# Ontology-based Policy Anomaly Management for Autonomic Computing

Hongxin Hu[1], Gail-Joon Ahn[1], and Ketan Kulkarni[2]
[1]Arizona State University, [2]Intel Corporation
{hxhu,gahn}@asu.edu; {ketankulkarni29}@gmail.com

***Abstract*—The advent of emerging computing technologies such as service-oriented architecture and cloud computing has enabled us to perform business services more efficiently and effectively. However, we still suffer from unintended security leakages by unauthorized actions in business services. Moreover, designing and managing different types of policies collaboratively in such a computing environment are critical but often error prone due to the complex nature of policies as well as the lack of effective analysis mechanisms and corresponding tools. In particular, existing mechanisms and tools for policy management adopt different approaches for different types of policies. In this work, we propose a unified framework to facilitate collaborative policy analysis and management for different types of policies, focusing on policy anomaly detection and resolution. Our generic approach captures the common semantics and structure of different types of access control policies with the notion of policy ontology. We also discuss a proof-of-concept implementation of our proposed framework and demonstrate how efficiently our approach can discover and resolve anomalies for different types of policies.**

***Index Terms*—Ontology, policy anomaly analysis, autonomic computing.**

## I. Introduction

We have witnessed explosive growth of the applications adopting service oriented architecture (SOA) and cloud computing on the Internet. SOA technology and Cloud computing brought the concept of multi-tenancy for serving various subscribers through a common pool of resources. In such an environment, it is necessary to have a more flexible and collaborative access control mechanism to prevent unintended access of shared resources and private user data. Therefore, the use of a policy-based approach has received considerable attention to accommodate the security requirements covering such large, open, distributed and heterogeneous computing environments.

A *policy*, the basic building block of policy-based system, is a set of rules that control the behaviors of a system. Policy-based computing handles complex system properties by separating policies from system implementation and enables dynamic adaptability of system behaviors by changing policy configurations without reprogramming the systems. Different types of access control policies have been developed to support policy-based computing, including *application-level policies* (e.g., XACML [25], SAML [24], Ponder [17] and EPAL [11]), *network-level policies* (e.g., firewall policy [15] and IPSec policy [16]), and *system-level policies* (e.g., SELinux policy [23] and AppArmor policy [1]).

Policies in modern systems are exponentially growing in size and complexity. In a typical policy, multiple rules may overlap, which means one access request may match several rules. Furthermore, multiple rules within one policy may conflict, implying that those rules not only overlap each other but also yield different decisions. Conflicts in a policy may lead to both safety problem (e.g. allowing unauthorized access) and availability problem (e.g. denying legitimate access). On the other hand, there might be some rules that are redundant, meaning that an access request matching one rule also matches other rules with the same effect. In such a case, the performance of an access control system might be degraded since it directly depends on the number of rules to be evaluated within policies. Consequently, the increasing complexity of policy-based computing strongly demands automated policy analysis techniques. Without having such analysis techniques in place, most benefits of policy-based techniques may be in vain.

Recently, policy anomaly analysis has received a great deal of attention [9], [10], [19], [18], [20], [26]. Corresponding policy analysis tools have been introduced. For example, Firewall Policy Advisor [9], FIREMAN [26] and FAME [19] were designed with the goal of detecting policy anomalies in firewall. Other tools, such as XAnalyzer [20], were developed for helping policy administrators to discover and resolve policy anomalies in XACML policies. However, most of these prior approaches handle policy analysis and management focusing on a particular type of policy. As a result, policy administrators have to get familiar with each of these tools for analyzing and managing different types of policies in their enterprise systems and may get confused with those different policy analysis methods. Therefore, a unified policy management mechanism is desirable for seamlessly managing different types of access control policies, which is especially critical for collaborative policy analysis in a heterogeneous computing environment.

In this paper, we present a unified anomaly management framework for representing and analyzing different types of access control policies in terms of policy ontology. Our approach employs a policy-based segmentation technique to

COLLABORATECOM 2011, October 15-18, Orlando, United States

DOI 10.4108/icst.collaboratecom.2011.247119

facilitate not only more accurate anomaly detection but also effective anomaly resolution. Furthermore, we implement a proof-of-concept tool based on our proposed framework. To evaluate the practicality of our approach, our experiments deal with a set of firewall and XACML policies.

The rest of this paper is organized as follows. Section II overviews policy anomalies in both firewall and XACML policies. We describe our generic ontology-based anomaly management framework in Section III. In Section IV, we discuss the implementation of our tool and the evaluation of our approach. Section V overviews the related work and we conclude this paper in Section VI.

## II. Background

In this section, we overview policy anomalies in two typical kinds of access control policies, firewall policy and XACML policy, which are used to demonstrate our approach in this paper.

### A. Anomalies in Firewall Policies

Firewalls are a widely deployed security mechanism to ensure the security of private networks in most businesses and institutions. The effectiveness of security protection provided by a firewall mainly depends on the quality of policy configured in the firewall. A firewall policy consists of a sequence of rules that define the actions performed on packets that satisfy certain conditions. The rules are specified in the form of $\langle condition, action \rangle$. A $condition$ in a rule is composed of a set of fields to identify a certain type of packets matched by this rule. Table I shows an example of a firewall policy, which includes five firewall rules $r_1$, $r_2$, $r_3$, $r_4$ and $r_5$. Note that the symbol "*" utilized in firewall rules denotes a domain range. For instance, a single "*" appearing in the IP address field represents an IP address range from 0.0.0.0 to 255.255.255.255.

We articulate firewall policy anomalies based on following classification:

- **Conflict**: One rule is conflicting with other rules, if a rule overlaps with others but defines a different action. In this case, the packets matched by the overlap of those rules may be permitted by one rule, but denied by others. In Table I, $r_2$ correlates with $r_5$, and all UDP packets coming from any port of 10.1.1.* to the port 53 of 172.32.1.* match the intersection of these rules. Since $r_2$ is a preceding rule of $r_5$, every packet within the intersection of these rules is denied by $r_2$. However, if their positions are swapped, the same packets will be allowed.
- **Redundancy**: A rule is redundant if there is another same or more general rule available that has the same effect. For example, $r_1$ is redundant with respect to $r_2$ in Table I, since all UDP packets coming from any port of 10.1.2.* to the port 53 of 172.32.1.* matched with $r_1$ can match $r_2$ as well with the same action.

```
1<PolicySet PolicySetId="PS1" PolicyCombiningAlgId="First-Applicable">
2   <Target/>
3   <Policy PolicyId="P1" RuleCombiningAlgId="Deny-Overrides">
4     <Target/>
5     <Rule RuleId="r1" Effect="Deny">
6       <Target>
7         <Subjects><Subject>      Designer  </Subject>
8                   <Subject>      Tester    </Subject></Subjects>
9         <Resources><Resource> Codes     </Resource></Resources>
10        <Actions><Action>       Change    </Action></Actions>
11      </Target>
12    </Rule>
13    <Rule RuleId="r2" Effect="Permit">
14      <Target>
15        <Subjects><Subject>      Designer  </Subject>
16                  <Subject>      Developer </Subject></Subjects>
17        <Resources><Resource> Reports   </Resource>
18                    <Resource> Codes     </Resource></Resources>
19        <Actions><Action>       Read      </Action>
20                 <Action>       Change    </Action></Actions>
21      </Target>
22      <Condition>      8:00 ≤ Time ≤ 17:00      </Condition>
23    </Rule>
24    <Rule RuleId="r3" Effect="Deny">
25      <Target>
26        <Subjects><Subject>      Designer  </Subject></Subjects>
27        <Resources><Resource> Reports   </Resource>
28                    <Resource> Codes     </Resource></Resources>
29        <Actions><Action>       Change    </Action></Actions>
30      </Target>
31      <Condition>      12:00 ≤ Time ≤ 13:00      </Condition>
32    </Rule>
33  </Policy>
34  <Policy PolicyId="P2" RuleCombiningAlgId="Permit-Overrides">
35    <Target/>
36    <Rule RuleId="r4" Effect="Deny">
37      <Target>
38        <Subjects><Subject>      Developer</Subject></Subjects>
39        <Resources><Resource> Reports   </Resource></Resources>
40        <Actions><Action>       Change    </Action></Actions>
41      </Target>
42    </Rule>
43    <Rule RuleId="r5" Effect="Permit">
44      <Target>
45        <Subjects><Subject>      Manager   </Subject>
46                  <Subject>      Designer  </Subject></Subjects>
47        <Resources><Resource> Reports   </Resource>
48                    <Resource> Codes     </Resource></Resources>
49        <Actions><Action>       Change    </Action></Actions>
50      </Target>
51    </Rule>
52  </Policy>
53</PolicySet>
```

Fig. 1. An example XACML policy.

### B. Anomalies in XACML Policies

XACML has become the *de facto* standard for describing access control policies and offers a large set of built-in functions, data types, combining algorithms, and standard profiles for defining application-specific features. Figure 1 shows an example XACML policy. The root policy set $PS_1$ contains two policies, $P_1$ and $P_2$, which are combined using *First-Applicable* combining algorithm. The policy $P_1$ has three rules, $r_1$, $r_2$ and $r_3$, and its rule combining algorithm is *Deny-Overrides*. The policy $P_2$ includes two rules $r_4$ and $r_5$ with *Deny-Overrides* combining algorithm. In this example, there are four subjects: *Manager*, *Designer*, *Developer* and *Tester*; two resources: *Reports* and *Codes*; and two actions: *Read* and *Change*. Note that both $r_2$ and $r_3$ define conditions over the *Time* attribute.

An XACML policy may contain both policy components

TABLE I
AN EXAMPLE FIREWALL POLICY.

| Order | Rule | Protocol | Source IP | Source Port | Destination IP | Destination Port | Action |
|---|---|---|---|---|---|---|---|
| 1 | $r_1$ | UDP | 10.1.2.* | * | 172.32.1.* | 53 | deny |
| 2 | $r_2$ | UDP | 10.1.*.* | * | 172.32.1.* | 53 | deny |
| 3 | $r_3$ | TCP | 10.1.*.* | * | 192.168.*.* | 25 | allow |
| 4 | $r_4$ | TCP | 10.1.1.* | * | 192.168.1.* | 25 | deny |
| 5 | $r_5$ | * | 10.1.1.* | * | * | * | allow |

and policy set components. Often, a rule anomaly occurs in a policy component, which consists of a sequence of rules. On the other hand, a policy set component consists of a set of policies or other policy sets, thus anomalies may also arise among policies or policy sets.

- **Anomalies at Policy Level:** A rule is ***conflicting*** with other rules, if this rule overlaps with others but defines a different effect. For example, the *deny* rule $r_1$ is in conflict with the *permit* rule $r_2$ in Figure 1 because rule $r_2$ allows the access requests from a designer to change codes in the time interval [8:00, 17:00], which are supposed to be denied by $r_1$; and a rule is ***redundant*** if there is other same or more general rules available that have the same effect. For instance, if we change the effect of $r_2$ to *Deny*, $r_3$ becomes redundant since $r_2$ will also deny a designer to change reports or codes in the time interval [12:00, 13:00].
- **Anomalies at Policy Set Level:** Anomalies may also occur across policies or policy sets in an XACML policy. For example, considering two policy components $P_1$ and $P_2$ of the policy set $PS_1$ in Figure 1, $P_1$ is ***conflicting*** with $P_2$, because $P_1$ permits the access requests that a developer changes reports in the time interval [8:00, 17:00], but which are denied by $P_2$. On the other hand, $P_1$ denies the requests allowing a designer to change reports or codes in the time interval [12:00, 13:00], which are permitted by $P_2$. Supposing the effect of $r_2$ is changed to *Deny* and the condition of $r_2$ is removed, $r_4$ is turned to be ***redundant*** with respect to $r_2$, even though $r_2$ and $r_4$ are placed in different policies $P_1$ and $P_2$, respectively.

## III. ONTOLOGY-BASED ANOMALY MANAGEMENT FRAMEWORK

Our ontology-based policy anomaly management framework is composed of three core functionalities: *policy ontology extraction*, *policy ontology population* and *policy anomaly analysis*, as depicted in Figure 2. First, the general policy domain concepts, policy structure and semantics are captured from diverse access control policies for the construction of a policy ontology. Then, the generated policy ontology can be employed by different types of access control policies to populate corresponding policy ontology instances, which are in turn utilized by a unified policy analysis mechanism adopting a policy-based segmentation technique to facilitate effective anomaly detection and resolution.

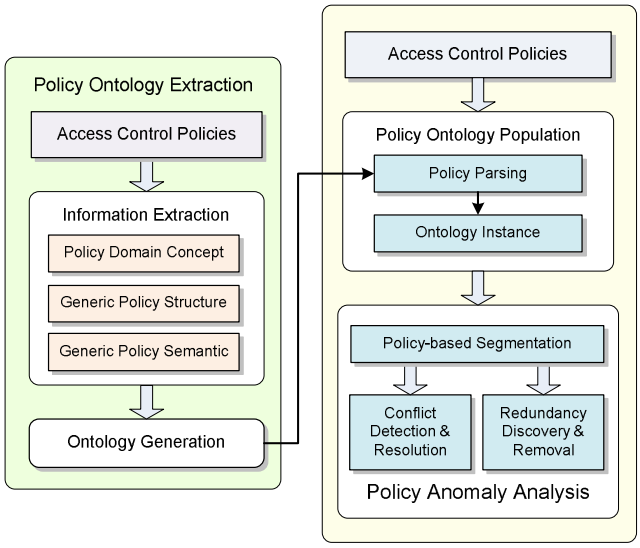



Fig. 2. Ontology-based policy anomaly management framework.

### A. Policy Ontology Extraction

Generic access control policy representation enables the policy analysis and management mechanisms to be independent of different types of access control policies. We examined a variety of existing access control policies, and observed that following characteristics should be extracted to capture a generic policy representation for building our policy ontology.

- *Policy Domain Concepts*: Policy domain concepts can be considered as the terms that are generally used to describe the access control policies.
- *Policy Structure*: Policy structure depicts how policy components are arranged within policies.
- *Policy Semantics*: Policy semantics represent the relationships among policy components and also describe the behaviors of policies.

In order to achieve a uniform policy analysis and management, generic policy representation should successfully capture above characteristics from different types of policies. Then, the generated policy ontology can be reused by different types of policies for policy analysis. However, the policy ontology may need further refinement whenever changes are made to the existing policy specifications or a new type of policy is considered in our generic policy management.

*1) Capturing Policy Domain Concepts:* Capturing access control policy domain concepts can be considered as the first step towards the capturing generic policy representation. We analyzed the specifications of several typical access control policies and enlisted the terms used for specifying those policies. After that, we classified these terms from different types of policies under common classes that can be treated as access control policy domain concepts. For example, we first enlisted terms such as rule, policy, policy set, access control list, subject, action, resource, effect, combining algorithm, conflict resolution strategy and so on from different policies. Then, we classified them under common concepts, such as Policy, Policy Group, PolicyRule. To give an example of our classification process, consider XACML has a notion of *combining algorithm*, which indicates the behavior of a policy in case of conflicts. However, firewall policy utilizes a *First-Match strategy* for conflict resolution. Both of them can be classified into one class *Meta-Policy*.

*2) Capturing Generic Policy Structure:* We examined the structures of different access control policies. For example, XACML policy structure has the notion of *<Rule>*, *<Policy> and <PolicySet>*. PolicySet is the container for other policies as well as policy sets. Policy defines the list of rules. Considering other access control policy such as firewall policy, it has the concept of *Access Control List (ACL)*, which includes a list of firewall rules. A typical firewall policy may contain a number of ACLs. Thus, each ACL in a firewall policy is similar to the *policy* node containing a group of rules, and a firewall policy can be also treated as a group of *policy* nodes.

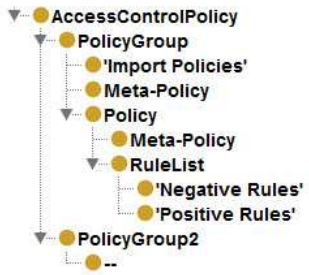



Fig. 3. Generic access control policy structure.

Based on our observation and examination of different types of policy specifications, we construct a generic access control policy structure. An access control policy defines what activities a member of the *subject* domain can perform on a set of objects in the *Resource* domain. The basic node or unit for defining a policy is *Policy Rule*. These rules either permit or deny access to the resource objects and hence can be classified into two types: *Positive Rule* and *Negative Rule*, respectively. Rules that are applicable to the same subject or resource objects can be arranged into a *Policy* node as a *Rule List*. Each *Policy* node may have meta-policies associated with it, which specify the policy behaviors with respect to the rule conflicts. For example, conflict resolution strategy specified for policy node defines the behavior of a policy in case of conflict. Multiple related policies can be grouped together to generate a composite *Policy Group*. For example, policies related to the same department or same application may be grouped together for policy organization and management. A policy group may also have meta-policies associated with it. Access control policy might contain *Policy* as root node or it might have a hierarchical structure where root node is *Policy Group* containing other policies or policy groups. Figure 3 depicts a generic structure of access control policy.

*3) Capturing Generic Policy Semantics:* Access control policies are defined in terms of the policy attributes. Attributes are named values of known items and are characteristics of the *Subject*, *Resource* and *Action*, in which the access requests are made. Rules in an access control policy are defined based on these attributes. A general semantic of a rule in an access control policy can be described as *which subject(s) has access to which resource(s) and with what action(s) (permit or deny)?* Thus, a high-level semantic of access control policies is typically based on the policy rule expressed in terms of attributes of *Subject*, *Resource*, *Action* and *Effect*. Moreover, rules may have conditions that need to be satisfied for making access decisions.

To capture the high-level semantics mentioned above in generic policy representation, we need to identify attributes of items, *Subject*, *Resource*, *Action* and *Effect* from different types of policies. In addition, we need to identify more attributes considering other items, such as *condition* and *conflict resolution strategy*, for comprehensive policy representation.

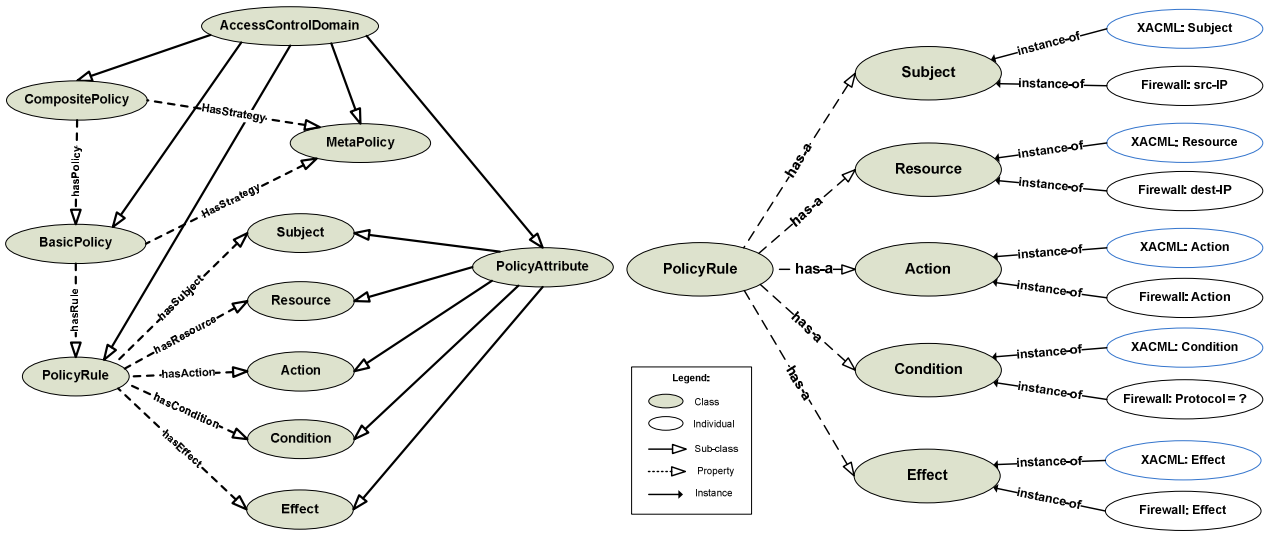
For example, considering a rule $r_2$ from the example XACML policy in Figure 1, we can easily extract following attributes based on identified items:

**Subject** – *Designer, Developer*
**Resource** – *Reports, Codes*
**Action** – *Read, Change*
**Condition** – *Time between 8:00 AM to 5:00 PM*
**Effect** – *Permit*

Then, regarding a rule $r_4$ in the example firewall policy in Table I, we can identify rule attributes as follows:

**Subject** – *10.1.1.*: **
**Resource** – *192.186.1.*: 25*
**Action** – *Access (default action)*
**Condition** – *(Protocol == UDP)*
**Effect** – *Deny*

In addition to the above rule level semantics, a policy may also specify attributes such as *conflict resolution strategy* which defines the behavior of a policy in case of conflicts. For example, XACML defines four different *combining algorithms*: *Deny-Overrides*, *Permit-Overrides*, *First-Applicable* and *Only-One-Applicable*, while firewall policy uses a default *First-Match* strategy. Those specific attributes should be additionally identified and associated with proper policy structure components such as *Policy* or




(a) Access control policy ontology. (b) Ontology instance representation.

Fig. 4. Policy ontology and instance.

*Policy Group* node for generic policy representation.

*4) Policy Ontology Generation:* A generic representation of policy requires to identify the domain concepts, policy structure and semantics shared by different types of policies. We discussed the process of capturing these characteristics previously. Then, a uniform format or template is needed to store the generic information about policy domain concepts, structure and semantics effectively and accurately, so that the identified information is reusable for different kinds of policies based on a generic policy analysis mechanism. To this end, we adopt a method of modeling the policy domain using *ontology*

*Ontology* is a formal representation of knowledge as a set of concepts within a particular domain and relationships between those concepts. It provides us the shared vocabulary for modeling a domain, which includes the types of concepts in the domain, their properties and relationships. Creating access control policy ontology enables us to model the access control policy domain by defining its vocabulary, objects and their relations along with the properties. Policy ontology which represents shared domain knowledge provides us a template which can be instantiated with the information extracted from different types of access control policies to enable a generic policy representation for policy analysis.

Figure 4(a) shows a generic ontology created for the access control policy domain. We utilize the policy domain concepts, structure and semantics capetured from a variety of policy specifications to generate this policy ontology. Web Ontology Language (OWL) [4], which represents the family of knowledge representation languages for authoring an ontology, is adopted to represent our policy ontology. To create the policy ontology using OWL, we use protege tool [5].

OWL allows us to describe the domain in terms of `classes`, `properties` and `individuals`. `Classes` represent a collection of objects, usually sharing some common properties. We created the base classes for domain concepts such as *subject* and *action*. `Properties` represent the relationships between individual concepts. There are two types of OWL properties used to define the relationships:

1) *Object Properties*: An object property is a relationship between two individuals or concepts of ontology domain. We defined different object properties to capture the generic structure as well as semantic information of access control policies. For example, an object property *hasRule* between concepts *AccessControlPolicy* and *PolicyRule* captures the structure information that a policy node may contain one or more rules.
2) *Data Properties*: A data property links an individual concept to its literal value. For example, we use a data property *hasValue* to assign a literal value to a concept such as *Subject*.

Defining policy ontology enables us to capture the general information of policy domain concepts, policy structure and policy semantics. Policy ontology can be easily extended to support new characteristics introduced due to the changes in the access control policy specifications. We can easily add new data properties or object properties along with the new domain concepts in our policy ontology to reflect these changes.

### B. Policy Ontology Population

To create a generic representation of a particular type of access control policy, we instantiate the policy ontology with the structure and semantic information extracted from the particular policy. This method of instantiating the basic ontology with the specific attributes, properties

and relationships extracted from a specific domain, is called *Ontology Population*. Figure 4(b) shows an ontology instance representation with respect to XACML and firewall policies. A node is either labeled with an ontology concept or an information instance obtained from the particular policies. A link labeled *instance Of* from an information instance to an ontology concept represents an instance generated for the corresponding policy ontology concept.

We use policy parsers to extract the information required for ontology population. A parser understands the semantic and structure information for a particular policy and abstract the information, such as *Subject*, *Resource*, *Action* and *PolicyRule*, required by policy ontology. For example, a parser for the firewall policy parses out `source-ip` information in a rule as the *Subject* for that particular rule in terms of the firewall policy semantics.

### C. Policy-based Segmentation for Anomaly Detection

*1) BDD-based Policy Representation:* Our policy-based segmentation technique introduced in subsequent sections requires a well-formed representation of policies for performing a variety of set operations. Binary Decision Diagram (BDD) [14] is a data structure that has been widely used for formal verification and simplification of digital circuits. In this work, we leverage BDD as the underlying data structure to represent policy ontology instances and facilitate uniform policy analysis.

Given the ontology instance corresponding to a particular policy, we can use `OWLOntologyWalker` and `OWLOntologyWalkerVisitor` provided by OWL API to walk the asserted structure of the policy ontology to obtain the information about required attributes such as *Subject*, *Resource*, *Action* and *Effect* for each policy rule object in an ontology instance. Once these attributes are identified, all policy rule instances can be transformed into Boolean expressions. Each Boolean expression of a rule is composed of atomic Boolean expressions combined by logical operators $\vee$ and $\wedge$. Atomic Boolean expressions are treated as equality constraints or range constraints on attributes.

Considering the attribute information extracted form an ontology instance of the example XACML policy in Figure 1 in terms of *atomic Boolean expressions*, the Boolean expression for rule $r_1$ is:

$$(Subject = \text{“}Designer\text{”} \vee Subject = \text{“}Tester\text{”}) \wedge (Resource = \text{“}Codes\text{”}) \wedge (Action = \text{“}Change\text{”})$$

Similarly, based on the ontology instance of the firewall policy in Table I, the rule $r_4$ can be represented in Boolean expression as follows:

$$(Subject = \text{“10.1.1.} * : * \text{”}) \wedge (Resource = \text{“192.186.1.} * : 25\text{”}) \wedge (Protocol = \text{“}UDP\text{”})$$

We encode each of the atomic Boolean expression as a Boolean variable. A list of Boolean encoding for the example rules is shown in Table II. We then utilize the Boolean encoding to construct Boolean expressions in terms of Boolean variables for rules.

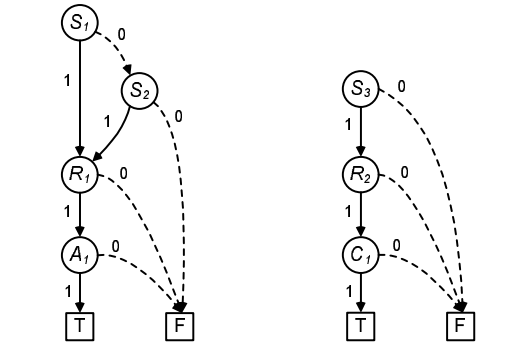
TABLE II
ATOMIC BOOLEAN EXPRESSIONS AND CORRESPONDING BOOLEAN VARIABLES FOR EXAMPLE RULES.

| Unique Atomic Boolean Expression | Boolean Variable |
|---|---|
| $Subject =$ “$Designer$” | $S_1$ |
| $Subject =$ “$Tester$” | $S_2$ |
| $Subject =$ “10.1.1.$*:*$” | $S_3$ |
| $Resource =$ “$Codes$” | $R_1$ |
| $Resource =$ “192.186.1.$*:25$” | $R_2$ |
| $Action =$ “$Change$” | $A_1$ |
| $Protocol =$ “$UDP$” | $C_1$ |

The Boolean expression for XACML rule $r_1$ is:
$(S1 \vee S2) \wedge R1 \wedge A1$

The Boolean expression for firewall rule $r_4$ is:
$S3 \wedge R2 \wedge C1$

BDDs are acyclic directed graphs which represent Boolean expressions compactly. Each nonterminal node in a BDD represents a Boolean variable, and has two edges with binary labels, 0 and 1 for *nonexistent* and *existent*, respectively. Terminal nodes represent Boolean value `T` (True) or `F` (False). Figures 5(a) and 5(b) give BDD representations of above two rules, respectively.



(a) *BDD for an XACML rule* (b) *BDD for a firewall rule*

Fig. 5. Unified BDD-based policy representation.

Once the BDDs are constructed for policy ontology instances, performing set operations, such as unions ($\cup$), intersections ($\cap$) and set differences ($\setminus$), required by our policy-based segmentation is efficient as well as straightforward.

*2) Policy-based Segmentation for Anomaly Detection:* In order to precisely identify policy anomalies and enable an effective anomaly resolution, we adopts a *policy-based segmentation technique* introduced in [19], [20], which utilizes the above-mentioned BDD-based data structure to represent policies and perform various set operations, and to convert a policy into a set of disjoint authorization spaces.

By adopting the policy-based segmentation technique, an entire packet space can be divided into a set of pairwise disjoint segments. We classify the policy segments as follows: *non-overlapping* segment and *overlapping* segment, which is further divided into *conflicting overlapping* segment and *non-conflicting overlapping* segment. Each *non-*

TABLE III
ANOMALY ANALYSIS EVALUATION.

| Policy | Partitions (#) | Conflict Analysis | | | Redundancy Analysis | | |
|---|---|---|---|---|---|---|---|
| | | Policy Level(#) | Group Level(#) | Time (s) | Policy Level(#) | Group Level(#) | Time (s) |
| **XACML Policies** | | | | | | | |
| 1 (CodeA) | 6 | 1 | 1 | 0.095 | 1 | 0 | 0.096 |
| 2 (SamplePolicy) | 8 | 0 | 2 | 0.106 | 0 | 2 | 0.109 |
| 3 (GradeSheet) | 18 | 0 | 4 | 0.125 | 0 | 2 | 0.132 |
| 4 (SynPolicy-1) | 205 | 8 | 14 | 0.364 | 7 | 4 | 0.359 |
| 5 (Continue) | 439 | 9 | 14 | 0.621 | 10 | 7 | 0.597 |
| 6 (SynPolicy-2) | 523 | 29 | 15 | 0.914 | 14 | 8 | 0.903 |
| **Firewall Policies** | | | | | | | |
| 1 (A) | 15 | 3 | 0 | 0.119 | 2 | 0 | 0.113 |
| 2 (B) | 36 | 5 | 0 | 0.153 | 3 | 0 | 0.151 |
| 3 (C) | 89 | 11 | 0 | 0.196 | 5 | 0 | 0.189 |
| 4 (D) | 127 | 18 | 0 | 0.224 | 6 | 0 | 0.213 |
| 5 (E) | 183 | 23 | 5 | 0.417 | 13 | 3 | 0.431 |
| 6 (F) | 405 | 41 | 11 | 0.589 | 16 | 7 | 0.603 |

*overlapping* segment associates with one unique rule and each *overlapping* segment is related to a set of rules, which may conflict with each other (*conflicting overlapping* segment) or have the same action (*non-conflicting overlapping* segment) indicating possible redundancies.

### D. Policy Anomaly Resolution

An intuitive means for resolving policy conflicts by a policy designer is to remove all conflicts by modifying the policies. However, resolving conflicts through changing the policies is remarkably difficult, even impossible, in practice. In [20], we introduced a flexible and extensible conflict resolution framework to achieve a fine-grained policy conflict resolution. In addition, we proposed a property assignment mechanism, which performs three property assignment processes, to effectively identify redundant rules in an examined policy. We adopt those anomaly resolution approaches in our ontology-based policy anomaly management.

## IV. IMPLEMENTATION AND EVALUATION

We have implemented a policy analysis tool called `Generic Policy Analyzer` in Java based on our ontology-based policy anomaly management framework. We used *protege* [5] to extract and define the policy ontology. To create ontology instances for specific policies, we utilized Java-based OWL API [7]. To support policy ontology population, our current implementation supports the parsers for both XACML and firewall policies based on *Sun* XACML implementation [6] and FIREMAN [26] implementation, respectively. JavaBDD [3], which is based on BuDDy package [2], is employed by our tool to support BDD representation and authorization space operations.

We evaluated the efficiency of our tool for policy analysis on both XACML and firewall policies. We performed our experiments on Intel Core 2 Duo CPU 3.00 GHz with 3.25 GB RAM running on Windows XP SP2. Table III summarizes the policies used for our evaluation. Real-life XACML policies utilized for evaluation were collected from different sources. Two of the policies, *CodeA* and *Continue* are XACML policies used in [18]; among them, *Continue* is designed for a real-world Web application supporting a conference management. *GradeSheet* is utilized in [12]. It is difficult to get a large volume of real-world policies because they are often considered to be highly confidential. Thus, we generated two large synthetic policies *SynPolicy-1* and *SynPolicy-2* for further evaluating the performance and scalability of our tool. We also used *SamplePolicy*, which is the example XACML policy represented in Figure 1, in our experiments. Similarly, firewall polices used for evaluation were obtained from our campus networks and synthetical generation.

Time required by our tool for policy anomaly analysis highly depends upon the number of segments generated for each policy. The increase of the number of segments is proportional to the number of components contained in an policy. From Table III, we observe that our tool performs fast enough to handle larger size of policies, such as firewall policies *E* and *F*, even for some complex policies with multiple levels of hierarchies along with hundreds of rules, such as the real-life XACML policy, *Continue*, and the synthetic XACML policy, *SynPolicy-2*. The time trends observed from Table III clearly provide the evidence of efficiency of our tool.

## V. RELATED WORK

Many research efforts have been devoted to policy analysis. However, most existing research work only focus on developing techniques for one specific policy. None of them could design a uniform analysis mechanism to accommodate policy analysis requirements for different types of policies. We only overview some work closely related to this paper.

In [13], the authors formalized XACML policies using a process algebra known as Communicating Sequential Processes (CSP). This work utilizes a model checker to formally verify properties of policies, and to compare access control policies with each other. Fisler et al. [18] introduced

an approach to represent XACML policies with Multi-Terminal Binary Decision Diagrams (MTBDDs). A policy analysis tool called Margrave was developed. Margrave can verify XACML policies against the given properties and perform change-impact analysis based on the semantic differences between the MTBDDs representing the policies. Ahn et al. [8] presented a formalization of XACML using answer set programming (ASP), which is a recent form of declarative programming, and leveraged existing ASP reasoners to conduct policy verification. Hu et al. [20] designed an XACML policy analysis tool called XAnalyzer, which ensures an accurate anomaly detection at both policy level and policy set level, and a fine-grained conflict resolution.

Several work also presented policy analysis tools with the goal of detecting policy anomalies in firewall. Al-Shaer et al. [9] designed a tool called Firewall Policy Advisor which can only detect *pairwise* anomalies in firewall rules. Yuan et al. [26] presented a toolkit, FIREMAN, which can detect anomalies among *multiple* firewall rules by analyzing the relationships between *one* rule and the collections of packet spaces derived from all preceding rules. However, the anomaly detection procedures of FIREMAN are still incomplete [10]. Hu et al. [19] developed a tool, FAME, which could conduct a complete examination of policy anomaly and provide more accurate anomaly diagnosis information for firewall policy analysis.

## VI. Conclusion

We have designed an innovative framework for managing policy anomalies for different types of access control policies. This framework presents a unified policy anomaly analysis approach in terms of policy ontology. We have also described a proof-of-concept implementation of our method and demonstrated how our approach can efficiently discover and resolve policy anomalies. As part of future work, we would like to leverage existing automatic ontology extraction tools for more accurate policy ontology generation. In addition, we would further evaluate our approach with other types of access control policies. Also, we would explore how our ontology-based policy analysis approach can be extended to handle more complicated scenarios in emerging computing environments, such as multiparty access control in online social networks [21], [22].

## Acknowledgments

This work was partially supported by the grants from National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360) and Department of Energy (DE-SC0004308).

## References

[1] AppArmor. http://de.opensuse.org/AppArmor.
[2] Buddy version 2.4. http://sourceforge.net/projects/buddy.
[3] Java BDD. http://javabdd.sourceforge.net.
[4] OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/.
[5] Protege Ontology Editor. http://protege.stanford.edu/.
[6] Sun XACML Implementation. http://sunxacml.sourceforge.net.
[7] The OWL API. http://owlapi.sourceforge.net/.
[8] G. Ahn, H. Hu, J. Lee, and Y. Meng. Representing and reasoning about web access control policies. In *2010 34th Annual IEEE Computer Software and Applications Conference*, pages 137–146. IEEE, 2010.
[9] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM*, volume 4, pages 2605–2616. Citeseer, 2004.
[10] J. Alfaro, N. Boulahia-Cuppens, and F. Cuppens. Complete analysis of configuration rules to guarantee reliable network security policies. *International Journal of Information Security*, 7(2):103–122, 2008.
[11] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (epal). http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/.
[12] A. Birgisson, M. Dhawan, U. Erlingsson, V. Ganapathy, and L. Iftode. Enforcing authorization policies using transactional memory introspection. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 223–234. ACM New York, NY, USA, 2008.
[13] J. Bryans. Reasoning about XACML policies using CSP. In *Proceedings of the 2005 workshop on Secure web services*, page 35. ACM, 2005.
[14] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on computers*, 100(35):677–691, 1986.
[15] D. Chapman, E. Zwicky, and D. Russell. *Building internet firewalls*. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 1995.
[16] M. Condell, C. Lynn, and J. Zao. Security policy specification language. *Internet Engineering Task Force (IETF) Internet Draft*, 2000.
[17] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Policies for Distributed Systems and Networks*, pages 18–38, 2001.
[18] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 196–205, New York, NY, USA, 2005. ACM.
[19] H. Hu, G. Ahn, and K. Kulkarni. Fame: a firewall anomaly management environment. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, pages 17–26. ACM, 2010.
[20] H. Hu, G. Ahn, and K. Kulkarni. Anomaly discovery and resolution in web access control policies. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 165–174. ACM, 2011.
[21] H. Hu and G.-J. Ahn. Multiparty authorization framework for data sharing in online social networks. In *Proceedings of the 25th annual IFIP WG 11.3 conference on Data and applications security and privacy*, DBSec'11, pages 29–43. Springer-Verlag, 2011.
[22] H. Hu, G.-J. Ahn, and J. Jorgensen. Detecting and resolving privacy conflicts for collaborative data sharing in online social networks. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC'11. ACM, 2011.
[23] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. In *Proc. 2001 USENIX Annual Technical Conference REENIX Track*, pages 29–40, 2001.
[24] OASIS. Security Assertion Markup Language. http://www.oasis-open.org/committees/security/.
[25] XACML. OASIS eXtensible Access Control Markup Language (XACML) V2.0 Specification Set. http://www.oasis-open.org/committees/xacml/, 2007.
[26] L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, P. Mohapatra, and C. Davis. Fireman: A toolkit for firewall modeling and analysis. In *2006 IEEE Symposium on Security and Privacy*, page 15, 2006.