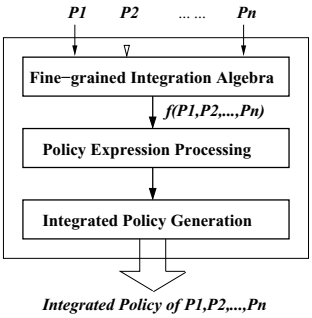


Figure 4.1. Policy integration

work in order to generate the integrated policy. We demonstrate the effectiveness of our framework through an implementation that supports the integration of XACML policies. We choose XACML because of its widespread adoption and its many features, such as attribute-based access control and 3-valued policy evaluation. We use Multi-Terminal Binary Decision Diagrams (MTBDD) [24] for representing policies and generating the integrated policies in XACML syntax. With the aid of the implementation of algebra operators, users can now easily specify their integration requirements by an expression and do not need to write a new complex policy combining algorithm by themselves.

4.1 An Illustrative Example

We now introduce an example of XACML policies that will be used throughout this chapter.

Example 16 *Consider a company with two departments D_1 and D_2. Each department has its own access control policies for the data under its control. Assume that P_1 and P_2 are the access control policies of D_1 and D_2 respectively. P_1 contains two rules, $P_1.Rul_{11}$ and $P_1.Rul_{12}$. $P_1.Rul_{11}$ states that the manager is allowed to read and update any data in the time interval [8am, 6pm]. $P_1.Rul_{12}$ states that any other staff is not allowed to read. P_2 also contains two rules, $P_2.Rul_{21}$ and $P_2.Rul_{22}$. $P_2.Rul_{21}$ states that the manager and staff can read any data in the time interval [8am, 8pm], and $P_2.Rul_{22}$ states that the staff cannot perform any update action. For simplicity, we adopt the following succinct representation*

in most discussion, where "role", "act" and "time" are attributes representing information on role, action and time, respectively.

$P_1.Rul_{11}$: *role=manager, act=read or update,*
time= [8am, 6pm], effect= Permit.

$P_1.Rul_{12}$: *role=staff, act=read, effect = Deny.*

$P_2.Rul_{21}$: *role=manager or staff, act=read,*
time = [8am, 8pm], effect = Permit.

$P_2.Rul_{22}$: *role=staff, act=update, effect = Deny.*

4.2 Policy Semantics

Before we introduce our algebra we need to find a suitable definition for a *policy*. We propose a simple yet powerful definition for a policy according to which a policy is defined by the set of requests that are permitted by the policy and the set of requests that are denied by the policy. This simple notion will provide us with a precise characterization of the meaning of policy integration in terms of the sets of permitted and denied requests. In the rest of this chapter, we use Y, N and NA to denote the "Permit", "Deny" and "NotApplicable" decisions respectively.

In our work, we assume the existence of a vocabulary Σ of attribute names and domains. Each attribute, characterizing a subject or an object or the environment, has a name a and a domain, denoted by $dom(a)$, in Σ. The following two definitions introduce the notion of access request (request, for short) and policy semantics.

Definition 4.2.1 *Let a_1, a_2, ..., a_k be a set of attribute names and let $v_i \in dom(a_i)$ ($1 \leq i \leq k$) in the vocabulary Σ. $r \equiv \{(a_1, v_1), (a_2, v_2), \cdots, (a_k, v_k)\}$ is a request over Σ. The set of all requests over Σ is denoted as R_Σ.*

Example 17 *Consider policy P_1 from Example 16. An example of request to which this policy applies is that of a manager wishing to read any resource at 10am. According to Definition 1, such request can be expressed as $r \equiv$* {(role, manager), (act, read), (time, 10am)}.

Definition 4.2.2 *A 3-valued access control policy P is a function mapping each request to a value in $\{Y, N, NA\}$. R^P_Y, R^P_N and R^P_{NA} denote the set of requests permitted, denied and not applicable by the policy P respectively, and $R_\Sigma = R^P_Y \cup R^P_N \cup R^P_{NA}$, $R^P_Y \bigcap R^P_N = \emptyset$, $R^P_Y \bigcap R^P_{NA} = \emptyset$, $R^P_N \bigcap R^P_{NA} = \emptyset$. We define a policy P as a triple $\langle R^P_Y, R^P_N, R^P_{NA}\rangle$.*

Our approach to formulating the definition of a policy is independent of the language in which access control policies are expressed. Therefore, our approach can be applied to languages other than XACML.

4.3 A Fine-grained Integration Algebra

The Fine-grained Integration Algebra (FIA) is given by $\langle \Sigma, \mathsf{P_Y}, \mathsf{P_N}, +, \&, \neg, \Pi_{dc}\rangle$, where Σ is the vocabulary of attribute names and their domains, $\mathsf{P_Y}$ and $\mathsf{P_N}$ are two policy constants, $+$ and $\&$ are two binary operators, and $\neg$ and Π_{dc} are two unary operators.

4.3.1 Policy Constants and Operators in FIA

We now describe the policy constants and operators in FIA. In what follows, $P_1 \equiv \langle R^{P_1}_Y, R^{P_1}_N, R^{P_1}_{NA}\rangle$ and $P_2 \equiv \langle R^{P_2}_Y, R^{P_2}_N, R^{P_2}_{NA}\rangle$ denote two policies to be combined, and $P_I \equiv \langle R^{P_I}_Y, R^{P_I}_N, R^{P_I}_{NA}\rangle$ denotes the policy obtained from the combination. Operators on policies are described as set operations.

Permit policy $(\mathsf{P_Y})$. $\mathsf{P_Y}$ is a policy constant that permits everything. Thus $\mathsf{P_Y} \equiv \langle R_\Sigma, \emptyset, \emptyset\rangle$

Deny policy $(\mathsf{P_N})$. $\mathsf{P_N}$ is a policy constant that denies everything. Thus $\mathsf{P_N} \equiv \langle \emptyset, R_\Sigma, \emptyset\rangle$

Addition (+). Addition of policies P_1 and P_2 results in a combined policy P_I in which requests that are permitted by either P_1 or P_2 are permitted, requests that are denied by one policy and are not permitted by the other are denied. More precisely:

$$P_I = P_1 + P_2 \Longleftrightarrow \begin{cases} R^{P_I}_Y = R^{P_1}_Y \cup R^{P_2}_Y \\ R^{P_I}_N = (R^{P_1}_N \backslash R^{P_2}_Y) \cup (R^{P_2}_N \backslash R^{P_1}_Y) \end{cases}$$

A binary operator can be viewed as a function that maps a pair of values $\{Y, N, NA\}$ to one value. We give this view of addition, intersection, and two other derived binary operators to be introduced later in Table 4.1. A binary operator is represented using a matrix that illustrates the effect of integration for a given request r. The first column of each matrix denotes the effect of P_1 with respect to r and the first row denotes the effect of P_2 with respect to r.

Table 4.1
Policy combination matrix of operator $+$, $\&$, $-$, $\triangleright$

$P_1 + P_2$

P_1 \ P_2	Y	N	NA
Y	Y	Y	Y
N	Y	N	N
NA	Y	N	NA

$P_1 \,\&\, P_2$

P_1 \ P_2	Y	N	NA
Y	Y	NA	NA
N	NA	N	NA
NA	NA	NA	NA

$P_1 - P_2$

P_1 \ P_2	Y	N	NA
Y	NA	NA	Y
N	NA	NA	N
NA	NA	NA	NA

$P_1 \triangleright P_2$

P_1 \ P_2	Y	N	NA
Y	Y	Y	Y
N	N	N	N
NA	Y	N	NA

Intersection ($\&$). Given two policies P_1 and P_2, the intersection operator returns a policy P_I which is applicable to all requests having the same decisions from P_1 and P_2. More precisely,

$$P_I = P_1 \,\&\, P_2 \Longleftrightarrow \begin{cases} R_Y^{P_I} = R_Y^{P_1} \cap R_Y^{P_2} \\ R_N^{P_I} = R_N^{P_1} \cap R_N^{P_2} \end{cases}$$

The intersection operator can be viewed as taking minimum on the information order.

The integrated policy makes a decision only when the two policies agree.

The intersection operator is useful in situations in which consensus is required for making an allow or deny decision.

Example 18 *Consider Example 16. The result of adding two policies will give more authorization to both manager and staff. The integrated policy P_I may contain the following four rules, from which we can see that now the staff from department D_1 are also allowed to read customer information.*

$P_I.Rul_{I1}$: role=manager or staff, act=read,

time= [8am, 8pm], effect= Permit.

$P_I.Rul_{I2}$: *role=manager, act=update,*

time= [8am,6pm], effect = Permit.

$P_I.Rul_{I3}$: *role=staff, act = read,*

time = [8am, 8pm], effect = Deny.

$P_I.Rul_{I4}$: *role=staff, act=update, effect = Deny.*

Negation ($\neg$). Given a policy P, $\neg P$ returns a policy P_I, which permits (denies) all requests denied (permitted) by P. The negation operator does not affect those requests that are not applicable to the policy. More precisely:

$$P_I = \neg P \Longleftrightarrow \begin{cases} R_Y^{P_I} = R_N^P \\ R_N^{P_I} = R_Y^P \end{cases}$$

Table 4.2 summarizes all operators.

Table 4.2
Basic Policy Operators

Operator	Meaning	Format
$+$	Addition	$P_1 + P_2$
$\&$	Intersection	$P_1 \,\&\, P_2$
$-$	Subtraction	$P_1 - P_2$
$\neg$	Negation	$\neg P$
Π	Projection	$\Pi_{dc,ec}(P)$

Domain projection (Π_{dc}) The domain projection operator takes as parameter the domain constraint dc and restricts a policy to the set of requests identified by dc.

Definition 4.3.1 *A domain constraint dc takes the form $\{(a_1, range_1), (a_2, range_2), \cdots, (a_k, range_k)\}$[1], where a_1, a_2, ..., a_k are attribute names, and $range_i (1 \leq i \leq k)$ is a set of values in* dom*(a_i). Given a request $r = \{(a_{r_1}, v_{r_1}), \cdots, (a_{r_m}, v_{r_m})\}$, we say that r satisfies dc if the following condition holds: for each $(a_{r_j}, v_{r_j}) \in r$ $(1 \leq j \leq m)$ there exists $(a_i, range_i) \in dc$, such that $a_{r_j} = a_i$ and $v_{r_j} \in range_i$.*

[1] In case of an ordered domain, these sets can be represented by ranges.

The semantics of $\Pi_{dc}(P)$ is given by

$$P_I = \Pi_{dc}(P) \Longleftrightarrow \begin{cases} R_Y^{P_I} = \{r | r \in R_Y^P \ \ and \ \ r \ \ satisfies \ \ dc\} \\ R_N^{P_I} = \{r | r \in R_N^P \ \ and \ \ r \ \ satisfies \ \ dc\} \end{cases}$$

4.3.2 FIA Expressions

The integration of policies may involve multiple operators, and hence we introduce the concept of FIA expressions.

Definition 4.3.2 *A FIA expression is recursively defined as follows:*
- If P is policy, then P is a FIA expression.
- If f_1 and f_2 are FIA expressions so are $(f_1) + (f_2)$, $(f_1) \,\&\, (f_2)$, and $\neg(f_1)$.
- If f is a FIA expression and dc is a domain constraint then $\Pi_{dc}(f)$ is a FIA expression.

In what follows we will use the terms "policy" and "expression" synonymously. In FIA expressions, the binary operators are viewed as left associative and unary operators are right associative. The precedence are $\neg$ and Π_{dc} together have the highest precedence, followed by $\&$, and then by $+$. For example, $P_1 + \Pi_{dc} P_2 + \neg P_3 \,\&\, P_4$ is interpreted as $((P_1 + (\Pi_{dc} P_2)) + ((\neg P_3) \,\&\, P_4)$. FIA has algebraic properties including commutativity, associativity, absorption, distributivity, complement, idempotence, boundedness and involution.

Theorem 4.3.1 Let P_1, P_2 and P_3 denote policies. FIA has the following algebraic properties.

Proof

- **Commutativity:**

  $P_1 + P_2 = P_2 + P_1$;

  $P_1 \,\&\, P_2 = P_2 \,\&\, P_1$;

- **Associativity:**

  $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$;

  $(P_1 \,\&\, P_2) \,\&\, P_3 = P_1 \,\&\, (P_2 \,\&\, P_3)$;

- **Absorption:**

  $P_1 + (P_1 \,\&\, P_2) = P_1$;

  $P_1 \,\&\, (P_1 + P_2) = P_1$;

- **Distributivity:**

  $P_1 + (P_2 \,\&\, P_3) = (P_1 + P_2) \,\&\, (P_1 + P_3)$;

  $P_1 \,\&\, (P_2 + P_3) = (P_1 \,\&\, P_2) + (P_1 \,\&\, P_3)$;

  $\Pi_{dc}(P_1 + P_2) = (\Pi_{dc} P_1) + (\Pi_{dc} P_2)$;

  $\Pi_{dc}(P_1 \,\&\, P_2) = (\Pi_{dc} P_1) \,\&\, (\Pi_{dc} P_2)$

- **Complement:** $\mathsf{P_Y} = \neg \mathsf{P_N}$; $\quad \mathsf{P_N} = \neg \mathsf{P_Y}$;
- **Idempotence:** $P_1 + P_1 = P_1$; $\quad P_1 \,\&\, P_1 = P_1$;
- **Boundedness:** $P_1 + \mathsf{P_Y} = \mathsf{P_Y}$;
- **Involution:** $\neg(\neg P_1) = P_1$.

■

4.3.3 Derived Operators

In this section, we introduce some commonly used operators. They are defined using the core operators.

Not-applicable policy ($\mathsf{P_{NA}}$). $\mathsf{P_{NA}}$ is a policy constant that is not applicable for every request. Since the $\&$ operator applies only to requests that have common effects and $\mathsf{P_Y}$

and $\mathsf{P_N}$ have no requests with common effects, $\mathsf{P_Y} \,\&\, \mathsf{P_N}$ yields a policy that is not applicable for every request. Thus $\mathsf{P_{NA}}$ can be defined as $\mathsf{P_Y} \,\&\, \mathsf{P_N}$.

Effect projection (Π_Y and Π_N). $\Pi_Y(P)$ restricts the policy P to the requests allowed by it. It is defined as: $\Pi_Y(P) = P \,\&\, \mathsf{P_Y}$. Similarly, $\Pi_N(P)$ restricts the policy P to the requests denied by it; it is defined as $\Pi_N(P) = P \,\&\, \mathsf{P_N}$. We are overloading Π to denote both effect projection and domain projection; the meaning should be clear from the subscript.

Subtraction ($-$). Given two policies P_1 and P_2, the subtraction operator returns a policy P_I which is obtained by starting from P_1 and limiting the requests that the integrated policy applies only to those that P_2 does not apply to. The subtraction operator is defined as:

$$P_1 - P_2 = (\mathsf{P_Y} \,\&\, (\neg(\neg P_1 + P_2 + \neg P_2))) + \\ (\mathsf{P_N} \,\&\, (P_1 + P_2 + \neg P_2)) \,.$$

To see why this is correct, observe that $\neg P_1 + P_2 + \neg P_2$ will deny a request if and only if P_1 allows it and P_2 gives NA for it. Thus $\mathsf{P_Y} \,\&\, (\neg(\neg P_1 + P_2 + \neg P_2))$ allows a request if and only if P_1 allows it and P_2 gives NA it, and is not applicable for all other requests. Similarly, $\mathsf{P_N} \,\&\, (P_1 + P_2 + \neg P_2)$ denies a request if and only if P_1 denies it and P_2 gives NA for it.

An example is when controlling access to an important document, owned by multiple parties; in such case the access is granted only when all the relevant access control policies of these parties agree.

Consider Example 1 again, the result of intersecting two policies is the following.

Example 19 *($P_I = P_1 \,\&\, P_2$).*

$P_I.Rul_{I1}$: role=manager, act=read,

time=[8am, 6pm], effect = Permit.

Precedence ($\triangleright$). Given two policies P_1 and P_2, the precedence operator returns a policy P_I which yields the same decision as P_1 for any request applicable to P_1, and yields the same decisions as P_2 for the remaining requests. The precedence operator can be expressed

as $P_1 + (P_2 - P_1)$. By limiting P_2 to requests that P_1 does not decide, this operator can be used as a building block for resolving possible conflicts between two policies.

Example 20 Policies P_1 and P_2 in Example 1 grant different authorizations to the manager and staff, which means that some requests may receive different decisions from two policies. The integrated policy $P_I = P_1 - P_2$ removes possible conflicts by providing a new policy, presented below, that only contains requests applicable to P_1.

$P_I.Rul_{I1}$: role=manager, act=update,

time= [8am, 8pm], effect= Permit.

In Theorem 1, (1),(2),(3), and (4) together state that operators $+$, $\&$ are both commutative and associative; (5) (6) states that the absorption property holds between $+$, $\&$, and $+$, $-$; (7), (8), (9) and (10) state that the operators are distributive; and finally, (11) and (12) show the properties of the negation operator. This theorem can be easily proved by using the set representations of FIA operators and the well-known properties of set operators. The proof is similar to that for the Boolean algebra, and hence we do not include the details here.

4.4 Expressiveness of FIA

In this section, we first show that our operators can express the standard policy-combining algorithms defined for XACML policies as well as other more complex policy integration scenario. We then show that the operators in FIA are minimal and complete in that any possible policy integration requirements can be expressed using a FIA expression. Finally, we discuss some interesting reasonability properties of FIA.

4.4.1 Expressing XACML Policy Combining Algorithms in FIA

In XACML there are six standard policy-combining algorithms as follows:

Permit-overrides: The combined result is "Permit" if any policy evaluates to "Permit", regardless of the evaluation result of the other policies. If no policy evaluates to "Permit"

and at least one policy evaluates to "Deny", the combined result is "Deny". The combination of policies P_1, P_2,..., P_n under this policy-combining algorithm can be expressed as $P_1 + P_2 + \cdots + P_n$.

Ordered-Permit-overrides: The behaviour of this policy combining algorithm is exactly the same as Permit-overrides except that the policies must be evaluated in the originally specified order. Thus, if $P_1, P_2, \cdots P_n$ is a ordered set of policies, the combination of these policies under this policy-combining algorithm can be expressed as $P_1 + P_2 + \cdots + P_n$.

Deny-overrides: The combined result is "Deny" if any policy is encountered that evaluates to "Deny". The combined result is "Permit" if no policy evaluates to "Deny" and at least one policy evaluates to "Permit". Deny-overrides is the opposite of permit-overrides. By using the combination of the negation and addition operator, we can express deny-overrides as $\neg((\neg P_1) + (\neg P_2) + \cdots + (\neg P_n))$.

Ordered-Deny-overrides: The behaviour of this policy combining algorithm is exactly the same as Deny-overrides except that the policies must be evaluated in the originally specified order. Thus, if $P_1, P_2, \cdots P_n$ is a ordered set of policies to be combined, the combination of these policies under this policy-combining algorithm can be expressed as $\neg((\neg P_1) + (\neg P_2) + \cdots + (\neg P_n))$.

First-applicable: The combined result is the same as the result of the first applicable policy. This combining algorithm can be expressed by using the precedence operator. Given policies P_1, P_2, ..., P_n, the expression is $P_1 \rhd P_2 \rhd \cdots \rhd P_n$.

Only-one-applicable: The combined result corresponds to the result of the unique policy in the policy set which applies to the request. Specifically, if no policy or more than one policies are applicable to the request, the result of policy combination should be "NotApplicable"; if only one policy is considered applicable, the result should be the result of evaluating the policy.

When combining policies $P_1, \cdots, P_n$ under this policy-combining algorithm, we need to remove from each policy the requests applicable to all the other policies and then com-

bine the results using the addition operator. The final expression is : $(P_1 - P_2 - P_3 - \cdots - P_n) + (P_2 - P_1 - P_3 - \cdots - P_n) + \cdots + (P_n - P_1 - P_2 - \cdots - P_{n-1})$.

Note that the behaviour of *Ordered-Permit-overrides* and *Ordered-Deny-overrides* policy combining algorithms is exactly the same as *Permit-overrides* and *Deny-overrides* respectively except that the policies are evaluated in the originally specified order. The behaviour of the combined policy is the same in the unordered and ordered versions of *Permit(Deny)-overrides* except the set of obligations enforced might be different in the two versions. Thus *Ordered-Permit-overrides* and *Ordered-Deny-overrides* policy combining algorithms can be expressed using the same FIA expressions used for *Permit-overrides* and *Deny-overrides*.

4.4.2 Expressing Complex Policy Integration Requirements in FIA

Our algebra supports not only the aforementioned policy combining algorithms, but also other types of policy combining requirements, like rule constraints. A rule constraint specifies decisions for a set of requests. It may require that the integrated policy has to permit a critical request. Such an integration requirement can be represented as a new policy. Let P be a policy, and c be the policy specifying an integration constraint. We can combine c and P by using the first-applicable combining-algorithm. The corresponding expression is $c \rhd P$. Another frequently used operator is to find the portion of a policy P_1 that differs from a policy P_2, which can be expressed as: $P_1 \,\&\, (\neg P_2)$.

By using the two policy constants, we can easily modify a policy P as an *open policy* or a *closed policy*. An open policy of P allows everything that is not explicitly denied, which can be represented as $P \rhd \mathsf{P_Y}$. A closed policy of P denies everything that is not explicitly permitted, which can be represented as $P \rhd \mathsf{P_N}$.

Our algebra can also express the policy jump(similar to if-then-else), a feature in the iptables firewall languages. The specific requirement is that if a request is permitted by

policy P_1, then the final decision on this request is given by policy P_2; otherwise, the final decision is given by policy P_3. This can be expressed using

$$\Pi_Y(P_1 \,\&\, P_2) + \Pi_N(\neg P_1 \,\&\, P_2)) + \\ \Pi_Y(\neg P_1 \,\&\, P_3) + \Pi_N(P_1 \,\&\, P_3))$$

Among the four sub-expressions, the first one gives Y when both P_1 and P_2 do so, and gives NA in all other cases. Similarly, the second sub-expression gives N when P_1 gives Y and P_2 gives N, and gives NA otherwise. The third sub-expression gives Y when P_1 gives N and P_3 gives Y and finally the fourth sub-expression gives N when both P_1 and P_3 give N.

Next, we elaborate the example mentioned in the introduction where the combination requirements are given for parts of a policy.

Example 21 *Consider the policies introduced in Example 16. Assume that the policies must be integrated according to the following combination requirement: for users whose role is manager, the access has to be granted according to policy P_1; for users whose role is a staff, the access has to be granted according to policy P_2.*

The resultant policy will consist of two parts. One part is obtained from P_1 by restricting the policy to only deal with managers. Such extraction can be expressed in our algebra as $\Pi_{dc_1}(P_1)$ where dc_1 ={(role, {manager}), (act, {read,update}), (time, [8am,8pm])}. *The other part is obtained from P_2 by restricting the policy to only deal with staff. Correspondingly, we can use the expression: $\Pi_{dc_2}(P_2)$ with $dc_2 =$* {(role, {staff}), (act, {read,update}), (time, [8am,8pm])}. *Finally, we have the following expression representing the integrated policy : $\Pi_{dc_1}(P_1) + \Pi_{dc_2}(P_2)$. The integrated policy P_I is thus:*

$P_I.Rul_{I1}$: role=manager, act=read or update,
time=[8am, 6pm], effect=Permit.

$P_I.Rul_{I2}$: role=staff, act=read,
time=[8am, 8pm], effect=Permit.

$P_I.Rul_{I3}$: role=staff, act=update, effect=Deny.

Table 4.3
n policies

$P_1, P_2, ..., P_{k-1}$	P_k	M^*	$f^{k-1}(P_1, P_2, ..., P_{k-1})$
$Y, Y, ..., Y$	Y	$e_{1,1}$	$f^{k-1}_{1,1}(P_1, P_2, ..., P_{k-1})$
...	...	...	...
$NA, NA, ..., NA$	Y	$e_{1,3^{k-1}}$	$f^{k-1}_{1,3^{k-1}}(P_1, P_2, ..., P_{k-1})$
$Y, Y, ..., Y$	N	$e_{2,1}$	$f^{k-1}_{2,1}(P_1, P_2, ..., P_{k-1})$
...	...	...	...
$NA, NA, ..., NA$	N	$e_{2,3^{k-1}}$	$f^{k-1}_{2,3^{k-1}}(P_1, P_2, ..., P_{k-1})$
$Y, Y, ..., Y$	NA	$e_{3,1}$	$f^{k-1}_{3,1}(P_1, P_2, ..., P_{n-1})$
...	...	...	...
$NA, NA, ..., NA$	NA	$e_{3,3^{k-1}}$	$f^{k-1}_{3,3^{k-1}}(P_1, P_2, ..., P_{k-1})$

(a)

P_k	$e_{i,j}$	$f^k_{i,j}$
Y	Y	$f^{k-1}_{i,j}\&(P_k\&\mathsf{P_Y})$
Y	N	$f^{k-1}_{i,j}\&[\neg(P_k\&\mathsf{P_Y})]$
N	Y	$f^{k-1}_{i,j}\&[\neg(P_k\&\mathsf{P_N})]$
N	N	$f^{k-1}_{i,j}\&(P_k\&\mathsf{P_N})$
NA	Y	$f^{k-1}_{i,j}$
NA	N	$f^{k-1}_{i,j}$

(b)

4.4.3 Completeness

While we have shown that many policy integration scenarios can be handled by the operators in the algebra, our list of examples is certainly not exhaustive. A question of both theoretical and practical importance is whether FIA can express all possible ways of integrating policies, that is, whether FIA is *complete*. Addressing this question requires choosing a suitable notion of completeness. There are different degrees of completeness, and we show that FIA is complete in the strongest sense. First, while Table 4.1 gave the *policy combination matrices* for the four binary operators, many other matrices are possible, and each such matrix can be viewed as a binary operator for combining two policies. As there are three possibilities for each cell in a matrix, namely, Y, N, and NA, and there are nine cells, the total number of matrices is $3^9 = 19683$. Second, when n ($n \geq 2$) policies are combined, policy combination can be expressed using a n-dimensional matrix. We show that each such n-dimensional matrix can be expressed using $\langle \mathsf{P_N}, \mathsf{P_Y}, +, \&, \neg \rangle$. Finally, a fine-grained integration may use different policy combination matrices for different requests. We show that this can be handled by using the operator Π_{dc} in addition to $\langle \mathsf{P_N}, \mathsf{P_Y}, +, \&, \neg \rangle$.

Theorem 4.4.1 *(Completeness) Given n ($n \geq 1$) policies P_1, P_2, ..., P_n, let $M^*(P_1, P_2, ..., P_n)$ be a n-dimensional policy combination matrix which denotes the combination result of the n policies. There exists a FIA expression $f_I(P_1, P_2, ..., P_n)$ that is equivalent to $M^*(P_1, P_2, ..., P_n)$.*

Proof We prove this theorem by induction. The base case is when $n = 1$. Given a policy P_1, its 1-dimensional matrix (Table 3) contains three entries corresponding to the Permit, Deny and NotApplicable request sets . For each entry, we aim to find an expression f_i ($1 \leq i \leq 3$). When e_1 is Y, $f_1 = \mathsf{P_Y}\&P_1$; when e_1 is N, $f_1 = \neg(\mathsf{P_Y}\&P_1)$. Similarly, we can obtain f_2, which is $\mathsf{P_N}\&P_1$ for e_2 equal to N and $\neg(\mathsf{P_N}\&P_1)$ for e_2 equal to Y. f_3 is $\mathsf{P_Y} - P_1$ when e_3 is Y, and $\mathsf{P_N} - P_1$ when e_3 is N. Finally, f_I is the sum of three expressions, i.e., f_I=f_1+f_2+f_3. Note that when all three entries are NA, the integrated policy will be $\mathsf{P_{NA}}$.

Table 4.4
1-dimensional Policy Combining Matrix

P_1	Y	N	NA
P_I	e_1	e_2	e_3

Assuming that when $n = k - 1$ the theorem holds, we now consider the case when $n = k$. As shown in Table 4.3(a), $M^*(P_1, ..., P_k)$ has 3^k entries in total, each of which is denoted as $e_{i,j}$ ($1 \leq i \leq 3, 1 \leq j \leq 3^{k-1}$). Take entries $e_{i,1}$ to $e_{i,3^{k-1}}$ as a *(k-1)*-dimensional policy combination matrix, and we have three such *(k-1)*-dimensional policy combination matrices corresponding to the policy P_k's effect. Based on the assumption, we obtain the FIA expression for each cell for the $k - 1$ policies as shown in the column of $f^{k-1}(P_1, ..., P_{k-1})$.

Next, we extend $f^{k-1}(P_1, ..., P_{k-1})$ to $f^k(P_1, ..., P_k)$ for each cell in M^* (in what follows we use f^{k-1} and f^k for short). According to the effect of P_k and $e_{i,j}$, we summarize the expressions of f^k in Table 4.3(b). Note that we do not need to consider the cell where $e_{i,j}$ is NA.

Finally, we add up f^k for all the cells and obtain the expression $f(P_1, P_2, ..., P_k)$.

We have shown that the theorem holds for $n = 1$, and we have also shown that if the theorem holds for $n = k - 1$ then it holds for $n = k$. We can therefore state that it holds for all n. ■

So far, we have proved the completeness in the scenario when there is one n-dimensional combination matrix for all requests. In the following theorem, we further consider the fine-grained integration when there are multiple combination matrices each of which is corresponding to a subset of the requests.

Definition 4.4.1 *A fine-grained integration specification is given by* $[(R_1, M_1^*), (R_2, M_2^*), \cdots, (R_k, M_k^*)]$*, where* $R_1, R_2, \cdots, R_k$ *form a partition of* $\mathcal{R}_\Sigma$ *(the set of all requests over the vocabulary* Σ*), i.e.,* $\mathcal{R}_\Sigma = R_1 \cup R_2 \cup ... \cup R_k$ *(*$k \geq 1$*) and* $R_i \cap R_j = \emptyset$ *when* $i = j$*, and each* $M_i^*(P_1, .., P_n)$ *(*$1 \leq i \leq k$*) is a* n*-dimensional policy combination matrix. This specification asks requests in each set* R_i *to be integrated according to the matrix* M_i^**.*

Theorem 4.4.2 *Given a fine-grained integration specification* $[(R_1, M_1^*), (R_2, M_2^*), \cdots, (R_k, M_k^*)]$*, if for each* R_i*, there exists* $dc_{i,1}, \cdots, dc_{i,m_i}$ *such that* $R_i = R(dc_{i,1}) \cup \cdots \cup R(dc_{i,m_i})$*(where* $R(dc_{i,j})$ *denotes the set of requests satisfying* $dc_{i,j}$*), then there exists a FIA expression* $f_I(P_1, P_2, ..., P_n)$ *that achieves the integration requirement.*

Proof We first use the domain projection operator Π_{dc} to project each policy according to $dc_{1,1}, \cdots, dc_{k,m_k}$. For requests in each $R(dc_{i,j})$, there is one fixed M_i^*. By Theorem 4.4.1, there is a FIA expression (denoted as $f_{i,j}$) for integrating policies $\Pi_{dc_{i,j}}(P_1)$, ..., $\Pi_{dc_{i,j}}(P_n)$ according to M_i^*. Finally, f_I is the addition of all $f_{i,j}$'s. ■

We note that the above theorem requires that each R_i in the partition to be expressible in a finite number of domain constraints.

4.4.4 Minimal Set of Operators

Recall that FIA has $\{\mathsf{P_Y}, \mathsf{P_N}, +, \&, \neg, \Pi_{dc}\}$. The operator Π_{dc} is needed to deal with fine-grained integration. Operators $\{\mathsf{P_Y}, \mathsf{P_N}, +, \&, \neg\}$ are complete in the sense that

any policy combination matrix can be expressed using them. A natural question is among the set $\Theta = \{\mathsf{P_N}, \mathsf{P_Y}, \mathsf{P_{NA}}, +, \&, \neg, \Pi_Y, \Pi_N, -, \triangleright\}$, what subsets are *minimally complete*. We say a subset of Θ is minimally complete, if operators in the subset are sufficient for defining all other operators in Θ, and any smaller subset cannot define all operators in Θ. The following theorem answers this question. The only redundancy in $\{\mathsf{P_Y}, \mathsf{P_N}, +, \&, \neg\}$ is that only one of $\mathsf{P_Y}$ and $\mathsf{P_N}$ is needed.

Theorem 4.4.3 *Among the 10 operators in Θ, there are 12 minimally complete subsets. They are the 12 elements in the cartesian product $\{\neg\} \times \{\mathsf{P_Y}, \mathsf{P_N}\} \times \{\Pi_Y, \Pi_N, \&\} \times \{+, \triangleright\}$.*

Proof

- The policy constant $\mathsf{P_N}$ cannot be expressed using $\Theta \setminus \{\mathsf{P_Y}, \mathsf{P_N}\}$. When given $\neg$, $\mathsf{P_Y}$ and $\mathsf{P_N}$ can be derived from each other: $\mathsf{P_Y} = \neg\mathsf{P_N}$ and $\mathsf{P_N} = \neg\mathsf{P_Y}$.

  We need to show that no policy expression using operators in $\Theta \setminus \{\mathsf{P_Y}, \mathsf{P_N}\}$ exists that is equivalent to $\mathsf{P_N}$. Consider the information ordering among the three values: $Y > NA$ and $N > NA$. The key observation is that the operators in $\Theta \setminus \{\mathsf{P_Y}, \mathsf{P_N}\}$ are all non-increasing in the information ordering.

  Suppose, for the sake of contradiction, that a policy expression $f(P_1, P_2, \cdots, P_n)$ constructed from $\Theta \setminus \{\mathsf{P_Y}, \mathsf{P_N}\}$ and policies $P_1, \cdots, P_n$ is equivalent to $\mathsf{P_N}$. Then this must mean that no matter what actual policies are used to instantiate $P_1, \cdots, P_n$, the result is $\mathsf{P_N}$. Let $e_0 = f(\mathsf{P_{NA}}, \mathsf{P_{NA}}, \cdots, \mathsf{P_{NA}}) = \mathsf{P_N}$. We now use a structural induction to show that e_0 must give NA for every request and thus a contradiction follows. For the base case, we have policy constant $\mathsf{P_{NA}}$, this is true. For the unary operators, if e gives NA for a request, then $\Pi_Y(e)$, $\Pi_N(e)$, and $\neg(e)$ are also NA. For the binary operators $+$, $-$, $\&$, and $\triangleright$, if both operands are NA for a request, then the result is also NA for the request.

- The unary operator $\neg$ cannot be expressed using $\Theta \setminus \{\neg\}$.

  The key observation is that without $\neg$, one cannot switch Y and N.

Suppose, for the sake of contradiction, that $e_0(P)$ is equivalent to $\neg P$. Let P be a policy that returns Y on r_1 and N on r_2. Then $e_0(P)$ must return N on r_1 and Y on r_2, i.e., it must give (N, Y) on r_1 and r_2. We use structural induction to show that the result $e_0(P)$ gives for r_1 and r_2 must be among (Y, N), (Y, Y), (N, N), (NA, NA), (Y, NA), (NA, N). That is, if the answer for r_1 is N, then the answer for r_2 must be N, and if the answer for r_2 is Y, the answer for r_1 must be Y. Hence contradiction. For the base case, this holds for P and the three constants $\mathsf{P_Y}, \mathsf{P_N}, \mathsf{P_{NA}}$. One can verify that the six pairs are closed under $\Pi_Y, \Pi_N, +, -, \,\&\,, \rhd$.

- The binary operator $\&$ cannot be expressed using $\Theta \setminus \{\,\&\,, \Pi_Y, \Pi_N\}$. Given $\neg$, Π_Y and Π_N can be expressed from each other: $\Pi_Y(P) = P \,\&\, \neg \mathsf{P_N}$, $\Pi_N(P) = \neg \Pi_Y(\neg P)$. Π_Y can be expressed using $\{\,\&\,, \mathsf{P_Y}\}$ by definition, and $\&$ can be expressed using $\{\mathsf{P_N}, +, \neg, \Pi_N\}$.

  Assume, for the sake of contradiction, that $e_0(P_1, P_2)$ is equivalent to $P_1 \& P_2$. Let P_1 be a policy that returns (Y, Y) on r_1 and r_2, and P_2 be a policy that returns (Y, N) on r_1 and r_2. Then $e_0(P_1, P_2)$ must return (Y, NA) on r_1 and r_2. We show that this is not possible. The key insight here is that without $\,\&\,, \Pi_Y, \Pi_N$, one cannot get information asymmetry Y or N for one request and NA for another from symmetric policies.

  We use a structural induction to show that the result $e(P_1, P_2)$ gives for r_1 and r_2 must be an element of the following set: $\{(Y, N), (Y, Y), (NA, NA), (N, Y), (N, N)\}$. This holds for all $P_1, P_2, \mathsf{P_Y}, \mathsf{P_N}, \mathsf{P_{NA}}$. One can verify that the set is closed under $\neg, +, -, \rhd$.

  Given $\{\mathsf{P_N}, +, \neg\}$, $\&$ can be expressed using either Π_Y or Π_N:

$$P_1 \,\&\, P_2 = (P_1 \sqcap P_2) + \neg(\neg P_1 \sqcap \neg P_2),$$

  where

$$P_1 \sqcap P_2 = \Pi_Y(P_1) - (\Pi_N(\mathsf{P_N} + P_2)),$$

and $-$ can be defined using Π_Y, Π_N as:

$$P_1 - P_2 = (\ \Pi_Y(\neg(\neg P_1 + P_2 + \neg P_2))) + (\Pi_N(P_1 + P_2 + \neg P_2))\,.$$

The effect of $P_1 \sqcap P_2$ is to authorize any request that is authorized by both P_1 and P_2, and to be NA for all other requests.

- The binary operator $+$ cannot be expressed using $\Theta \setminus \{+, \rhd\}$. However, $+$ can be expressed using $\{\mathsf{P_N}, \neg, \&\,, \rhd\}$ or $\{\mathsf{P_N}, \neg, \Pi_N, \rhd\}$, and $\rhd$ can be expressed using $\{\mathsf{P_N}, +, \neg, \&\}$.

  Suppose, for the sake of contradiction, that an expression $e_0(P_1, P_2)$ constructed from P_1, P_2 and $\Theta \setminus \{+, \rhd\}$ is equivalent to $P_1 + P_2$. Let P_1 be a policy that returns (Y, NA) on r_1 and r_2, and P_2 be a policy that returns (NA, N) on r_1 and r_2. Then $e_0(P_1, P_2)$ must return (Y, N) on r_1 and r_2.

  We use a structural induction to show that the result $e(P_1, P_2)$ gives for r_1 and r_2 must be an element of the set $\{(Y, Y), (Y, NA), (N, N), (N, NA), (NA, Y), (NA, N), (NA, NA)\}$. Hence contradiction. This is satisfied by $P_1, P_2, \mathsf{P_Y}, \mathsf{P_N}, \mathsf{P_{NA}}$. One can verify that these seven values are closed under $\neg, \Pi_Y, \Pi_N, -, \&$.

  The $+$ operator can be expressed using $\{\mathsf{P_N}, \neg, \&\,, \rhd\}$:

$$P_1 + P_2 = (P_1 \,\&\, \mathsf{P_Y}) \rhd (P_2 \,\&\, \mathsf{P_Y}) \rhd (P_1 \,\&\, \mathsf{P_N}) \rhd (P_2 \,\&\, \mathsf{P_N}).$$

  Similarly, the $+$ operator can be expressed using $\{\mathsf{P_N}, \neg, \Pi_N, \rhd\}$:

$$P_1 + P_2 = (\Pi_Y P_1) \rhd (\Pi_Y P_2) \rhd (\Pi_N P_1) \rhd (\Pi_N P_2).$$

  Recall that the operator $\rhd$ is defined using $+$ and $-$ as $P_1 \rhd P_2 = P_1 + (P_2 - P_1)$, and $P_1 - P_2 = (\mathsf{P_Y} \,\&\, (\neg(\neg P_1 + P_2 + \neg P_2))) + (\mathsf{P_N} \,\&\, (P_1 + P_2 + \neg P_2))$. Thus, $\rhd$ can be expressed using $\{\mathsf{P_N}, -, \neg, \&\}$.

In summary, among the 10 operators in Θ, for completeness, we must have $\neg$, one in $\{\mathsf{P_Y}, \mathsf{P_Y}\}$, one in $\{\Pi_Y, \Pi_N, \&\}$, and one in $\{+, \triangleright\}$. There are 12 combinations. It is not difficult to verify that every such combination is in fact complete. For example, once we have $\{\neg, \mathsf{P_N}, \Pi_N\}$, adding $+$ allows us to derive $\&$, and then derive $\triangleright$, adding $\triangleright$ allows us to derive $+$ and then $\&$. There are thus 12 minimally complete subsets in Θ. ■

4.5 Integrated Policy Generation

In this section, we present an approach to automatically generate the integrated policy given the FIA policy expression. Internally, we represent each policy as a Multi-Terminal Binary Decision Diagram (MTBDD) [24], and then perform operations on the underlying MTBDD structures to generate the integrated policy. We have chosen an MTBDD based implementation of the proposed algebra because (i) MTBDDs have proven to be a simple and efficient representation for XACML policies [5] and (ii) operators in FIA can be mapped to efficient operations on the underlying policy MTBDDs. Our approach consists of three main phases:

1. **Policy representation**: For each policy P_i in the FIA expression $f(P_1, P_2, ..., P_n)$, we construct a policy MTBDD, T^{P_i}.
2. **Construction of the integrated policy MTBDD**: We combine the individual policy MTBDD structures according to the operations in the FIA expression to construct the *integrated policy MTBDD*.
3. **Policy generation**: The *integrated policy MTBDD* is then used to generate the actual integrated XACML policy.

4.5.1 Policy Representation

Recall from Section 3 that we characterize a policy P as a 3-*tuple* $\langle R_Y^P, R_N^P, R_{NA}^P \rangle$, where R_Y^P is the set of requests permitted by the policy, R_N^P is the set of requests denied by the policy and R_{NA}^P is the set of requests not applicable to the policy. Alternatively, we

can define P as a function $P : R \rightarrow E$ from the domain of requests R onto the domain of effects E, where $E = \{Y, N, NA\}$.

An XACML policy can be transformed into a compound Boolean expression over request attributes [30]. A compound Boolean expression is composed of atomic Boolean expressions (AE) combined using the logical operations $\vee$ and $\wedge$. Atomic Boolean expressions that appear in most policies belong to one of the following two categories: (i) one-variable equality constraints, $a \triangleright c$, where a is an attribute name, c is a constant, and $\triangleright \in \{=, =\}$; (ii) one-variable range constraints, $c_1 \triangleleft a \triangleright c_2$, where a is an attribute name, c_1 and c_2 are constants, and $\triangleleft$,$\triangleright \in \{<, \leq\}$.

Example 22 *Policy P_1 from Example 16 can be defined as a function :*

$$P_1(r) = \begin{cases} Y & \textit{if } role = manager \wedge (act = read \vee \\ & \quad act = update) \wedge 8am \leq time \leq 6pm \\ N & \textit{if role=staff } \wedge \textit{ act=read} \\ NA & \textit{Otherwise} \end{cases}$$

where r is a request of the form $\{(role, v_1), (act, v_2), (time, v_3)\}$.

We now encode each *unique* atomic Boolean expression AE_i in a policy into a Boolean variable x_i such that: $x_i = 0$ if AE_i is `false`; $x_i = 1$ if AE_i is `true`. To determine *unique* atomic Boolean expressions we use the following definition. The Boolean encoding for policy P_1 is given in Table 4.5.

Table 4.5
Boolean encoding for P_1

i	AE_i	x_i
0	$role = manager$	x_0
1	$role = staff$	x_1
2	$act = read$	x_2
3	$act = update$	x_3
4	$8am \leq time \leq 6pm$	x_4

Using the above Boolean encoding, a policy P can be transformed into a function $P : B^n \mapsto E$, over a vector of Boolean variables, $\vec{x} = x_0, x_1, \cdots, x_n$, onto the finite set of effects $E = \{Y, N, NA\}$, where n is the number of unique atomic Boolean expressions in policy P. A request r corresponds to an assignment of the Boolean vector $\vec{x}$, which is derived by evaluating the atomic Boolean expressions with attribute values specified in the request.

Example 23 *After Boolean encoding, the policy P_1 is transformed into the function :*

$$P_1(\vec{x}) = \begin{cases} Y & \textit{if } x_0 \wedge (x_2 \vee x_3) \wedge x_4 \\ N & \textit{if } x_1 \wedge x_4 \\ NA & \textit{Otherwise} \end{cases}$$

The transformed policy function can now be represented as a MTBDD. A MTBDD provides a compact representation of functions of the form $f : \mathbb{B}^n \mapsto \mathbb{R}$, which maps bit vectors over a set of variables ($\mathbb{B}^n$) to a finite set of results ($\mathbb{R}$). The structure of a MTBDD is a rooted acyclic directed graph. The internal (or non-terminal) nodes represent Boolean variables and the terminals represent values in a finite set. Each non-terminal node has two edges labeled 0 and 1 respectively. Thus when a policy is represented using a MTBDD, the non-terminal nodes correspond to the unique atomic Boolean expressions and the terminal nodes correspond to the effects. Each path in the MTBDD represents an assignment for the Boolean variables along the path, thus representing a request r. The terminal on a path represents the effect of the policy for the request represented by that path. Note that different orderings on the variables may result in different MTBDD representations and hence different sizes of the corresponding MTBDD representation. Several approaches for determining the variable ordering that results in an optimally sized MTBDD can be found in [31]. For examples discussed here, we use the variable ordering $x_0 \prec x_1 \prec x_2 \prec x_3 \prec x_4$. The MTBDD of the policy P_1 is shown in Figure 4.2, where the dashed lines are 0-edges and solid lines are 1-edges.

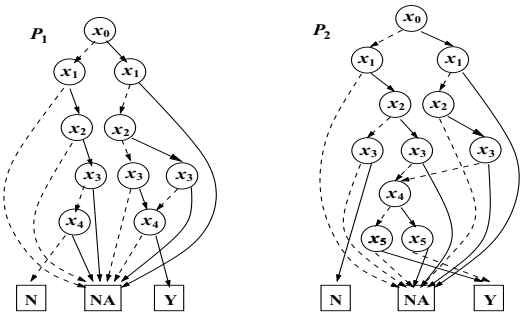


Figure 4.2. MTBDDs of P_1 and P_2

Compound Boolean expression representing the policies to be integrated may have atomic Boolean expressions with matching attribute names but overlapping value ranges. In such cases, we need to transform the atomic Boolean expressions with overlapping value ranges into a sequence of new atomic Boolean expressions with disjoint value ranges, before performing the Boolean encoding. A generic procedure for computing the new atomic Boolean expression is described below.

Assume that the original value ranges of an attribute a are $[d_1^-, d_1^+]$, $[d_2^-, d_2^+]$, ..., $[d_n^-, d_n^+]$ (the superscript '-' and '+' denote lower and upper bound respectively). We sort the range bounds in an ascending order, and then employ a plane sweeping technique to obtain the disjoint ranges: $[d_1'^-, d_1'^+]$, $[d_2'^-, d_2'^+]$, ..., $[d_m'^-, d_m'^+]$, which satisfy the following three conditions: (i) $d_i'^-, d_i'^+ \in D$, $D = \{d_1^-, d_1^+, ..., d_n^-, d_n^+\}$; (ii) $\cup_{i=1}^{m}[d_i'^-, d_i'^+] = \cup_{j=1}^{n}[d_j^-, d_j^+]$; and (iii) $\cap_{i=1}^{m}[d_i'^-, d_i'^+] = \emptyset$.

Consider policy P_2 from Example 16. We can observe that the atomic Boolean expression $8am \leq time \leq 6pm$ in P_1 refers to the same attribute as in the atomic Boolean expression $8am \leq time \leq 8pm$ in P_2 and their value ranges overlap. In order to distinguish these two atomic Boolean expressions during the later policy integration, we split the value ranges and introduce the new atomic Boolean expression $6pm \leq time \leq 8pm$. The expression $8am \leq time \leq 8pm$ in P_2 is replaced with $(8am \leq time \leq 6pm \vee 6pm \leq time \leq 8pm)$. Boolean encoding is then performed for the two policies by considering unique atomic Boolean expressions across both policies.

Example 24 *By introducing another atomic Boolean expression* $6pm \leq time \leq 8pm$*, i.e.* x_5*, the transformed function for policy* P_2 *is :*

$$P_2(\vec{x}) = \begin{cases} Y, & \textit{if } (x_0 \vee x_1) \wedge x_2 \wedge (x_4 \vee x_5) \\ N, & \textit{if } x_1 \wedge x_3 \\ NA & \textit{Otherwise} \end{cases}$$

Using the same variable ordering $x_0 \prec x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$ we construct the MTBDD for P_2 shown in Figure 4.2.

Procedure Apply($Node_1$**,** $Node_2$**,** OP**)**
Input : $Node_1$, $Node_2$ are MTBDD nodes,
OP is a policy operation

1. initiate $Node_I$ // $Node_I$ is the combination result
2. **if** $Node_1$ and $Node_2$ are terminals **then**
3. $Node_I \leftarrow (Node_1\ OP\ Node_2, null, null)$
4. **else**
5. **if** $Node_1.var = Node_2.var$ **then**
6. $Node_I.var \leftarrow Node_1.var$
7. $Node_I.left \leftarrow$ Apply($Node_1.left$, $Node_2.left$, OP)
8. $Node_I.right \leftarrow$ Apply($Node_1.right$, $Node_2.right$, OP)
9. **if** $Node_1.var$ precedes $Node_2$.var **then**
10. $Node_I.var \leftarrow Node_1.var$
11. $Node_I.left \leftarrow$ Apply($Node_1.left$, $Node_2$, OP)
12. $Node_I.right \leftarrow$ Apply($Node_1.right$, $Node_2$, OP)
13. **if** $Node_2.var$ precedes $Node_1$.var **then**
14. $Node_I.var \leftarrow Node_2.var$
15. $Node_I.left \leftarrow$ Apply($Node_2.left$, $Node_1$, OP)
16. $Node_I.right \leftarrow$ Apply($Node_2.right$, $Node_1$, OP)
17. return $Node_I$

Figure 4.3. Description of the `Apply` procedure

4.5.2 Construction of Integrated Policy MTBDD

Given the FIA expression $f(P_1, P_2, ..., P_n)$ and the MTBDD representations $T^{P_1}, T^{P_2}, ..., T^{P_n}$ of the policies P_1,P_2,...,P_n respectively, we construct the integrated policy MTBDD T^{P_I}, by performing the operations (specified in f) on the individual policy MTBDDs.

Operations on policies can be expressed as operations on the corresponding policy MTBDDs. Many efficient operations have been defined and implemented for MTBDDs [24]. In particular, we use the `Apply` operation defined on MTBDDs to perform the FIA binary operations $\{+, -, \&, \triangleright\}$ and `not` operation defined on MTBDD to perform the FIA unary negation ($\neg$) operation. We introduce a new MTBDD operation called `Projection` to perform the effect(Π_Y and Π_N) projection and domain projection(Π_{dc} operations defined in FIA.

The `Apply` operation combines two MTBDDs by a specified binary arithmetic operation. A high level description of the `Apply` operation is shown in Figure 4.3, where *var*, *left*, *right* refer to the variable, left child and right child of a MTBDD node, respectively. The `Apply` operation traverses each of the MTBDDs simultaneously starting from the root node. When the terminals of both MTBDDs are reached, the specified operation is applied on the terminals to obtain the terminal for the resulting combined MTBDD. A variable ordering needs to be specified for the `Apply` procedure.

The integrated MTBDD T^{P_I} for the policy expression $f(P_1, P_2) = P_1 + P_2$ is obtained by using MTBDD operation `Apply(`$T^{P_1}.root$`,` $T^{P_2}.root$`,` $+$`)`, where "root" refers to the root node of the corresponding MTBDD. Figure 4.4 (in appendix) shows the integrated policy MTBDD. The same variable ordering $x_0 \prec x_1 \prec x_2 \prec x_3 \prec x_4 \prec x_5$ has been used in the construction of the integrated policy MTBDD.

The procedure for performing effect projection operations is the following. For Π_Y, those paths in T^P that lead to N are redirected to the terminal NA. Similarly, for Π_N, those paths in T^P that lead to Y are redirected to the terminal NA.

For the domain projection operation with domain constraint dc, we traverse the policy MTBDD from the top to the bottom and check the atomic Boolean expression associated

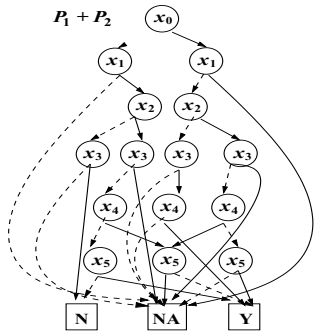


Figure 4.4. MTBDDs of $P_1 + P_2$

Procedure Projection(T^P**,** dc**)**
Input : T^P is the MTBDD of policy P,
dc is a domain constraint

1. **for** each internal node $Node$ in T^P
2. **if** $Node.var$ is in dc **then**
3. replace the domain of $Node.var$ according to dc
4. **else**
5. **if** $Node$ is the root **then**
6. let $Node.left$ become new root
7. **else**
8. let $Node$'s parent node point to $N.left$
9. delete $Node$
10. delete those nodes that have no incoming edges

Figure 4.5. Description of the `Projection` operation

with each node (denoted as $Node$). There are two cases. If the atomic Boolean expression of $Node$ contains an attribute specified in dc, we simply replace the attribute domain with the new domain given by dc. Otherwise, it means $Node$ represents an attribute no longer applicable to the resulting policy, and hence we should remove it. After removing $Node$, we need to adjust the pointer from its parent node by redirecting it to $Node$'s left child which leads to the path when N is not considered. After all nodes have been examined, those nodes that have no incoming edges are also removed. Figure 4.5 summarizes the algorithm.

Thus, given any arbitrary FIA expression $f(P_1, P_2, ..., P_n)$, we can use a combination of the `Apply`, `not`, `Projection` MTBDD operations on the policy MTBDDs to generate the integrated policy MTBDD. An example is given below.

Consider the FIA policy expression for the only-one-applicable policy combining algorithm together with the domain constraint *dc* = {*(role,* {*manager*}*), (act,* {*read, update*}*), (time, [8am, 8pm])*}. Here, $f(P_1, P_2) = \Pi_{dc}((P_1 - P_2) + (P_2 - P_1))$. The integrated MTBDD can be obtained by using the `Apply` and `Projection` operations as follows :

$$\texttt{Projection}(\texttt{Apply}(\texttt{Apply}(T^{P_1}.root, T^{P_2}.root, -),$$
$$\texttt{Apply}(T^{P_2}.root, T^{P_1}.root, -), +), dc) .$$

4.5.3 XACML Policy Generation

In the previous section, we have presented how to construct the integrated MTBDD given any policy expression f. Though such integrated MTBDD can be used to evaluate requests with respect to the integrated policy, they cannot be directly deployed in applications using the access control system based on XACML. Therefore, we develop an approach that can automatically transform MTBDDs to actual XACML policies. The policy generation consists of three steps :

1. Find the paths in the combined MTBDD that lead to the Y and N terminals, and represent each path as a Boolean expression over the Boolean variable of each node.
2. Map the above Boolean expressions to the Boolean expressions on actual policy attributes.
3. Translate the compound Boolean expression obtained in step 2 into a XACML policy.

We first elaborate on step 1. In the MTBDD, each node has two edges, namely *0-edge* and *1-edge*. The 0-edge and 1-edge of a node labelled x_i correspond to *edge-expressions* $\bar{x}_i$ and x_i respectively. A path in the MTBDD corresponds to an expression which is the conjunction of *edge-expressions* of all edges along that path. We refer to this as a *path-expression*. Those paths leading to the same terminal correspond to the disjunction of *path-*

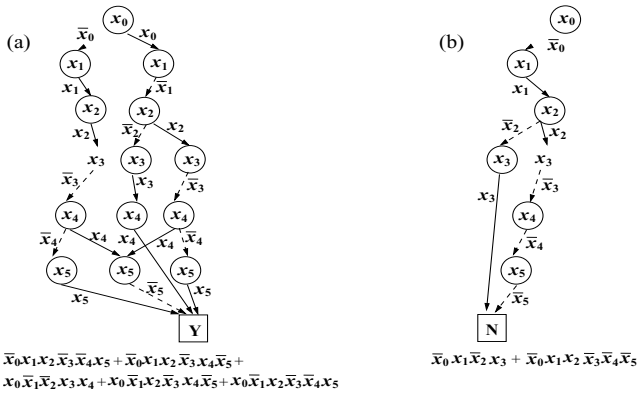


Figure 4.6. Policy generation using MTBDD

PolicyId=P1+P2
<RuleId=R1 Effect=Deny>
<Target>
<Subject role=staff>
<Action act=update>
</Target>
</Rule>
<RuleId=R2 Effect=Deny>
<Target>
<Subject role=staff>
<Action act=read>
</Target>
<Condition time $= [8am, 6pm]$ AND time $= [6pm, 8pm]$>
</Rule>
<RuleId=R3 Effect=Permit> $\cdots$

Figure 4.7. The integrated XACML policy representing $P_1 + P_2$

expressions. Figure 4.6 shows an example of $P_1 + P_2$, where the Figure 4.6(a) and (b) show the paths leading to the Y and N terminals and the corresponding Boolean expressions.

Next, we replace Boolean variables in the path-expressions with the corresponding atomic Boolean expressions by using the mapping constructed in the Boolean encoding phase. During the transformation in each path-expression, we need to remove some redundant information. For instance, the resulting expression may contain an attribute with both

equality and inequality functions like $(role = manager) \wedge (role = staff)$. In that case, we only need to keep the equality function of the attribute.

Example 25 *After the replacement, the Boolean expression of the N terminal in Figure 4.6 is transformed as follows:*
$(role = staff \wedge act = update) \vee (role = staff \wedge act = read \wedge time = [8am, 6pm] \wedge time = [6pm, 8pm])$

The last step is to generate the actual XACML policy from the compound Boolean expression obtained in previous step. Specifically, for each path-expression whose evaluation is Y, a permit rule is generated; and for each path-expression whose evaluation is N, a deny rule is generated. Attributes that appear in conditions of the rules in original policies still appear in conditions of the newly generated rules, and attributes that appear in targets in the original policies still appear in targets in the integrated policy. Here we do not distinguish the policy target with rule target. Instead, all targets appear as rule targets.

Consider policies P_1 and P_2 in Example 16, and the Boolean expression in Example 25. We generate the corresponding deny rules for the integrated policy of $P_1 + P_2$ as shown in Figure 4.7.

4.6 Obligations

In the discussions so far we have not considered the impact of the presence of *obligations* when integrating policies using FIA. In this section, we briefly discuss how to generate the correct set of obligations for an XACML policy that is generated as result of integrating policies using operations in FIA. We plan to extend the current algebra with the notion of obligations as part of future work.

An XACML policy can have obligations associated with "Permit" and/or "Deny" effects. According to the XACML standard specification, the set of obligations returned by the policy decision point for a given request is derived from the evaluation tree of policy sets and policies. The final set of obligations returned by the policy decision point includes

obligations of policies along those paths of the tree where effect at each level is the same as the final effect returned by the policy decision point.

Treating the obligations similarly in our framework, we can obtain the set of obligations that must be enforced by integrated policy. The set of obligations can be easily derived when using the $\&$ or $-$ operations for integrating policies. For example, consider the $\&$ operation. In this case, since a request in the integrated policy is permitted(denied) only when both the policies permit(deny) the request, the set of "Permit"("Deny") obligations enforced by the integrated policy must be the union of the set of "Permit"("Deny") obligations specified in the individual policies that are being combined. Similarly, when combining policies P_1 and P_2 using the $-$ operation, a request is permitted(denied) in the combined policy when the request is permitted(denied) by P_1 and is not applicable to P_2 and hence the set of "Permit"("Deny") obligations enforced by the integrated policy is exactly the same as the "Permit"("Deny") obligations of P_1.

Deriving the set of obligations that must be enforced by the integrated policy is not straightforward in the case of the $+$ and $\triangleright$ operations because when a request is permitted(denied) in the policy integrated using these operations, the decision of the individual policies participating in the integration with respect to this request is not known. However, this problem can be solved by generating an integrated XACML *policy set* instead of an integrated XACML *policy* from the integrated MTBDD. Note that each terminal in the integrated MTBDD corresponding to integration of two policies P_1 and P_2 using the $+(\triangleright)$ operation represents a cell in the matrix representation of $+(\triangleright)$ shown in Figure 4.1. Thus the terminals correspond to E_{1i}-E_{2i} ($E_{1i}, E_{2i} \in \{Y, N, NA\}$ and $1 \leq i \leq 9$) where E_{1i} represents effect of P_1 and E_{2i} represents effect of P_2 in cell i. Each terminal is mapped to a final decision E_{fi} which could be one of Y, N or NA depending on the operation used to generate the integrated MTBDD. Using the same techniques for generating a single policy from a given MTBDD, we can generate a sub policy SP_i corresponding to cell E_{1i}-E_{2i} by collecting all paths leading to the E_{1i}-E_{2i} terminal. The correct set of obligations for each of these sub policies in the integrated policy set can be obtained based on the individual decisions E_{1i} and E_{2i} and the final decision E_{fi} given by the specific operation. That is, the

obligations for a sub policy SP_i generated by traversing paths leading to E_{1i}-E_{2i} includes the obligations of policy P_k corresponding to effect E_{ki} ("Permit" obligations if $E_{ki} = Y$ or "Deny" obligations if $E_{ki} = N$) only if E_{ki} is the same as E_{fi} for $k = 1, 2$.

For example, consider an integrated MTBDD that corresponds to the $+$ operation on two policies P_1 and P_2. Then,

- For the policy corresponding to the Y-Y terminal the set of obligations is the union of the set of "Permit" obligations of P_1 and P_2,
- For the policy corresponding to the N-N terminal the set of obligations is the union of the set of "Deny" obligations of P_1 and P_2,
- For the policies corresponding to the Y-N(N-Y) and Y-NA(NA-Y) the set of obligations consists only of the set of "Permit" obligations of P_1(P_2),
- For the policy corresponding to the N-NA(NA-N) terminal the set of obligations consists only of the set of "Deny" obligations of P_1(P_2).

Similar method can be used to derive the obligations for an XACML policy obtained by integrating n policies $P_1, P_2, \cdots, P_n$ using arbitrary FIA operations. In this case, each terminal of the integrated MTBDD will correspond to E_{1i}-E_{2i}-$\cdots$-E_{ni}, $1 \leq i \leq 3^n$. The set of obligations for the sub policy SP_i will include the obligations of policy P_k corresponding to effect E_{ki} ("Permit" obligations if $E_{ki} = Y$ or "Deny" obligations if $E_{ki} = N$) only if E_{ki} is the same as E_{fi} for $k = 1, 2, \cdots, n$. Note that the integrated policy set is only generated from the final integrated MTBDD and not for the MTBDDs obtained when performing the intermediate operations.

4.7 Experimental Evalutation

We performed experiments to evaluate the time taken for performing FIA operations and the time for generating an integrated policy. We also examined the size of the generated integrated policy in terms of the number of rules and number of atomic Boolean expressions

in each rule. All experiments were conducted on a Pentium III 3GHz 500 MB machine. MTBDD operations were implemented using the modified CUDD library developed in [5].

We implemented a random attribute based access control policy generator to generate XACML policies in Boolean form. Each policy contained atomic Boolean expressions on a set of predefined attribute names and values. The Boolean expressions corresponding to the *Condition* element of an XACML policy was derived by randomly concatenating atomic Boolean expressions with the logical $\vee$, $\wedge$ and $\neg$ operators. Each rule was randomly assigned to either permit or deny effect. Each policy was also associated with either a deny-override or permit-override rule combining algorithm.

In the first set of experiments we measured the average time required for performing the FIA operations and the size of the obtained MTBDDs. Figure 4.8 shows along the left y-axis the average time (in ms) for performing $+$ and $\triangleright$ operations on policies in which the total number of atomic Boolean expressions in a policy was fixed to 50 and the number of rules was varied between 2 and 10. This graph shows along the right y-axis the average size, i.e., the number of nodes, in the corresponding integrated MTBDDs. From Figure 4.8,

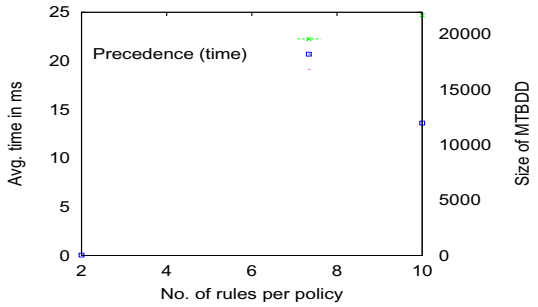


Figure 4.8. Average time and average integrated MTBDD size with respect to operations "+" and "$\triangleright$"

we can observe that the average time taken to perform these operations increases with the increase in the size of the integrated MTBDDs, and it differs for different operators. The reason is that the actual time for performing operations depend on the size of the resulting integrated MTBDD. The larger the MTBDD is, the longer time the integration will take. Performing the $\triangleright$ operation usually resulted in MTBDDs with a smaller size and hence it

took lesser time. Considering that for typical policies we have encountered in real world applications the average number of atomic Boolean expressions lies between 10 and 50, the time trends observed in Figure 4.8 is very encouraging.

We also evaluated the time taken to perform FIA operations on policies with 10 rules and varying number of atomic Boolean expressions. The results are shown in Figure 4.9.

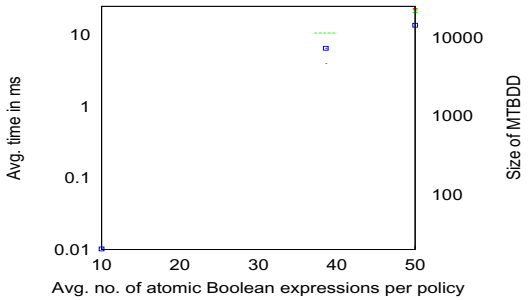


Figure 4.9. Average time(in ms) and average integrated MTBDD size for policies with varying number of atomic Boolean expressions per policy

Although the time complexity of the $+$ and $\triangleright$ operations is proportional to the product of the sizes of the input policy MTBDDs, the actual times observed for performing these operations differed. This is because the actual times for these operations also depend on the size of the resulting integrated MTBDD. Observe that the actual time taken to perform these operations increases with the increase in the size of the generated MTBDDs. Performing the $\triangleright$ operation almost always resulted in MTBDDs whose size was smaller and hence took lesser time. Considering that for typical policies we have encountered in real world applications the average number of atomic Boolean expressions lie between 10 and 50, the time trends observed in figures 4.8 and 4.9 are very encouraging.

In the second set of experiments, we studied the characteristics of the integrated policy. Because the number of rules generated in the integrated policy is equal to the number of paths which can be exponential in the size of the integrated MTBDD a large number of rules can be generated. We used ESPRESSO [26], a two level logic minimizer to reduce the number of rules. ESPRESSO uses state-of-the-art heuristic Boolean minimization algorithms to produce a minimal equivalent representation of two-valued or multiple-valued

Boolean functions. The minterms obtained from the integrated MTBDD were transformed to ESPRESSO inputs and the minimized output was used for policy generation. Table 4.6 summarizes the results obtained for + operation performed on data sets that contained policies with 4 and 8 rules with an average 20 atomic Boolean expressions per policy. We observe that using ESPRESSO a substantial decrease in the number of rules and atomic Boolean expressions(terms) was obtained. For the data sets used in our experiments we observed a 75% to 99% reduction in the number of rules and 35% to 71% reduction in the number of atomic Boolean expressions per rule.

Table 4.6
Characteristics of integrated policy

		Without ESPRESSO		With ESPRESSO	
# of Rules in a Policy	Rule Type	Avg # of Rules	Avg # of terms per Rule	Avg # of Rules	Avg # of terms per Rule
4 Rules	Permit Deny	790 3625	19 21	69 233	9 6
8 Rules	Permit Deny	20192 131348	37 36	1221 152	19 11

Finally, we evaluated the time required for generating the integrated XACML policy. It is worth noting that this step is optional. Users can also directly use the integrated MTBDD for request evaluation. Figure 4.10 shows the policy generation times for integrated MTBDDs with different number of paths. The time for policy generation is observed to be proportional to the number of paths in the integrated MTBDD. The number of paths in the integrated MTBDD is in turn determined by the nature of the compound Boolean expressions in the policies and the chosen variable ordering.

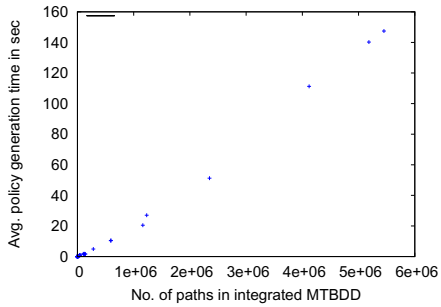


Figure 4.10. Average time (in seconds) for integrated policy generation

5 RELATED WORK

The techniques proposed in this thesis are closely related to works in the area of access control policy analysis and policy integration.

5.1 Policy Analysis

Works in the area of policy analysis can be broadly classified into two categories: (i) those that deal with verification of properties of a single policy and (ii) those that deal with policy similarity analysis that may involve verification of different relationships such as equivalence, refinement, redundancy etc among two or more policies.

5.1.1 Single Policy Analysis

Most approaches for single policy property analysis are based on model checking techniques [32–34]. Ahmed et al. [32] propose a methodology for analyzing four different policy properties in the context of role-based CSCW (Computer Supported Cooperative Work) systems; this methodology uses finite-state based model checking. Since they do not present any experimental results, it is not clear if their state-exploration approach can scale well to policies with a very large set of attributes and conditions. Guelev et al. propose a formal language for expressing access-control policies and queries [33]. Their subsequent work [34] proposes a model-checking algorithm which can be used to evaluate access control policies written in their proposed formal language. The evaluation includes not only assessing whether the policies give legitimate users enough permissions to perform their tasks, but also checking whether the policies prevent intruders from achieving some malicious goals. However, the tool can only check policies of reasonable size.

5.1.2 Policy Similarity Analysis

Existing approaches to the policy similarity analysis are mostly based on graph, model checking or SAT-solver techniques [5, 13, 35–38]. Koch et al. [36] use graph transformations to represent policy change and integration, which may be used to detect differences among policies. Such an approach supports an intuitive visual representation which can be very useful in the design of a customized access control policy. However, it can only be used as a specification method but not as an execution method. Backes et al. [35] propose an algorithm for checking refinement of enterprise privacy policies. However, their algorithm only identifies which rule in one policy needs to be compared with the rules in the other policy. They do not provide an approach to the evaluation of condition functions.

A more practical approach is by Fisler et al. [5], who have developed a software tool known as Margrave for analyzing role-based access-control policies written in XACML. Margrave represents policies using the Multi-Terminal Binary Decision Diagram (MTBDD), which can explicitly represent all variable assignments that satisfy a Boolean expression and hence provides a good representation for the relationships among policies. Policy property verification is then formulated as a query on the corresponding MTBDD structures. For processing a similarity query involving two policies, the approach proposed by Fisler et al. is based on combining the MTBDDs of the policies into a CMTBDD (change-analysis MTBDD) which explicitly represents the various requests that lead to different decisions in the two policies. The MTBDD structure has been credited with helping model checking scale to realistic systems in hardware verification. The major shortcoming of Margrave is that it can only handle simple conditions, like string equality matching. A direct consequence of such limitation is an explosion of the MTBDD size when conditions on different data domains (e.g. inequality functions) have to be represented. For example, to represent the condition “time is between 8am to 10pm”, the MTBDD tool needs to enumerate all possible values between “8am” to “10pm”(e.g., “time-is-8:00am”, “time-is-8:01am”, “time-is-8:02am”, ...).

Other relevant approaches are the ones based on SAT-solver techniques. Most such approaches [37, 38] however only handle policy conflict detection. A recent approach by Agrawal et al. [13] investigates interactions among policies and proposes a ratification tool by which a new policy is checked before being added to a set of policies. In [39], McDaniel et al. carry out a theoretical study on automated reconciliation of multiple policies and then prove that this is an NP-complete problem. In [40], Kolovski et al. formalize XACML policies by using description logics and then employ logic-based analysis tools for policy analysis. These SAT-solver based approaches formulate policy analysis as a Boolean satisfiability problem on Boolean expressions representing the policies. Such approaches can handle various types of Boolean expressions, including equality functions, inequality functions, linear functions and their combinations. By construction, the SAT algorithms look for one variable assignment that satisfies the given Boolean expression, although they may be extended to find all satisfying variable assignments. For each round of analysis or query, SAT algorithms need to evaluate the corresponding Boolean expression from scratch. They cannot reuse previous results and are not able to present an integrated view of relationships among policies.

More recently, Mazzoleni et al. [14] also considered policy similarity problem in their proposed policy integration algorithm. In contrast to our approach, they do not quantify the similarity between two policies by assigning a score. Instead, they determine whether two policies *converge*, *diverge*, *extend*, *extend* or *shuffle* with respect to the sets of requests they authorize. Moreover, their method for computing policy similarity assumes that each rule in a policy contains predicates on one attribute. They do not address cases where predicates on multiple attributes are contained in a single rule or cases where multiple predicates concerning the same attribute are contained in a single rule.

Unlike aforementioned works that focus on a special case or a certain type of policy analysis, our approach aims at providing an environment in which a variety of analysis can be carried out. In particular, our environment is able not only to handle conventional policy property verification and policy comparison, but also to support queries on common portions and different portions of multiple policies.

Unlike all approaches to policy similarity analysis which require extensive comparison between policies, our proposed similarity measure is a lightweight approach which aims at reducing the searching space, that is, at reducing the number of policies that need to be fully examined. From the view of an entire policy analysis system, our policy similarity measure can be seen as a tool which can act as a filter phase, before more expensive analysis tools are applied.

For completeness it is also important to mention that the problem of similarity for documents has been investigated in the information retrieval area. Techniques are thus available for computing similarity among two documents (e.g. [41–43]). However, these cannot be directly applied because of the special structures and properties of the XACML policies.

5.1.3 Visualization

Several visualization techniques to improve the usability of access control policy systems have been proposed in literature. However the main focus of these works has been towards building better interfaces that can aid in more efficient and error free policy authoring and not towards visualizing the results of policy analysis.

Reeder et al [27] have proposed the *Expandable Grids* tool for displaying and authoring policies. They have implemented an interface based on the concept of expandable grids for setting file permission in Windows NTFS file system. The grid is a matrix with subjects along rows, resources along columns, and the effective access for each subject/resource combination in the matrix cells. Expandable grids are useful for authoring policies, but they do not support visualization of policy analysis results. , where since they cannot represent attribute predicates which are necessary for characterizing requests.

Vaniea et al [44] have proposed the *Prismos* system which provides visualization of both policy and analysis results. The interface contains a grid where rows represent attribute predicates and columns represent rules. A side bar displays analysis results, e.g., which rules are conflicting and which are redundant. However, the interface is not useful for

displaying sets of requests, as their visualization would be equivalent to displaying a list of request predicate tuples.

The *SPARCLE Policy Workbench* by Brodie et al [45] allows policy authors to express policies in natural language, which are subsequently parsed to create machine-readable policies. SPARCLE also supports a table-based policy visualization, but the representation is textual and mainly applicable to privacy policies. Furthermore, SPARCLE does not support visualization of results from conflict analysis.

5.2 Policy Integration

In the literature, many efforts have been devoted to policy composition [28, 29, 39, 46–49]. Few approaches have been proposed for dealing with the fine-grained integration of XACML policies. Approaches most closest to ours are by Halpern et al. [48], Mazzoleni et al. [14], Bonatti et al. [28], Wijesekera et al. [29], Backes et al. [46] and Jagadeesan et al. [49].

One early work on policy composition is the policy algebra proposed by Bonatti et al. [28], which aims at combining authorization specifications originating from heterogenous independent parties. They model an access control policy as a set of ground (variable-free) authorization terms, where an authorization term is a triple of the form (subject, object, action). However, their algebra only supports 2-valued policies and they do not clearly point out what authorization specifications can be expressed and what cannot by using their algebra. Regarding the algebra implementation, they suggest to use logic programming, but do not show any experimental result. Compared to their work, we have proved that our algebra can express any possible policy integration requirement and our implementation is based on representations used in model checking techniques which have been proven to be very efficient.

Halpern et al. [48] have used first-order logic to specify policies and support the composition of policies in this context. Only those policies which do not have any conflicts when logically combined can be composed using their framework. In contrast, our work defines

operators that can be used to resolve any potential conflicts that may arise when performing the composition. Mazzoleni et al. [14] have proposed an extension to the XACML language, called *policy integration preferences*, using which a party can specify the approach that must be adopted when its policies have to be integrated with policies by other parties. They do not discuss mechanisms to perform such integrations. Also, the integration preferences discussed in such work are very limited and do not support fine-grained integration requirements. Wijesekera et al. [29] have proposed a propositional algebra for access control. They model policies as nondeterministic transformers of permission set assignments to subjects and interpret operations on policies as set-theoretic operations on the transformers. They make use of *signed* action terms to represent negative authorizations. Backes et al. [46] have proposed an algebra for combining enterprise privacy policies. They define conjunction, disjunction and scoping operations on 3-valued EPAL [50] policies. Unlike our work which can handle 3-valued policies, the above works do not explicitly support negative authorizations. Jagadeesan et al. [49] have proposed a 3-valued policy algebra in the timed concurrent constraint programming paradigm and define boolean operators whose expressive power is equivalent to the algebra in [28] in addition to added temporal features. However, unlike our work none of the above works study the completeness and minimality of their algebras.

Most recently, Bruns et al. [51] proposed an algebra for four-valued policies based on Belnap bilattice. In particular, they map four possible policy decisions, i.e. grant, deny, conflict and unspecified, to Belnap bilattice and claim that their algebra is complete and minimal. However, such completeness is limited to Belnap space where policy decisions need to follow certain order according to the Belnap bilattice. Moreover, they did not propose any implementation of their algebra.

Our work is also related to the area of *many* valued logics. Most of these works focus on establishing criteria for *Sheffer functions* in *m-valued* logic. A Sheffer function can be defined as a logical function that is complete. Martin [52] isolates all binary sheffer functions in 3-valued logic and proves properties of such functions. Wheeler [53] proves a generalization of [52] and establishes the necessary and sufficient conditions for *n-nary*

Sheffer functions in the context of 3-valued propositional calculus. Rousseau [54] provides further generalization and proves the necessary and sufficient conditions for any finite algebra with a single operation to be complete. Haddad et al. [55] characterize binary and ternary *partial Sheffer functions* in 3-valued logic. Arieli et al. [56] propose a propositional language with four-valued semantics and study in detail the expressive power of their language. They also compare 3-valued and 4-valued formalisms. In contrast to these works, we do not find or establish criteria for all possible complete operators or functions for a 3-valued algebra. Instead, we focus on the definition of a set of operators that have intuitive semantic meaning in the context of combining 3-valued policies and study whether this set of operators is complete. We also study properties such as expressive power and minimality for this set of operators.

6 CONCLUSIONS AND FUTURE WORK

The use of policy based security management in distributed collaborative applications and architectures has led to the proliferation of policies. A direct consequence of this is the need for tools and techniques to manage and consolidate a large set of policies.

In this thesis we have proposed EXAM, a comprehensive environment for the analysis and management of access control policies. We consider policies expressed using XACML (Extensible Access Control Markup Language) [1] because XACML is a rich language which can represent many policies of interest to real world applications and is gaining widespread adoption in the industry. We identified and defined three types of basic policy analysis queries which can be combined to perform various other advanced analyses.

We have proposed a novel policy similarity measure which can be used as a filter approach in policy comparison. The policy similarity measure represents a lightweight approach to quickly analyze similarity of two policies. Detailed algorithms of computation of similarity scores are presented. To the best of our knowledge ours is the first work to introduce the notion of a *similarity score* for access control policies. We have extended the basic policy similarity measure computation to incorporate a new *semantic matching* score for attributes and values that do not exactly match using ontology matching techniques and WordNet lexical database [15] thus solving the problem of attribute name and value heterogeneity when comparing policies for similarity. We have also incorporated the notion of *predicate selectivity* to improve the effectiveness of the similarity score. We have implemented all variations of the similarity score computation and reported experimental results that demonstrate the efficiency and effectiveness of these approaches. We have also conducted a pilot study among system adminstrators and students to validate the practical value of our proposed similarity measure. Encouraged by the positive results of the pilot study we are currently conducting a survey with larger number of participants including information security executives.

We have proposed a policy similarity analyzer that combines the advantages of MTBDD based model checking and SAT-solver based techniques to provide a precise characterization of the set of requests permitted, denied or not-applicable to the policies being analyzed. The experimental results obtained from the prototype implementation of the analyzer demonstrate the efficiency and scalability of the proposed approach. We have also proposed and implemented a novel multi-level grid visualization technique to visualize the results of policy analysis.

We have proposed an algebra for the fine-grained integration of language independent 3-valued policies. Our operations can not only express existing policy-combining algorithms but can also express any arbitrary combination of policies at a fine granularity of requests, effects and domains, as we have proved in the completeness theorem. We have also proved algebraic and minimality properties for the algebra. Based on this algebra, we have proposed and implemented a framework for integration of XACML policies. The framework generates XACML policies as the policy integration result. We present experimental results which demonstrate the efficiency of our integration approach.

Finally, we have incorporated all the above policy analysis and integration techniques and have developed a functional prototype of EXAM. EXAM is a web-based application and is available in a virtual appliance for use.

The current thesis opens several interesting directions for future research:

- Policy authoring tools [27, 44, 45] are an important part of today's policy based security landscape. However these tools do not focus on the policy analysis aspects. An interesting direction of research would be to explore the integration of such policy authoring tools with policy analysis tools like EXAM. Such an integration would help policy authors to leverage the results of policy analysis during the process of policy authoring, thus leading to the development of a consistent and error free policy repository.

- Policy similarity measure techniques can be used to compare several other types of policies as discussed in Section 3.1.4. For example, it can be used to define a distance

function for Platform for Privacy Preferences(P3P) [57] policies. The distance function can then be used for clustering these P3P policies and deriving a meta-policy that can serves as a representative P3P policy for individual business domains (like bookstores, apparels) across the internet.

- Currently the policy similarity analyzer supports the comparison of two policies at a time. However it would be interesting to extend the current technique to compare more than two policies at once. This is particularly useful in a scenario where large number of policies need to be integrated in a time efficient manner. Preliminary experiments have indicated that the order in which the policies are compared can result in an order of magnitude difference in the time needed to perform the comparison. It would be interesting to study in detail the cause for such difference and explore optimization techniques to determine the optimum order in which to compare the policies so as to minimize the comparison time.

- An interesting question that arises when using integration algebras such as proposed is: *how can one be sure that a given algebriac expression will behave as expected?*. We believe that software engineering techniques can be leveraged to examine the integrated policy with a random number of requests and provide assurance with some high probability that the expression indeed corresponds to the expected behaviour.

LIST OF REFERENCES

LIST OF REFERENCES

[1] T. Moses. Extensible access control markup language (XACML) version 2.0. *Technical report, OASIS*, 2005.

[2] M. Blaze, J. Feigenbaum, and M.Strauss. Compliance checking in the policymaker trust management system. In *Proceedings of the International Conference on Financial Cryptography*, pages 254 – 274, 1998.

[3] ISO 10181-3 access control framework.

[4] Extensible access control markup language (XACML) version 2.0, 2005.

[5] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 196–205, 2005.

[6] Parthenon XACML evaluation engine.

[7] Sun's XACML open source implementation.

[8] P. Rao, D. Lin, and E. Bertino. XACML function annotations. In *IEEE Workshop on Policies for Distributed Systems and Networks*, 2007.

[9] D. Lin, P. Rao, E. Bertino, and J. Lobo. An approach to evaluate policy similarity. In *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 1 – 10, 2007.

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

[11] M. Baker, K. Kimberly, and M. Sean. Why traditional storage systems do not help us save stuff forever. *HPL-2005-120. HP Labs 2005 Technical Reports*, 2005.

[12] D. Morr. Lionshare: A federated p2p app. In *Internet2 members meeting*, 2007.

[13] D. Agrawal, J. Giles, K. W. Lee, and J. Lobo. Policy ratification. In *Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 223–232, 2005.

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

[15] WordNet. http://wordnet.princeton.edu/.

[16] N. Choi, I. Song, and H. Han. A survey on ontology mapping. *Special Interest Group on Management Of Data (SIGMOD)*, 35(3):34–41, 2006.

[17] A. Schaad, J. D. Moffett, and J. Jacob. The role-based access control system of a european bank: a case study and discussion. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 3–9, 2001.

[18] Falcon. http://iws.seu.edu.cn/projects/matching/.

[19] swportal. http://sw-portal.deri.org/ontologies/swportal.

[20] swrc_update. http://ontoware.org/frs/download.php/354/swrc_updated.

[21] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. EXAM – A comprehensive environment for analysis of access control policies. In *Proceedings of the International Journal of Information Security (IJIS)*, 2010.

[22] SELinux. http://oss.tresys.com/projects/refpolicy.

[23] Sediff. http://oss.tresys.com/projects/setools/wiki/wikistart#secmds.

[24] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient datastructure for matrix representation. *Formal Methods in System Design*, 10(2-3):149–169, 1997.

[25] J. E. Hopcroft and J. D. Ullman. Introduction to automata theory, languages and computation. *Addison Wesley*, 1979.

[26] http://fke.utm.my/downloads/espresso/.

[27] R. Reeder, L. Bauer, L. Cranor, M. Reiter, K. Bacon, K. How, and H. Strong. Expandable grids for visualizing and authoring computer security policies. In *CHI '08: Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems*, 2008.

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

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

[30] A. Anderson. Evaluating XACML as a policy language. *Technical report, OASIS*, 2003.

[31] O. Grumberg, S. Livne, and S. Markovitch. Learning to order BDD variables in verification. *Journal of Artificial Intelligence Research*, 18:83–116, 2003.

[32] T. Ahmed and A. R. Tripathi. Static verification of security requirements in role based cscw systems. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 196–203, 2003.

[33] D. P. Guelev, M. Ryan, and P. Schobbens. Model-checking access control policies. In *Proceedings of the 7th Information Security Conference (ISC)*, pages 219–230, 2004.

[34] N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In *Proceedings of the 8th Information Security Conference (ISC)*, pages 446–460, 2005.

[35] M. Backes, G. Karjoth, W. Bagga, and M. Schunter. Efficient comparison of enterprise privacy policies. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)*, pages 375–382, 2004.

[36] M. Koch, L. V. Mancini, and F. P. Presicce. On the specification and evolution of access control policies. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 121–130, 2001.

[37] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering (TSE)*, 25(6):852–869, 1999.

[38] J. D. Moffett and M. S. Sloman. Policy conflict analysis in distributed system management. *Journal of Organizational Computing*, 1993.

[39] P. McDaniel and A. Prakash. Methods and limitations of security policy reconciliation. *ACM Transactions on Information and System Security (TISSEC)*, 9(3):259 – 291, 2006.

[40] V. Kolovski, J. Hendler, and B. Parsia. Analyzing web access control policies. In *Proceedings of the International World Wide Web Conference*, page 677, 2007.

[41] M. Ehrig, P. Haase, M. Hefke, and N. Stojanovic. Similarity for ontologies – A comprehensive framework. In *Proceedings of the 13th European Conference on Information Systems, Information Systems in a Rapidly Changing Economy (ECIS)*, 2005.

[42] T. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science and Technology*, 54(3):203–215, 2003.

[43] D. Metzler, Y. Bernstein, W. B. Croft, A. Moffat, and J. Zobel. Similarity measures for tracking information flow. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 517–524, 2005.

[44] K. Vaniea, Q. Ni, L. Cranor, and E. Bertino. Access control policy analysis and visualization tools for security professionals. In *USM'08: Workshop on Usable IT Security Management*, 2008.

[45] C. Brodie, C. Karat, and J. Karat. An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench. In *SOUPS '06: Proceedings of the Second Symposium on Usable Privacy and Security*, 2006.

[46] M. Backes, M. Duermuth, and R. Steinwandt. An algebra for composing enterprise privacy policies. In *Proceedings of 9th European Symposium on Research in Computer Security(ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 33–52. Springer, September 2004.

[47] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[48] J. Halpern and V. Weissman. Using first-order logic to reason about policies. *In Proceedings of the Computer Security Foundations Workshop (CSFW'03)*, 2003.

[49] R. Jagadeesan, W. Marrero, C. Pitcher, and V. Saraswat. Timed constraint programming: a declarative approach to usage control. In *PPDP '05: Proceedings of the 7th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 164–175, New York, NY, USA, 2005. ACM.

[50] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (EPAL). *Research Report 3485, IBM Research*, 2003.

[51] G. Bruns, D. S. Dantas, and M. Huth. A simple and expressive semantic framework for policy composition in access control. In *Proceedings of the 5th ACM Workshop on Formal Methods in Security Engineering (FMSE)*, 2007.

[52] N. Martin. The Sheffer functions of 3-valued logic. *The Journal of Symbolic Logic*, 19(1):45–51, 1954.

[53] R. Wheeler. Complete connectives for 3-valued propositional calculus. *Proceedings of London Mathematical Society*, 3(16):167–191, 1966.

[54] G. Rousseau. Completeness in finite algebras with a single operation. *Proceedings of the American Mathematical Society*, 18(6):1009–1013, 1966.

[55] L. Haddad and D. Lau. Characteriztion of partial Sheffer functions in 3-valued logic. *Proceedings of the 37th International Symposium on Multiple-Valued Logic(ISMVL'07)*, 2007.

[56] O. Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[57] M. Marchiori M. Presler-Marshall J. Reagle L. Cranor, M. Langheinrich. The platform for privacy preferences 1.0 (p3p1.0) specification. *W3C Recommendation*, 2002.

APPENDIX

A POLICY SIMILARITY SURVEY QUESTIONNAIRE

In the following survey, we would like to ask you three questions about the Company X. This company is in retail business and works with a network of partners, suppliers, and customers. An information security breach directed at any one link, for instance, a distribution center that manages inventory and shipments electronically, can affect the entire chain. In the following survey, we ask you about the similarity and appropriateness of information security policies that Company X may implement to protect its information system infrastructure. In questions about similarity, we ask your opinion about how similar are these pairs of policies in the scale of 1-7 (1 is not similar at all, and 7 is very similar). In questions about appropriateness, we ask your opinion about the appropriateness of these policies in the scale of 1-7 (1 is not appropriate at all, and 7 is very appropriate).

1) Company X can collect, and use customers personal sensitive and non-sensitive data on a need-to-know basis.

1.2) Company X can collect, and use, customers personal sensitive and non-sensitive data. This information may be used to provide discounts to customers.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 1:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 1.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

2) Company X can store customers personal sensitive and non-sensitive data for a limited time– as it is needed for the stated purposes.

2.2) Company X can store customers personal information for an unlimited time. This information may benefit customers networking purposes.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 2.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

3) Opt-in and opt-out require each individual's explicit, informed consent.

3.2) Only Opt-in requires each individual's explicit, informed consent.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 3:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 3.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

4) Policies must be explicit, when they deal with personal sensitive and non-sensitive data.

4.2) Policies must be explicit only when they deal with personal non-sensitive data.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 4:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 4.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

5) Users can access and modify their stored personal sensitive and non-sensitive data.

5.2) Users can access their stored personal sensitive and non-sensitive data, only if they can fulfill training requirements.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 5:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 5.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

6) Third parties cannot access personal sensitive and non-sensitive data of the customers.

6.2) Third parties can access personal data if it can help customers networking purposes.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 6:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 6.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

7) All Internet-connected computers must be running an intrusion detection system approved by the Information Security Department.

7.2) All Internet-connected computers with personal and sensitive data must be running an intrusion detection system approved by the Information Security Department.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 7:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 7.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

8) All Web servers accessible via the Internet must be protected by a router or firewall approved by the Information Security Department.

8.2) Web servers which deal with sensitive data must be protected by a router or firewall approved by the Information Security Department.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 8:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

9) All Internet commerce servers must employ unique digital certificates and must use encryption to transfer information in and out of these servers.

Appropriateness of Policy 8.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

9.2) Internet commerce servers which run personal and sensitive data servers must employ unique digital certificates and must use encryption to transfer information in and out of these servers.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 9:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 9.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

10) Employees must not construct passwords which are identical or substantially similar to passwords that they had previously used.
10.2) Employees who have access to sensitive data must not construct passwords which are identical or substantially similar to passwords that they had previously used.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 10:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 10.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

11) All employees must be automatically forced to change their passwords at least once every ninety days.

11.2) Employees who have access to sensitive data must be automatically forced to change their passwords at least once every ninety days.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 11:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 11.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

12) Employees should have no expectation of privacy in anything they store, send or receive on the company's email system. The company may monitor messages without prior notice.

12.2) Employees and customers should have no expectation of privacy in anything they store, send or receive on the company's email system. The company may monitor messages without prior notice.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 12:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 12.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

13) Instant message conversations of employees, who have access to private and sensitive data, that are Administrative or Fiscal in nature should be copied into an email message and sent to the appropriate email retention address.

13.2) Instant message conversations of employees and customers that are Administrative or Fiscal in nature should be copied into an email message and sent to the appropriate email retention address.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 13:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 13.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

14) All employees must refrain from using the same password on multiple computer systems.

14.2) Employees, who have access to personal and sensitive data, must refrain from using the same password on multiple computer systems.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 14:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 14.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

15) At log-in time, every employee must be given information reflecting the last log-ins time.

15.2) At log-in time, every employee, who has access to personal and sensitive data, must be given information reflecting the last log-ins time.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 15:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 15.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

16) Personal electronic devices cannot be used for companys sensitive information, unless they have first been configured with the necessary controls by Information Security Dept.

16.2) Personal electronic devices can be used for companys non sensitive information.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 16:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 16.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

17) All employees with computers which have remote real-time dialogue with companys network must run access control package approved by the Information Security Dept.

17.2) Employees, who have access to private and sensitive data, and have computers which

have remote real-time dialogue with companys network must run access control package approved by the Information Security Dept.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 17:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 17.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

18) When an employees working relationship is terminated with the company, all access rights to the companys data must be immediately revoked.

18.2) When an employees working relationship is terminated with the company, all access rights to the companys personal and sensitive data must be immediately revoked.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 18:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 18.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

19) The Information Security Dept. should be able to control user access to all objects on the systems according to the stated policy.

19.2) The Information Security Dept. should be able to control user access to sensitive data objects according to the stated policy.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 19:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 19.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

20) All employees should be allowed to label data with a classification.

20.2) Only employees of Information Security Dept. should be allowed to label data with a classification.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 20:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 20.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

21) Company X maintains and owns backup tapes from the email servers.

21.2) Company X may share maintaining and owning backup tapes from the e-mail servers with other companies.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 21:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 21.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

22) The responsibility for Information security and privacy is every employees duty.

22.2) The Responsibility for information security and privacy is only in the Information Security Department.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 22:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 22.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

23) All the security and privacy issues should be handled by internal Information Security Department.

23.2) Security and privacy issues can be handled with contractors and outside companies– This saves cost for the company and customers.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 23:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 23.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

24) Employees of the Information Security Department should comply with security policy and procedures.

24.2) All employees in the Company X should comply with security policy and procedures. This may cause additional cost to company and to customers.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 24:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 24.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

25) To mitigate risks, Company X should purchase cyber insurance and pay premium.

25.2) Company X should purchase cyber insurance and pay premium. Portions of premium should be paid by customers.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 25:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 25.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

26) Company X should spend its information security budget on technical vulnerabilities.

26.2) Company X should spend its information security budget on technical vulnerabilities, organizational awareness, and trainingthe people component of information security technology.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 26:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 26.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

27) Security measures should be applied to all electronic and physical (printouts, microfiche) copies of personal sensitive and non-sensitive data.

27.2) Security measures only need to be applied to electronic and physical (printouts, microfiche) copies of personal sensitive data. To provide security measures for non-sensitive data customers must enrollee in special security plans.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 27:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 27.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

28) In outsourcing, the service provider can share decisions with the company about who will be granted access to companys information and information systems.

28.2) In outsourcing, Company X is the only one that can make decisions about who will be granted access to companys information and information systems.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 28:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 28.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

29) Employees who deal with sensitive and personal data should receive a minimum security training before they can begin work.

29.2) All new employees should receive a minimum of security training before they can begin work– This may cause additional cost for customers and company.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 29:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 29.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

30) Company X is responsible for establishing a process for responding to security incidents.

30.2) Company X should follow standardized processes for responding to security incidents.

Similarity of the two policies:

Not similar at all	1	2	3	4	5	6	7	Very similar

Appropriateness of Policy 30:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

Appropriateness of Policy 30.2:

Not appropriate at all	1	2	3	4	5	6	7	Very appropriate

VITA

VITA

Prathima Rao was born in southern India in 1979 . She received her B.E degree in Computer Science at P.E.S. Institute of Technology in Bangalore, India in 2000. After working as a Software Engineer at IBM India for a year, she headed to USA to pursue her post graduate studies at Purdue University. She obtained her master's degree from the Department of Electrical and Computer Engineering at Purdue in 2003. After working as a research assistant for a year in the Computer Sciences deparment, she started pursuing her PhD degree in the fall of 2004. She successfully defended her PhD thesis in the summer of 2010 and received her degree that year. She looks forward to pursing a career in the software industry as a security consultant.