A Similarity Measure for Comparing Access
Control Policies

PRATHIMA RAO and DAN LIN and RODOLFO FERRINI and ELISA BERTINO
Department of Computer Science, Purdue University, USA
{prao,lindan,rferrini,bertino}@cs.purdue.edu
and
JORGE LOBO
IBM T.J. Watson Research Center
jlobo@us.ibm.com

Recent collaborative applications and enterprises very often need to efficiently integrate their
access control policies. An important step in policy integration is to analyze the similarity of
policies. Existing approaches to policy similarity analysis are mainly based on logical reasoning
and Boolean function comparison. Such approaches are computationally expensive and do not
scale well for large heterogeneous distributed environments (like Grid computing systems). In this
paper, we propose a policy similarity measure as a filter phase for policy similarity analysis. This
measure provides a lightweight approach to pre-compile a large amount of policies and only return
the most similar policies for further evaluation. In this paper we formally define the measure by
taking into account both the case of categorical attributes and numeric attributes. We solve the
problem of name heterogeneity when comparing policies by using ontology matching techniques.
Detailed algorithms are presented for the similarly computation. We also present experimental
results which demonstrate the efficiency and practical value of our approach.

Categories and Subject Descriptors: D.4 OPERATING SYSTEMS [D.4.6 Security and Pro-
tection]: Access controls

General Terms: Algorithms, Measurement, Design

Additional Key Words and Phrases: Policy similarity measure, XACML policies

1. INTRODUCTION

A key goal for collaborative applications is to share resources, such as services, data
and knowledge. Such applications may have different objectives, such as provision-
ing 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 [Mazzoleni et al. 2006]. 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.

A brute force approach 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. Therefore, two recent approaches have been proposed to address the problem of policy comparison. The first approach is based on model checking [Fisler et al. 2005], whereas the second is based on logical reasoning [Agrawal et al. 2005]. However, both these approaches are computationally expensive, especially when dealing with large scale heterogeneous distributed environments and time constraints. The problem is equivalent to solving Boolean satisfiability, which is NP-complete. In this paper, we thus propose an alternative approach, based on principles from the information retrieval field. Our approach uses the notion of policy similarity measure, based on which a similarity score can be quickly computed for two policies. Specifically, if the similarity score of policies $P_1$ and $P_2$ is higher than that of 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 policy similarity measure can serve as a filter before applying any additional logical reasoning or Boolean function comparison. It provides a useful lightweight approach to pre-compile a set of policies and return the most similar policies for further exploration. Such exploration could be a fine-grained policy analysis which determines the common and different parts of two policies; it can also be a visualization phase in which users can easily identify similar policies and make their own decisions. Our approach parallels the quick filter approach adopted for multimedia data querying. In multimedia data querying, a first quick filtering
of the data is executed; such filtering phase discards the data that are certainly not part of the query reply. The data that are not discarded by the quick filter are then analyzed by using specialized algorithms to determine the actual query results.

Our current work focuses on a similarity measure for policies written in XACML (Extensible Access Control Mark-up Language) [Moses 2003], because of its widespread adoption. The similarity measure takes into account the policy structure typical of XACML. Given two policies, our algorithm for computing the similarity score first groups the same components of the two input policies, and evaluates their similarity by using hierarchy distance and numerical distance. Then the scores obtained for the different components of the policies are combined according to a weighted combination in order to produce an overall similarity score. However, because policies being compared may belong to different organizations each of which may use a different vocabulary of attribute names and values to express their policies, name heterogeneity can occur. We thus also address such problem by enhancing our similarity measure with dictionary lookup and ontology matching techniques. Our experimental results indicate that our approach can efficiently identify similar policies.

A preliminary version of this work appears in [Lin et al. 2007a], where we presented the basic idea of a policy similarity measure. In this paper, we make the following additional contributions. First, we have developed a major extension to the similarity technique. The new similarity technique addresses the problem of name heterogeneity among policies; this is an important issue when dealing with access control policies by different organizations or administrative domains. Second, we have carried out experimental studies to evaluate both efficiency and effectiveness of our approach. Various versions of the similarity techniques have been evaluated. Compared to an accurate policy similarity analyzer, our approach can detect similar policies at two to three orders of magnitude lesser time. Third, we have discussed a list of applications which can benefit from our techniques. This demonstrates the potential application of our approach in a wide range of scenarios.

The rest of the paper is organized as follows. Section 2 introduces preliminary notions concerning XACML. Section 3 introduces the proposed policy similarity measure and the algorithm for computing the similarity. Section 4 discusses the properties of the similarity measure and a case study. Section 5 presents the experimental results. Section 7 and Section 8 discuss related works on access control policy analysis and applications of the proposed policy similarity measure. Finally, Section 9 concludes the paper and outlines future work.

2. PRELIMINARY NOTIONS AND AN ILLUSTRATIVE EXAMPLE

An XACML policy [Moses 2003] consists of three major components, namely a Target, a Rule set, and a rule combining algorithm for conflict resolution. The Target specifies some predicates on the attribute values in a request, which will be satisfied in order for the policy to be applicable to the request. The attributes in the Target element are categorized into Subject, Resource and Action. A Rule set consists of one or more Rules. Each Rule in turn consists of Target, Condition and Effect elements. The rule Target has the same structure of the policy Target. The only difference is that the rule Target specifies the situation when the rule can be
applied. A *Condition* element specifies some restrictions on request attribute values that must be satisfied in order to yield a *Permit* or *Deny* decision as specified by the *Effect* element. Figure 1 gives an overview of a policy structure.

As an example that we will use throughout the paper, 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

---

```xml
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 ≤ 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 ≤ Time ≤ 21:00>
  </Rule>
```

---

Fig. 2. Data Owner Policy $P_1$

ACM Journal Name, Vol. V, No. N, Month 20YY.
PolicyId = P2

<PolicyTarget>
  <Subject GroupName belong_to {IBMOpenCollaboration, IntelOpenCollaboration}>
  </PolicyTarget>
</Policy>

<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>

Fig. 3. Resource Owner Policy P2

or write files of size less than 120MB. P2 gives a special 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, P2 does not allow any user to operate on media files on the machine. P3 is an access control policy for another machine, mainly used by business staff. P3 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 2, 3 and 4 report the XACML specification for these policies.

From a user’s perspective, P1 is more similar to P2 than P3 because most activities described by P1 for the data owner are allowed by P2. Our motivation is to quickly compute similarity scores S1 between P1 and P2, and S2 between P1 and P3, where we would expect that S1 be larger than S2 to indicate that the similarity between P1 and P2 is much higher than the similarity between P1 and P3.
3. 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 5 gives the syntax of the simplified format.

We would like the policy similarity measure between any two given policies to assign a similarity score \( S_{\text{policy}} \) that approximates the percentage of the requests obtaining the same decisions (permitted or denied) from the two policies. The definition is given in Equation 1, where \( N_{\text{srreq}} \) denotes the requests with the same decisions from \( P_1 \) and \( P_2 \) and \( N_{\text{req}} \) is the set of requests applicable to either \( P_1 \) or \( P_2 \).

\[
S_{\text{policy}}(P_1, P_2) \approx \frac{N_{\text{srreq}}}{N_{\text{req}}}
\]  

(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.

3.1 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.
POLICY :
<policy policy-id = "policy-id" combining-algorithm = "combining-algorithm">
  <targetelement>?</p
  <permitrules>
    <ruleelement>*
  </permitrules>
  <denyrules>
    <ruleelement>*
  </denyrules>
</policy>

RULE ELEMENT :
<rule rule-id="rule-id" effect="rule-effect">
  <targetelement>?</p
  <condition>Predicate</condition>
</rule>

TARGET ELEMENT :
<target>
  <subject>Predicate</subject>
  <resource>Predicate</resource>
  <action>Predicate</action>
</target>

PREDICATE :
(attr_name ⊕ (attr_value)+)*

attr_name denotes attribute name, attr_value denotes attribute value
and ⊕ denotes any operator supported by the XACML standard.

Fig. 5. Simplified XACML format

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 com-
bining algorithm. In particular, consider a rule $R$ with a condition ($c_1$ OR $c_2$ OR ...
OR $c_k$), where $c_i$ (1 ≤ $i$ ≤ $k$) can be any arbitrary 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.

3.2 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 $\Phi$) for each rule in the two policies. For clarity, we choose to use four separate
$\Phi$ mappings $\Phi^P_1$, $\Phi^D_1$, $\Phi^P_2$ and $\Phi^D_2$. The mapping $\Phi^P_1(\Phi^D_1)$ 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 $\Phi^P_2(\Phi^D_2)$
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 \( \Phi \) mappings give similar rules in \( P_2(P_1) \) which satisfy certain similarity threshold. The computation of the \( \Phi \) mapping will be addressed in the Section 3.3.

By using the \( \Phi \) 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 \( \Phi \) mapping. \( rs_{1i} \) and \( rs_{2j} \) are computed according to Equations 2 and 3, where \( S_{\text{rule}} \) 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 \( S^P_{\text{rule-set}} \) and \( S^D_{\text{rule-set}} \) 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 2 and 3) for all rules in the set. The permit and deny rule set similarity scores are formulated by Equation 4 and 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.

\[
rs_{1i} = \begin{cases} 
\sum_{r_j \in \Phi^P_1(r_{1i})} \frac{S_{\text{rule}}(r_{1i}, r_j)}{|\Phi^P_1(r_{1i})|}, & r_{1i} \in PR_1 \\
\sum_{r_j \in \Phi^D_1(r_{1i})} \frac{S_{\text{rule}}(r_{1i}, r_j)}{|\Phi^D_1(r_{1i})|}, & r_{1i} \in DR_1 
\end{cases}
\]  

(2)

\[
rs_{2j} = \begin{cases} 
\sum_{r_i \in \Phi^P_2(r_{2j})} \frac{S_{\text{rule}}(r_{2j}, r_i)}{|\Phi^P_2(r_{2j})|}, & r_{2j} \in PR_2 \\
\sum_{r_i \in \Phi^D_2(r_{2j})} \frac{S_{\text{rule}}(r_{2j}, r_i)}{|\Phi^D_2(r_{2j})|}, & r_{2j} \in DR_2 
\end{cases}
\]  

(3)

\[
S^P_{\text{rule-set}} = \frac{\sum_{i=1}^{N_{PR_1}} rs_{1i} + \sum_{i=1}^{N_{PR_2}} rs_{2j}}{N_{PR_1} + N_{PR_2}}
\]  

(4)

\[
S^D_{\text{rule-set}} = \frac{\sum_{i=1}^{N_{DR_1}} rs_{1i} + \sum_{i=1}^{N_{DR_2}} rs_{2j}}{N_{DR_1} + N_{DR_2}}
\]  

(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, \( S_{\text{policy}} \). The formulation of \( S_{\text{policy}} \) is given by the following equation:

\[
S_{\text{policy}}(P_1, P_2) = w_T S_T(P_1, P_2) + w_p S_{\text{rule-set}}^P + w_d S_{\text{rule-set}}^D
\]

(6)

where \( S_T \) is a function that computes a similarity score between the Target elements of any two given policies; \( w_p \) and \( w_d \) are weights that can be chosen to reflect the relative importance to be given to the similarity of permit and deny rule sets respectively. For normalization purpose, the weight values should satisfy the constraint: \( w_T + w_p + w_d = 1 \). The weight value introduced here and in the later part aims to provide more flexibility for users to locate desired policies. For example, if one aims to find policies applicable to the same domain but do not care the decisions, he can assign a higher value to the policy target to achieve this purpose. Without any preference, equal weight values are assigned by default. An example is given in Section 4.2 (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 \( \Phi \) mappings and rule similarity score \( S_{\text{rule}} \). Table I lists main notations used throughout the paper.

3.3 Computation of \( \Phi \) Mappings

The one-many \( \Phi \) 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 \( \Phi \) mapping as follows:

\[
\Phi(r_i) = \{ r_j | S_{\text{rule}}(r_i, r_j) \geq \epsilon \}
\]

(7)

where \( S_{\text{rule}} \) is computed by Equation 8 and \( \epsilon \) is a threshold. The threshold term allows us to calibrate the quality of the similarity approximation and is suggested to set close to 0.75 as discussed later in Section 4.1. An example of the computation of a \( \Phi \) mapping is shown by steps 3 and 4 in Section 4.2.

The general \( \Phi \) 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 rule combining algorithm, the \( \Phi \) mapping for \( r_i \) will contain the only \( r_j \) which is the first rule with the similarity score \( S_{\text{rule}}(r_i, r_j) \) above the threshold \( \epsilon \). 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 \( \Phi \) 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.

Finally, we would like to point out that the mapping helps capture more similar
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\text{-rule-set}} | Similarity score of two sets of permit rules
S_{D\text{-rule-set}} | Similarity score of two sets of deny rules
S_{\langle Element\rangle} | Similarity score of elements, \(\langle Element\rangle \in \{T',{t}',{c}',{s}',{r}',{a}'\}\)
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
\Phi | 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
\text{SPath}_{\langle Element\rangle} | Length of shortest path of two categorical values
w_{\langle Element\rangle} | Weight of similarity scores of elements, \(\langle Element\rangle \in \{T',{t}',{c}',{s}',{r}',{a}'\}\)
\epsilon | Rule similarity threshold
\delta | Compensating score for unmatched values

Table I. Notations

policies, but there may exist false positive cases when two policies with a relatively high score are actually different (an example is shown in Section 4.1).

3.4 Similarity Score between Rules

Since our similarity measure serves as a lightweight filter phase, 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 \(S_{\text{rule}}\) between two rules \(r_i\) and \(r_j\) as follows:

\[
S_{\text{rule}}(r_i, r_j) = w_t S_t(r_i, r_j) + w_c S_c(r_i, r_j)
\]  

(8)

\(w_t\) and \(w_c\) are weights that can be used for emphasizing the importance of the 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:

\[
S_t(r_i, r_j) = w_s S_s(r_i, r_j) + w_r S_r(r_i, r_j) + w_a S_a(r_i, r_j)
\] (9)

In Equation 9, \( w_s, w_r, w_a \) represent weights that are assigned to the corresponding similarity scores. Like in the previous equations, weight values need to satisfy the constraint \( w_s + w_r + w_a = 1 \). \( S_s, S_r \) and \( S_a \) are functions that return a similarity score based on the Subject, Resource and Action attribute predicates respectively in the Target elements of the two given rules.

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

### 3.5 Similarity Score of Rule Elements

Each of the rule elements Subject, Resource, Action and Condition is represented as a set of predicates in the form of \{attr\_name\_1 \oplus \_attr\_value\_1, attr\_name\_2 \oplus \_attr\_value\_2, ...\}, where attr\_name denotes the attribute name, \( \oplus \) denotes a comparison operator and attr\_value represents an attribute value.

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 \( S_{(Element)} \), where \( (Element) \) refers to condition, subject, resource or action. The \( S_{(Element)} \) 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 $S_{\langle \text{Element} \rangle}$ is given by Equation 10.

\[
S_{\langle \text{Element} \rangle}(r_i, r_j) = \begin{cases} 
\sum_{(a_{1k}, a_{2l}) \in \mathcal{M}_a} S_{\langle \text{attr\_typ} \rangle}(a_{1k}, a_{2l}) \bigg/ \max(N_{a_1}, N_{a_2}), & N_{a_1} > 0 \text{ and } N_{a_2} > 0; \\
1, & \text{otherwise.}
\end{cases}
\] (10)

In Equation 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; $S_{\langle \text{attr\_typ} \rangle}$ 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. $S_t$.

### 3.5.1 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 $s_{\text{cat}}$) is then defined according to the shortest path of these two values in the hierarchy. The formal definition is shown below:

\[
s_{\text{cat}}(v_1, v_2) = 1 - \frac{SPath(v_1, v_2)}{2H}
\] (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 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 $s_{\text{cat}}$ will be obtained.

Figure 6 gives an example hierarchy, where each node represents a categorical value (specific values are given in Figure 11). The height of the hierarchy is 3, and the length 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 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 \( (v_{1k}, v_{2l}) \) and a compensating score \( \delta \) (for non-matching attribute values). Obviously, there could be many combinations of pairs. Our task is to find a set of pairs (denoted as \( M_v \)) which have the following properties:

1. If \( v_{1k} = v_{2l} \), then \( (v_{1k}, v_{2l}) \in M_v \).
2. For pairs \( v_{1k} \neq v_{2l} \), pairs contributing to the maximum sum of similarity scores belong to \( M_v \).
3. Each attribute value \( v_{1k} \) or \( v_{2l} \) occurs at most once in \( M_v \).
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 7 for an example of these values for the example hierarchy from Figure 6. Then we compute the similarity between pairs of attribute values with the help of the hierarchy code. Figure 8 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 $\delta$ is for attribute values which do not have matchings when two attributes have different number of values. $\delta$ 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 $\delta$ is $(0.33+0.67+0.17)/3 = 0.39$. Finally, the similarity score for the two attribute $a_1$ and $a_2$ 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:

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

(12)

$$\delta = \begin{cases} \sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} \frac{s_{\text{cat}}(v_{1k}, v_{2l})}{N_{v_2}}, & N_{v_2} > N_{v_1}; \\ \sum_{(v_{1k}, v_{2l}) \in \mathcal{M}_v} \frac{s_{\text{cat}}(v_{1k}, v_{2l})}{N_{v_1}}, & N_{v_1} > N_{v_2}. \end{cases}$$

(13)

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

<table>
<thead>
<tr>
<th>Policy $P_1$</th>
<th>Attr</th>
<th>Hcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_{11}$</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>$v_{12}$</td>
<td>1.2.1.1</td>
<td></td>
</tr>
<tr>
<td>$v_{13}$</td>
<td>1.2.1.2</td>
<td></td>
</tr>
<tr>
<td>$v_{14}$</td>
<td>1.3.2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Policy $P_2$</th>
<th>Attr</th>
<th>Hcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_{21}$</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>$v_{22}$</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>$v_{23}$</td>
<td>1.3.2</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 7. Hierarchy Code
Fig. 8. Similarity Score of Two Sets of Attributes

3.5.2 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 14.

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

$$S_{num}(a_1, a_2) = \frac{1}{2} \left[ 1 + \frac{\sum_{(v_{1k}, v_{2l}) \in M_e} s_{num}(v_{1k}, v_{2l}) + \delta}{\max(N_{v1}, N_{v2})} \right]$$ (15)

$$\delta = \begin{cases} \sum_{(v_{1k}, v_{2l}) \in M_e} s_{num}(v_{1k}, v_{2l}) & N_{v1} > N_{v2}; \\ \sum_{(v_{1k}, v_{2l}) \in M_e} s_{num}(v_{1k}, v_{2l}) & N_{v2} > N_{v1}. \end{cases}$$ (16)

3.6 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 $\text{salary} < 10000$ and $\text{salary} > 10000$ 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 3.5 to compute the similarity scores between clusters with matching attribute names. The new similarity score between rule elements is computed by Equation 17.
\[
S_{\text{Element}}^\text{op}(r_i, r_j) = \begin{cases} 
\sum_{(a_{1k}, a_{2l}) \in M_\text{op}} S_{\text{attr}_{\text{yp}}}^\text{op}(a_{1k}, a_{2l}) \over \max(N_{a_1}, N_{a_2}), & N_{a_1} > 0 \text{ and } N_{a_2} > 0; \\
1, & \text{otherwise.}
\end{cases}
\]

Note that Equation 17 now uses a new similarity score given by \(S_{\text{attr}_{\text{yp}}}^\text{op}\) which is defined in Equation 18.

\[
S_{\text{attr}_{\text{yp}}}^\text{op}(a_1, a_2) = \frac{1}{2} \left[ 1 + \sum_{(a_{1k}, a_{2l}) \in M_\text{op}} \frac{S_{\text{attr}_{\text{yp}}}^\text{op}(a_{1k}, a_{2l})}{\max(N_{a_{1p}}, N_{a_{2p}})} \right]
\]  \hspace{1cm} (18)

where \(N_{a_{1p}}\) and \(N_{a_{2p}}\) are the number of unique operators in the predicates corresponding to attributes \(a_1\) and \(a_2\) respectively. \(M_\text{op}\) is a set of pairs of predicates with the same operator and attribute name. Depending on the type of the attributes being compared \(S_{\text{attr}_{\text{yp}}}^\text{op}\) in Equation 18 is substituted with \(S_{\text{cat}}\) or \(S_{\text{num}}\) which can be calculated using equations 12 and 15 respectively.

### 3.7 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 [WordNet], 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 [Choi et al. 2006] on the different ontologies. We will now discuss 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 \(A_O\) [Choi et al. 2006] 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 \(O_1\) a matching concept \(C_j\) in \(O_2\) 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 \((C_i, C_j, m)\). 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 3.5.1 and use Equation 11 to calculate the scores \( s_1 \) and \( s_2 \) for pairs \( P_1 \) and \( P_2 \) respectively. Note that the score \( s_1 \) (or \( s_2 \)) is calculated using Equation 11 only if the matching score \( m_{21} \) (or \( m_{12} \)) 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 \( s_{\text{ONTO}}^{\text{cat}} \) denote the function that computes the semantic score. Equation 19 summarizes the computation of the semantic score.

\[
s_{\text{ONTO}}^{\text{cat}}(v_1, v_2) = \begin{cases} 
  \frac{s_{\text{cat}}(v_1, C_{21}) + s_{\text{cat}}(v_2, C_{12})}{2}, & C_{21}, C_{12} \neq \emptyset \text{ and } m_{21}, m_{12} \geq t_m \\
  0, & C_{21} = \emptyset \text{ or } C_{12} = \emptyset \\
  \frac{s_{\text{cat}}(v_1, C_{21})}{2}, & m_{21} < t_m \\
  \frac{s_{\text{cat}}(v_2, C_{12})}{2}, & m_{12} < t_m 
\end{cases}
\]  

(19)

We then revise Equations 12 and 15 for computing predicate similarity scores to Equations 20 and 21 respectively, by incorporating syntactic and semantic variations.

\[
S_{\text{cat}}(a_1, a_2) = \frac{1}{2} \left[ s_{\text{name}}(n_{a_1}, n_{a_2}) + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}} s_{\text{ONTO}}^{\text{cat}}(v_{1k}, v_{2l}) + \delta}{\max(N_{a_1}, N_{a_2})} \right] 
\]  

(20)

\[
S_{\text{num}}(a_1, a_2) = \frac{1}{2} \left[ s_{\text{name}}(n_{a_1}, n_{a_2}) + \frac{\sum_{(v_{1k}, v_{2l}) \in \mathcal{M}} s_{\text{num}}(v_{1k}, v_{2l}) + \delta}{\max(N_{a_1}, N_{a_2})} \right] 
\]  

(21)

where \( n_{a_1} \) and \( n_{a_2} \) denote the attribute names associated with \( a_1 \) and \( a_2 \) respectively and \( s_{\text{name}}(n_{a_1}, n_{a_2}) \) 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 \( S_{\text{ONTO}}^{\text{cat}}(n_{a_1}, n_{a_2}) \) 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.
Algorithm PolicySimilarityMeasure($P_1, P_2$)

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

1. Categorize rules in $P_1$ and $P_2$ based on their effects. Let $PR_1(\Phi P_2)$ and $DR_1(\Phi D_2)$ 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 $\Phi$ mappings */
   8. $\Phi P_1 \leftarrow \text{ComputePhiMapping}(PR_1, PR_2, \epsilon)$
   9. $\Phi P_2 \leftarrow \text{ComputePhiMapping}(PR_2, PR_1, \epsilon)$
   10. $\Phi D_1 \leftarrow \text{ComputePhiMapping}(DR_1, DR_2, \epsilon)$
   11. $\Phi D_2 \leftarrow \text{ComputePhiMapping}(DR_2, DR_1, \epsilon)$

   /* Compute the rule set similarity scores */
   12. foreach rule $r_{1i} \in P_1$
   13.     if $r_{1i} \in PR_1$ then
   14.         $rs_{1i} \leftarrow \text{ComputeRuleSimilarity}(r_{1i}, \Phi P_1)$
   15.     elsif $r_{1i} \in DR_1$ then
   16.         $rs_{1i} \leftarrow \text{ComputeRuleSimilarity}(r_{1i}, \Phi D_1)$
   17.     foreach rule $r_{2j} \in P_2$
   18.     if $r_{2j} \in PR_2$ then
   19.         $rs_{2j} \leftarrow \text{ComputeRuleSimilarity}(r_{2j}, \Phi P_2)$
   20.     elsif $r_{1i} \in DR_1$ then
   21.         $rs_{2j} \leftarrow \text{ComputeRuleSimilarity}(r_{2j}, \Phi D_2)$
   22.     $S_{rule-set} \leftarrow \text{average of } rs \text{ of permit rules}$
   23.     $S_{rule-set} \leftarrow \text{average of } rs \text{ of deny rules}$

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

end PolicySimilarityMeasure.

---

3.8 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 9 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 $\Phi$ mappings (line 8-11). Fourth, we use the $\Phi$ 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 corre-
Procedure ComputeRuleSimilarity($r'$, $\Phi$)

**Input**: $r'$ is a rule and $\Phi$ is a mapping between rules

1. **foreach** rule $r'' \in \Phi$
2. $\text{sum} = \text{sum} + S_{\text{rule}}(r', r'')$
3. $\text{rs} = \frac{\text{sum}}{|\Phi|}$
4. **return** $\text{rs}$

end ComputeRuleSimilarity.

Fig. 10. Procedure for Computing Rule Similarity

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 $O(n_a \log n_a)$ 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. $\leq 10$). 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 $O(n_a \log n_a + n_a c)$. For each rule, there are at most 5 elements, and the computation complexity of $S_{\text{rule}}$ is still $O(n_a \log n_a)$.

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 [Fisler et al. 2005] is 50. The maximum number of attribute-value pairs in one policy we have seen so far is about 500 [Schaad et al. 2001]. Considering that the average number of attributes in one policy component is even smaller, our similarity score computation is very efficient.

4. PROPERTIES OF THE POLICY SIMILARITY MEASURE

This section presents the properties of the similarity measure followed by the case study.

4.1 Properties

In this discussion, we focus on policies with simple conditions. In particular, the condition in each rule is in the form of $(c_1 \text{ OR } c_2 \text{ OR } ... \text{ OR } c_k)$, where $c_i$ $(1 \leq i \leq k)$ is either a simple atomic Boolean expression or atomic Boolean expressions connected by AND. A simple atomic Boolean expression is defined as $x \oplus v$, where $x$ is an attribute, $v$ is a constant, and $\oplus$ can be $=$, $\neq$, $>$, $<$, $\geq$, or $\leq$. Here we do not consider complex Boolean expressions such as linear functions due to the following two reasons. First, comparing policies with complex conditions and providing accurate results is a very difficult topic even for fine-grained policy analyzers as discussed in the related work section. Second, most policies use only simple conditions \[.\] In what follows, we present the properties of the similarity measure for the policies with simple conditions.

As a preparation, we first define the notion of equivalent policies.

**Definition 1.** Let $P_1$ and $P_2$ be two policies. We say $P_1$ and $P_2$ are equivalent policies if for all requests applicable to either $P_1$ or $P_2$, the two policies will always yield the same decision (permit or deny) at the PDP (policy decision point).

Here we emphasize the applicability of the requests because that the decisions for ...
non-applicable requests rely on the type of the PEP (policy enforcement point). More specifically, non-applicable requests may obtain permit or deny decisions if permit-based PEP or deny-based PEP is employed respectively. The following example illustrates the issue. Policy $P_4$ uses a single permit rule to specify that Bob can access the server during 8am to 10am. Policy $P_5$ uses a single deny rule to specify that Bob cannot access the server before 8am or after 10am. These two policies are seemingly the same but essentially not. Consider a request sent by Bob at 9am. The request will be permitted by $P_4$ and is not applicable regarding $P_5$. If a deny-based PEP is used, $P_5$ will yield deny for the request. Therefore, $P_4$ and $P_5$ are not equivalent as they cannot always generate the same decision for the same request. This also indicates that rules with opposite conditions and effects are not equivalent, and conforms the philosophy of our similarity measure that only compares rules with the same effect.

We now proceed to introduce the first property.

**Property 1.** The similarity score $S_{policy}$ of two equivalent policies is no less than 0.83 when equal weights are used.

**Proof.** Let $P_1$ and $P_2$ be the two equivalent policies.

(i) When $P_1$ and $P_2$ are actually identical, the similarity score is obviously 1 according to our algorithm.

(ii) When $P_1$ and $P_2$ are different only due to the use of words, the dictionary lookup technique will handle it and the similarity score is also 1.

(iii) When $P_1$ and $P_2$ are semantically equivalent but syntactically different, we show that the obtained similarity score will not be less than 0.83. Since two policies are equivalent, they should have the same set of applicable requests, which means the policy targets and rule targets in both policies must match after the policy normalization process. Therefore, $S_T(P_1, P_2) = 1$ and $S_t(r_i, r_j) = 1$, where $r_i$ and $r_j$ are matching rules in $P_1$ and $P_2$ respectively.

For any request applicable to both policies, it will obtain the same decision from at least one of the rules in each policy respectively. This implies that the corresponding rules in two policies not only have matching targets but also contain the same set of attributes in their condition component though the entire expression can be different. Recall that we only consider conditions consisting of atomic Boolean expressions. Thus, each predicate in one rule’s condition will be able to find a predicate on the same attribute in the matching rule’s condition, i.e. $|M_\theta| = max(N_{a_1}, N_{a_2}$ in Equation 9. Next, from Equation 11 and 14, we obtain the score between two matching attributes $S_{\langle\text{att}\_\text{ype}\rangle}(a_{1k}, a_{2l})$ which is at least 0.5(1+0) = 0.5. Therefore, $S_{\langle\text{condition}\rangle}$ will also be at least 0.5 according to Equation 9.

Combining $S_t(r_i, r_j)$ and $S_{\langle\text{condition}\rangle}$, we obtain the lower bound of the score between two matching rules $S_{\text{rule}}$, which is 0.5(1+0.5) = 0.75. This also suggests that the threshold of the $\Phi$ mapping should be set close to 0.75. Consequently, from Equation 1 and 2, we know that $r_{s1}$ and $r_{s2}$ will also not be less than 0.75. Thus, the lower bound of $S_{\text{rule-}\_\text{set}}^P$ and $S_{\text{rule-}\_\text{set}}^D$ is also 0.75. Finally, we obtain the lower bound of $S_{\text{policy}}$ which is $1/3(1 + 0.75 + 0.75) = 0.83$ according to Equation 5. \qed
It is worth noting that there could be false positives by using the similarity measure. For example, policy $P_6$ has two rules $r_{61}$ and $r_{62}$ with permit and deny effect respectively. $r_{61}$ and $r_{62}$ apply to the same set of requests. Policy $P_7$ has the same two rules as $P_6$. The only difference is that $P_6$ employs the permit-override combining algorithm while $P_7$ employs the deny-override combining algorithm. These two policies will also generate opposite decisions for the same request though they will obtain the similarity score 1. However, from another point of view, these two policies are very similar and only a minor change on the policy combining algorithm can make them compatible. It would be helpful to report such policies to users so that they can make the final decision on whether they would like to negotiate with the potential collaborators.

**Property 2.** When the similarity score $S_{policy}$ is 0, two policies are totally different.

Property 2 can be easily verified because score 0 will be obtained only when two policies have no attributes in common. In other words, such policies do not apply to any same request.

**Property 3.** When $0 < S_{policy} < 1$, $S_{policy}$ indicates the average percentage of requests with the same decisions for policies with the same set of applicable requests.

Without any prior knowledge, the similarity measure is developed based on the assumption of a uniform distribution on requests. In other words, each rule in the policy is considered equally important. During each step of the calculation, an average score is obtained. The usage of Φ mapping helps pruning rules with similar targets but different conditions (e.g. conditions applicable to different sets of requests). For policies with the same set of applicable requests, the final similarity score is an approximation of the average percentage of requests with the same decisions as demonstrated by our experimental results.

In addition, the similarity measure also captures policies with potential to be revised to achieve high similarity. The measure for the categorical value is designed for such purpose.

Finally, the similarity measure can be easily extended to compare two policy sets. In particular, the scores between two policies can be aggregated in the same way as we handle scores between two rules and hence we can obtain a final score for two policy sets.

### 4.2 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 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...
In the following discussion we refer to the policies shown in Figures 2, 3 and 4. We also refer to two attribute hierarchies in the domain, namely the user hierarchy (Figure 11) and file type hierarchy (Figure 12). 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, $PR_1(\text{PR}_2)$ and $DR_1(\text{DR}_2)$. These sets are
   
   $PR_1 = \{R_{11}\}$
   $PR_2 = \{R_{21}, R_{22}\}$
   $DR_1 = \{R_{12}\}$
   $DR_2 = \{R_{23}, R_{24}\}$

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

   $S(R_{11}, R_{21}) = 0.81$
   $S(R_{11}, R_{22}) = 0.56$
   $S(R_{12}, R_{23}) = 0.81$
   $S(R_{12}, R_{24}) = 0.76$

3. For policy $P_1$, we find the $\Phi$ mappings $\Phi^P_1$ and $\Phi^D_1$ using the **ComputePhiMapping** procedure. We use 0.7 as the value of the threshold for this example when
computing the mappings. The $\Phi$ mappings obtained for policy $P_1$ are as follows:

$$
\begin{align*}
\Phi^P_1 &= \{R11 \rightarrow \{R21\}\} \\
\Phi^D_1 &= \{R12 \rightarrow \{R23, R24\}\}
\end{align*}
$$

(4) The $\Phi$ mappings $\Phi^P_2$ and $\Phi^D_2$ are calculated similarly for policy $P_2$.

$$
\begin{align*}
\Phi^P_2 &= \{R21 \rightarrow \{R11\}, R22 \rightarrow \{\}\} \\
\Phi^D_2 &= \{R23 \rightarrow \{R12\}, R24 \rightarrow \{R12\}\}
\end{align*}
$$

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

$$
\begin{align*}
rs_{11} &= S_{\text{rule}}(R11, R21) = 0.81 \\
rs_{12} &= \frac{1}{2} \left[ S_{\text{rule}}(R12, R23) + S_{\text{rule}}(R12, R24) \right] = 0.79
\end{align*}
$$

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

$$
\begin{align*}
rs_{21} &= S_{\text{rule}}(R11, R21) = 0.81 \\
rs_{22} &= 0 \\
rs_{23} &= S_{\text{rule}}(R12, R23) = 0.81 \\
rs_{24} &= S_{\text{rule}}(R12, R24) = 0.76
\end{align*}
$$

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

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

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

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

(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 $P_2$:

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

$S(R_{11}, R_{21}) = 0.7$
$S(R_{12}, R_{23}) = 0.66$

By using the threshold 0.7, we obtain the following $\Phi$ mappings:

$\Phi^p = \{ R_{11} \rightarrow \{ R_{31} \} \}$
$\Phi^D = \{ R_{12} \rightarrow \{ \} \}$

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

$S_{policy}(P_1, P_3) = \frac{1}{3} S_T + \frac{1}{3} S_{rule-set} + \frac{1}{3} S_{rule-set}$
$= \frac{1}{3} \cdot 0.5 + \frac{1}{3} \cdot 0.7 + \frac{1}{3} \cdot 0$
$= 0.4$

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.

$S(R_{81}, R_{91}) = 0.98$
$S(R_{82}, R_{91}) = 0.98$

$S_{rule-set}^p = \frac{r_{81} + r_{82} + r_{91}}{3}$
$= \frac{S(R_{81}, R_{91}) + S(R_{82}, R_{91}) + S(R_{82}, R_{91})}{3}$
$= \frac{0.98 + 0.98 + 0.98}{3}$
$= 0.98$

$S_{policy}(P_8, P_9) = \frac{1}{3} S_T + \frac{1}{3} S_{rule-set}^p + \frac{1}{3} S_{rule-set}$
$= \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 0.98 + \frac{1}{3} \cdot 1$
$= 0.99$
5. EXPERIMENTS

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 also measured the scalability of our approach.

We have used the Falcon-AO v0.7 [Falcon] 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 [WordNet] English language lexical 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.

5.1 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 [swportal] and swrc_updated [swrc_update] ontologies that are available online. In addition we also introduced semantic variations (synonyms) in the attribute list.

5.2 Results

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.

5.2.1 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 [Lin et al. 2007b]. 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 13(a) shows the policy similarity score and policy difference percentage for policy pairs in set-4 and set-8 with the threshold $\epsilon$ set to 0.5. 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.

We also evaluated the effect of the threshold $\epsilon$ by varying $\epsilon$ from 0.2 to 0.8 for the test set set-8. The result is shown in Figure 13(b). Observe that higher values of $\epsilon$ tend to provide a better approximation. This is because the overall similarity score is the average of the rule similarity scores above $\epsilon$ and using higher values of $\epsilon$ prunes more rules which are less similar to one another.

Similar experiments for policy pairs in set-4 and set-8 conducted with a variant of the policy similarity measure that considers operators in attribute predicates (in Section 3.6) showed only marginal improvement in the effectiveness of the scores obtained and the trends observed were similar to the version that did not consider operators. The possible reason is that the number of policies with similar attributes but totally different operators is not very significant in general and hence the average performance of two approaches is similar.

ACM Journal Name, Vol. V, No. N, Month 20YY.
5.2.2 Efficiency. The previous set of experiments demonstrate the effectiveness of the policy similarity measure. In order for our technique to be useful as a filter technique that can quickly prune dissimilar 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 14. Each point in the graphs corresponds to the average execution time for 10 different policy pairs. The value of $\epsilon$ 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 as a filter phase before invoking the more computationally expensive similarity analysis.

5.2.3 Scalability. In these set of experiments, we evaluated the scalability of the policy similarity measure implementation considering both versions with and without ontology matching\textsuperscript{1}. We used test cases that considered attribute names and values from the WordNet synonym set and ontologies as described in Section 5.1. 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 $\epsilon$ was set to 0.5.

Figure 15 reports the average time taken to compute similarity scores for 10 different policy pairs in data sets containing 4 and 8 rules per policy, when varying

---

\textsuperscript{1}We use the term “ontology mapping” to refer also to dictionary lookup
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 similarity analyzer indicating that incorporating ontology matching as part of a filtering phase can still be feasible.

5.2.4 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 II. 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>
<thead>
<tr>
<th>Policy Variant</th>
<th># of Changes</th>
<th>Score Without Ontology Matching</th>
<th>Score With Ontology Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant1</td>
<td>5</td>
<td>0.58</td>
<td>0.99</td>
</tr>
<tr>
<td>Variant2</td>
<td>8</td>
<td>0.58</td>
<td>0.98</td>
</tr>
<tr>
<td>Variant3</td>
<td>12</td>
<td>0.55</td>
<td>0.99</td>
</tr>
<tr>
<td>Variant4</td>
<td>17</td>
<td>0.43</td>
<td>0.98</td>
</tr>
<tr>
<td>Variant5</td>
<td>23</td>
<td>0.43</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table II. Similarity Scores for the With and Without Ontology Versions
6. RELATED WORK

Several efforts have been devoted to the analysis of access control policies with respect to policy property verification, policy conflict detection and so forth.

Most approaches to policy analysis are based on model checking. Ahmed et al. [Ahmed and Tripathi 2003] propose a methodology for specifying and verifying security constraints in role based CSCW (Computer Supported Cooperative Work) systems by using finite-state based model checking. Guelev et al. present a formal language for expressing access-control policies and queries [Guelev et al. 2004]. Their follow-up work [Zhang et al. 2005] provides a model-checking algorithm which can be used to evaluate access control policies written in their proposed language. The evaluation includes not only assessing whether the policies give legitimate users enough permissions to reach their goals, but also checking whether the policies prevent intruders from reaching their malicious goals.

Concerning the specific problem of policy similarity, this problem has not been much investigated and only a few approaches have been proposed. Koch et al. [Koch et al. 2001] propose a uniform framework for comparing different policy models. Specifically, they use graph transformations to represent policy change and integration. Though they present examples of changes and the result as graphs, they do not give any detailed algorithm. A more practical work is by Fisler et al. [Fisler et al. 2005], who 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) and is able to verify policy properties and analyze differences between versions of policies. In [Backes et al. 2004], Backes et al. propose an algorithm for checking refinement of privacy policies in an enterprise. The concept of policy refinement is similar to policy similarity in some sense because policy refinement checks if one policy is a ‘subset’ of another. Another category of relevant related work is represented by the approaches to the problem of policy conflict detection [Lupu and Sloman 1999; Moffett and Sloman 1993]. A recent work by Agrawal et al. [Agrawal et al. 2005] investigates interactions among policies and proposes a ratification tool by which a new policy is checked before being added to a set of policies. The main idea of such approach is to determine satisfiability of Boolean expressions corresponding to different policies.

More recently, Mazzoleni et al. [Mazzoleni et al. 2006] 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 existing 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. [Ehrig et al. 2005; hoad and Zobel 2003; Metzler et al. 2005]). However, these cannot be directly applied because of the special structures and properties of the XACML policies.

7. APPLICATIONS

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

7.1 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.

7.2 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.

7.3 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

ACM Journal Name, Vol. V, No. N, Month 20YY.
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 [SELinux] 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 [Sediff] 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 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 \( \{ \text{allow, neverallow, auditallow, dontaudit} \} \), (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 aggregate to determine a similarity score for two SELinux policies.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we defined 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. According to the obtained similarity scores, dissimilar policies can be safely pruned so that the number of policies which need to be further examined is largely reduced. Detailed algorithms for computation of similarity scores are presented. We have implemented a prototype of the proposed techniques and demonstrated their effectiveness, efficiency and scalability. We have addressed the problem of name heterogeneity by considering dictionary lookup and ontology matching techniques.

Our goal for future work is to integrate our algorithm with other policy analysis tools in order to develop a comprehensive environment for policy analysis. Another interesting direction is to extend current approach to find similarity between policies other than XACML according to what we have discussed in Section 7.

Acknowledgement

The work reported in this paper has been partially supported by IBM under the OCR project “Privacy and Security Policy Management”. Participants to this project are: Carnegie Mellon University, IBM T.J. Watson Research Center, Purdue University.

REFERENCES


ACM Journal Name, Vol. V, No. N, Month 20YY.


