Ijesm
Consulting, help, relaxation

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES
&
MANAGEMENT

# systematic detection and resolution of firewall policy anomalies

SHAIK NAZEER BASHA ,PGScholar, QCET,Nellore sknazeerbashamca@gamail.com
SK.Karimulla ,Asst Proof,QCET,Nellore karim.shaik777@gmail.com
P.Babu Associate Professor ,QCET,Nellore,babu123mca@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. Firewalls are the most 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. Unfortunately, designing and managing firewall policies are often error prone due to the complex nature of firewall configurations as well as the lack of systematic analysis mechanisms and tools. In this paper, we represent an innovative policy anomaly management framework for firewalls, adopting a rule-based segmentation technique to identify policy anomalies and derive effective anomaly resolutions. In particular, we articulate a grid-based representation technique, providing an intuitive cognitive sense about policy anomaly. We also discuss a proof-of-concept implementation of a visualization-based firewall policy analysis tool called Firewall Anomaly Management Environment (FAME). In addition, we demonstrate how efficiently our approach can discover and resolve anomalies in firewall policies through rigorous experiments.

**Index Terms**—Firewall, policy anomaly management, access control, visualization tool.

## 1 INTRODUCTION

As one of essential elements in network and information system security, firewalls have been widely deployed in defending suspicious traffic and unauthorized access to Internet-based enterprises. Sitting on the border between a private network and the public Internet, a firewall examines all incoming and outgoing packets based on security rules. To implement a security policy in a firewall, system administrators define a set of filtering rules that are derived from the organizational network security requirements.

Firewall policy management is a challenging task due to the complexity and interdependency of policy rules. This is further exacerbated by the continuous evolution of network and system environments. For instance, Al-Shaer and Hamed [1] reported that their firewall policies contain anomalies even though several administrators including *nine* experts maintained those policies. In addition, Wool [2] recently inspected firewall policies collected from different organizations and indicated that all examined firewall policies have security flaws.

The process of configuring a firewall is tedious and error prone. Therefore, effective mechanisms and tools for policy management are crucial to the success of firewalls. Recently, policy anomaly detection has received a great deal of attention [1], [3], [4], [5]. Corresponding policy analysis tools, such as Firewall Policy Advisor [1] and FIREMAN [5], with the goal of detecting policy anomalies have been introduced. Firewall Policy Advisor only has the capability of detecting *pairwise* anomalies in firewall rules. FIREMAN can detect anomalies among *multiple* rules by analyzing the relationships between *one* rule and the collections of packet spaces derived from all preceding rules. However, FIREMAN also has limitations in detecting anomalies [3]. For each firewall rule, FIREMAN only examines all preceding rules but ignores all subsequent rules when performing anomaly analysis. In addition, each analysis result from FIREMAN can only show that there is a misconfiguration between *one* rule and its preceding rules, but cannot accurately indicate all rules involved in an anomaly.

On the other hand, due to the complex nature of policy anomalies, system administrators are often faced with a more challenging problem in resolving anomalies, in particular, resolving policy conflicts. An intuitive means for a system administrator to resolve policy conflicts is to remove all conflicts by modifying the conflicting rules. However, changing the conflicting rules is significantly difficult, even impossible, in practice from many aspects. First, the number of conflicts in a firewall is potentially large, since a firewall policy may consist of thousands of rules, which are often logically entangled with each other. Second, policy conflicts are often very complicated. One rule may conflict with multiple other rules, and one conflict may be associated with several rules. Besides, firewall policies deployed on a network are often maintained by more than one administrator, and an enterprise firewall may contain legacy rules that are designed by different administrators. Thus, without a priori knowledge on the administrators' intentions, changing rules will affect the rules' semantics and may not resolve conflicts correctly. Furthermore, in some cases, a system administrator may intentionally introduce certain overlaps in firewall rules knowing that only the first rule is important. In reality, this is a commonly used technique to exclude specific parts from a certain action, and the proper use of this technique could result in a fewer number of compact rules [5]. In this case, conflicts are not an error, but intended, which would not be necessary to be changed.

Since the policy conflicts in firewalls always exist and are hard to be eliminated, a practical resolution method is to identify which rule involved in a conflict situation should take precedence when multiple conflicting rules (with different actions) can filter a particular network packet simultaneously. To resolve policy conflicts, a firewall typically implements a *first-match* resolution mechanism based on the order of rules. In this way, each packet processed by the firewall is mapped to the decision of the first rule that the packet matches. However, applying the *first-match* strategy to cope with policy conflicts has limitations. When a conflict occurs in a firewall, the existing first matching rule may not be a desired rule that should take precedence with respect to conflict resolution. In particular, the existing first matching rule may perform opposite action to the rule which should be considered to take precedence. This situation can cause severe network breaches such as permitting harmful packets to sneak into a private network, or dropping legal traffic which in turn could encumber the availability and utility of network services. Obviously, it is necessary to seek a way to bridge a gap between conflict detection and conflict resolution with the first-match mechanism in firewalls.

In this paper, we represent a novel anomaly management framework for firewalls based on a rule-based segmentation technique to facilitate not only more accurate anomaly detection but also effective anomaly resolution. Based on this technique, a *network packet space* defined by a firewall policy can be divided into a set of disjoint packet space segments. Each segment associated with a unique set of firewall rules accurately indicates an overlap relation (either conflicting or redundant) among those rules. We also introduce a flexible conflict resolution method to enable a fine-grained conflict resolution with the help of several effective resolution strategies with respect to the risk assessment of protected networks and the intention of policy definition. Besides, a more effective redundancy elimination mechanism is provided in our framework, and our experimental results show that our redundancy discovery mechanism can achieve approximately 70 percent improvement compared to traditional redundancy detection approaches [1], [6]. Moreover, the outputs of prior policy analysis tools [1], [5] are mainly a list of possible anomalies, which does not give system administrators a clear view of the origination of policy anomalies. Since information visualization technique [7] enables users to explore, analyze, reason, and explain abstract information by taking advantage of their visual cognition, our policy analysis tool adopts an information visualization technique to facilitate policy analysis. A grid-based visualization approach is introduced to represent policy anomaly diagnosis information in an intuitive way, enabling an efficient anomaly management.[1] In addition, we implement a visualization-based firewall anomaly management environment (FAME) based on our approach. To evaluate the practicality of our tool, our extensive experiments deal with a set of real-life firewall policies.

1. Our proposed methodology can be also extended to deal with the anomalies in other kinds of policies, such as XACML-based policies [8].

TABLE 1
An Example Firewall Policy

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

This paper is organized as follows: Section 2 overviews the anomalies in firewall policies. Section 3 presents an anomaly representation technique based on packet space. In Section 4, we articulate our policy anomaly management framework. In Section 5, we address the implementation details and evaluations of FAME. Section 6 describes several important issues followed by the related work in Section 7. Section 8 concludes this paper.

## 2 Overview of Anomalies in Firewall Policies

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 1 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.

Several related work has categorized different types of firewall policy anomalies [1], [5]. Based on following classification, we articulate the typical firewall policy anomalies:

1. *Shadowing*. A rule can be shadowed by one or a set of preceding rules that match all the packets which also match the shadowed rule, while they perform a different action. In this case, all the packets that one rule intends to deny (accept) can be accepted (denied) by previous rule(s); thus, the shadowed rule will never be taken effect. In Table 1, $r_4$ is shadowed by $r_3$ because $r_3$ allows every TCP packet coming from any port of 10.1.1.$^*$ to the port 25 of 192.168.1.$^*$, which is supposed to be denied by $r_4$.
2. *Generalization*. A rule is a generalization of one or a set of previous rules if a subset of the packets matched by this rule is also matched by the preceding rule(s) but taking a different action. For example, $r_5$ is a generalization of $r_4$ in Table 1. These two rules indicate that all the packets from 10.1.1.$^*$ are allowed, except TCP packets from 10.1.1.$^*$ to the port 25 of 192.168.1.$^*$. Note that, as we discussed earlier, generalization might not be an error.
3. *Correlation*. One rule is correlated with other rules, if a rule intersects with others but defines a different

action. In this case, the packets matched by the intersection of those rules may be permitted by one rule, but denied by others. In Table 1, $r_2$ 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.

4. *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 1, 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.

Anomaly detection algorithms and corresponding tools were introduced by [1], [5] as well. However, existing conflict classification and detection approaches only treat a policy conflict as an inconsistent relation between *one* rule and other rules. Given a more general definition on policy conflict as shown in Definition 1, we believe that identifying policy conflicts should always consider a firewall policy as a whole piece, and precise indication of the conflicting sections caused by a set of overlapping rules is critical for effectively resolving the conflicts.

**Definition 1 (Policy Conflict).** *A policy conflict pc in a firewall F is associated with a unique set of conflicting firewall rules* $\texttt{cr} = \{r_1, \ldots, r_n\}$, *which can derive a common network packet space. All packets within this space can match exactly the same set of firewall rules, where at least two rules have different actions:* `Allow` *and* `Deny`.

Similarly, we give a general definition for rule redundancy in firewall policies as follows, which serves as a foundation of our redundancy elimination approach.

**Definition 2 (Rule Redundancy).** *A rule r is redundant in a firewall F iff the network packet space derived from the resulting policy F′ after removing r is equivalent to the network space defined by F. That is, F and F′ satisfy following equations:* $S_F^A = S_{F'}^A$ *and* $S_F^D = S_{F'}^D$, *where* $S^A$ *and* $S^D$ *denote* allowed *and* denied *network packet spaces, respectively.*

## 3 Anomaly Representation Based on Packet Space

### 3.1 Packet Space Segmentation and Classification

As we discussed in Section 2, existing anomaly detection methods could not accurately point out the anomaly portions caused by a set of overlapping rules. In order to precisely identify policy anomalies and enable a more effective anomaly resolution, we introduce a *rule-based segmentation technique*, which adopts a binary decision diagram (BDD)-based data structure to represent rules and perform various set operations, to convert a list of rules into a set of disjoint network packet spaces. This technique has been recently introduced to deal with several research problems such as network traffic measurement [9], firewall testing [10] and optimization [11]. Inspired by those successful applications, we leverage this technique for the purpose of firewall policy anomaly analysis. Algorithm 1 shows the pseudocode of generating packet space segments for a set of firewall rules $R$.[2] This algorithm works by adding a network packet space $s$ derived from a rule $r$ to a packet space set $S$. A pair of packet spaces must satisfy one of the following relations: *subset* (line 5), *superset* (line 10), *partial match* (line 13), or *disjoint* (line 17). Therefore, one can utilize set operations to separate the overlapped spaces into disjoint spaces.

**Algorithm 1**: Segment Generation for Network Packet Space of a Set of Rule $R$: Partition(R)

```
   Input: A set of rules, $R$.
   Output: A set of packet space segments, $S$.
1  foreach $r \in R$ do
2      $s_r \longleftarrow PacketSpace(r)$;
3      foreach $s \in S$ do
4          /* $s_r$ is a subset of $s$*/
5          if $s_r \subset s$ then
6              $S.Append(s \setminus s_r)$;
7              $s \longleftarrow s_r$;
8              $Break$;
9          /* $s_r$ is a superset of $s$*/
10         else if $s_r \supset s$ then
11             $s_r \longleftarrow s_r \setminus s$;
12         /* $s_r$ partially matches $s$*/
13         else if $s_r \cap s \neq \emptyset$ then
14             $S.Append(s \setminus s_r)$;
15             $s \longleftarrow s_r \cap s$;
16             $s_r \longleftarrow s_r \setminus s$;
17     $S.Append(s_r)$;
18 return $S$;
```

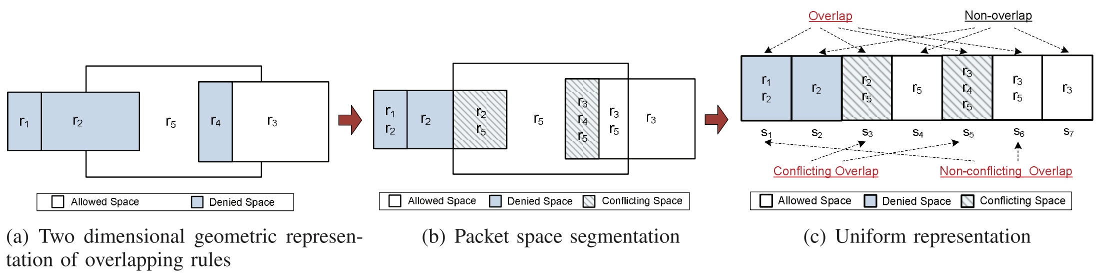
A set of segments $S : \{s_1, s_2, \ldots, s_n\}$ from firewall rules has the following two properties:

1. All segments are pairwise disjoint: $s_i \cap s_j = \emptyset$, $where\ 1 \leq i \neq j \leq n$; and
2. Any two different network packets $p$ and $p'$ within the same segment ($s_i$) are matched by the exact same set of rules: $GetRule(p) = GetRule(p'), \forall p \in s_i, p' \in s_i, p \neq p'$, where $GetRule()$ is a function to return all matched rules of a network packet.

To facilitate the correct interpretation of analysis results, a concise and intuitive representation method is necessary. For the purposes of brevity and understandability, we employ a two-dimensional geometric representation for each packet space derived from firewall rules. Note that a firewall rule typically utilizes five fields to define the rule condition; thus, a complete representation of packet space should be multidimensional. Fig. 1a gives the two-dimensional geometric representation of packet spaces derived from the example policy shown in Table 1. We utilize colored rectangles to denote two kinds of packet spaces: *allowed space* (white color) and *denied space* (gray color), respectively. In this example, there are two allowed spaces representing rules $r_3$ and $r_5$, and three denied spaces depicting rules $r_1$, $r_2$, and $r_4$.

Two spaces overlap when the packets matching two corresponding rules intersect. For example, $r_5$ overlaps with $r_2$, $r_3$, and $r_4$, respectively. An overlapping relation may involve multiple rules. In order to clearly represent all identical packet spaces derived from a set of overlapping

2. Similar partitioning functionalities were addressed in [9], [10] as well.



(a) Two dimensional geometric representation of overlapping rules (b) Packet space segmentation (c) Uniform representation

Fig. 1. Packet space representation derived from the example policy.

rules, we adopt the rule-based segmentation technique addressed in Algorithm 1 to divide an entire packet space into a set of pairwise disjoint segments. We classify the policy segments as follows: *nonoverlapping* segment and *overlapping* segment, which is further divided into *conflicting overlapping* segment and *nonconflicting overlapping* segment. Each *nonoverlapping* 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 (*nonconflicting overlapping* segment). Fig. 1b demonstrates the segments of packet spaces derived from the example policy. Since the size of segment representation does not give any specific benefits in resolving policy anomalies, we further present a uniform representation of space segments in Fig. 1c. We can notice that *seven* unique disjoint segments are generated. Three policy segments $s_2$, $s_4$, and $s_7$ are *nonoverlapping* segments. Other policy segments are *overlapping* segments, including two *conflicting overlapping* segments $s_3$ and $s_5$, and two *nonconflicting overlapping* segments $s_1$ and $s_6$.

## 3.2 Grid Representation of Policy Anomaly

To enable an effective anomaly resolution, complete and accurate anomaly diagnosis information should be represented in an intuitive way. When a set of rules interacts, one overlapping relation may be associated with several rules. Meanwhile, one rule may overlap with multiple other rules and can be involved in a couple of overlapping relations (overlapping segments). Different kinds of segments and associated rules can be viewed in the uniform representation of anomalies (Fig. 1c). However, it is still difficult for an administrator to figure out how many segments one rule is involved in. To address the need of a more precise anomaly representation, we additionally introduce a grid representation that is a *matrix-based* visualization of policy anomalies, in which space segments are displayed along the horizontal axis of the matrix, rules are shown along the vertical axis, and the intersection of a segment and a rule is a grid that displays a rule's subspace covered by the segment.

Fig. 2 shows a grid representation of policy anomalies for our example policy. We can easily determine which rules are covered by a segment, and which segments are associated with a rule. For example, as shown in Fig. 2, we can notice that a conflicting segment (CS) $s_5$, which points out a conflict, is related to a rule set consisting of three conflicting rules $r_3$, $r_4$, and $r_5$ (highlighted with a horizontal red rectangle), and a rule $r_3$ is involved in three segments $s_5$, $s_6$, and $s_7$ (highlighted with a vertical red rectangle). Our grid representation provides a better understanding of policy anomalies to system administrators with an overall view of related segments and rules.

## 4 Anomaly Management Framework

Our policy anomaly management framework is composed of two core functionalities: *conflict detection and resolution*, and *redundancy discovery and removal*, as depicted in Fig. 3. Both functionalities are based on the rule-based segmentation technique. For conflict detection and resolution, conflicting segments are identified in the first step. Each conflicting segment associates with a policy conflict and a set of conflicting rules. Also, the correlation relationships among conflicting segments are identified and conflict correlation groups (CG) are derived. Policy conflicts belonging to different conflict correlation groups can be resolved separately; thus, the searching space for resolving conflicts is reduced by the correlation process. The second step generates an action constraint for each conflicting segment by examining the characteristics of each conflicting segment. A strategy-based method is introduced for generating action

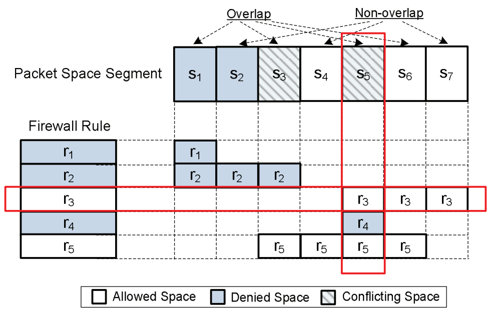
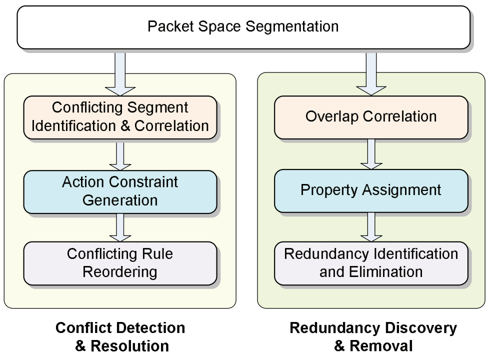


Fig. 2. Grid representation of policy anomaly.



Packet Space Segmentation
Conflicting Segment Identification & Correlation
Action Constraint Generation
Conflicting Rule Reordering
Overlap Correlation
Property Assignment
Redundancy Identification and Elimination
Conflict Detection & Resolution
Redundancy Discovery & Removal

Fig. 3. Policy anomaly management framework.

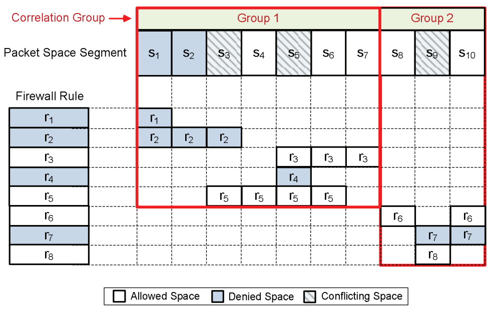


Fig. 4. Example of segment correlation.

constraints. The third step utilizes a reordering algorithm, which is a combination of a permutation algorithm and a greedy algorithm, to discover a near-optimal conflict resolution solution for policy conflicts. Regarding redundancy discovery and removal, segment correlation groups are first identified. Then, the process of property assignment is performed to each rule's subspaces. Consequently, redundant rules are identified and eliminated.

## 4.1 Correlation of Packet Space Segment

Technically, one rule may get involved in multiple policy anomalies. In this case, resolving one anomaly in an isolated manner may cause the unexpected impact on other anomalies. Similarly, we cannot resolve a conflict individually by only reordering conflicting rules associated with one conflict without considering possible impacts on other conflicts. On the other hand, it is also inefficient to deal with all conflicts together by reordering all conflicting rules simultaneously. Therefore, it is necessary to identify the dependency relationships among packet space segments for efficiently resolving policy anomalies.

Fig. 4 shows an example of segment correlation.[3] Suppose we add three new rules $r_6$, $r_7$, and $r_8$ in the example policy shown in Table 1. Several rules in this firewall policy are involved in multiple anomalies. For example, $r_2$ is associated with three segments $s_1$, $s_2$, and $s_3$. Also, we can identify $r_3$, $r_5$, $r_6$, and $r_7$ are also associated with multiple segments. Assume we need to resolve the conflict related to a conflicting segment $s_3$ by reordering associated conflicting rules, $r_2$ and $r_5$. The position change of $r_2$ and $r_5$ would also affect other segments, $s_1$, $s_2$, $s_4$, $s_5$, and $s_6$. Thus, a dependency relationship among those segments can be derived. We cluster such segments with a dependency relationship as a group called *correlation group*. Consequently, two correlation groups, $group1$ and $group2$, can be identified in our example as shown in Fig. 4: $group1$ contains seven segments and a rule set with five elements ($r_1$, $r_2$, $r_3$, $r_4$, and $r_5$); and $group2$ includes three segments and three associated rules, $r_6$, $r_7$, and $r_8$.

The major benefit of generating correlation groups for the anomaly analysis is that anomalies can be examined within each group independently, because all correlation groups are independent of each other. Especially, the searching space for reordering conflicting rules in conflict resolution can be significantly lessened and the efficiency of resolving conflicts can be greatly improved.

3. Note that for conflict resolution we only need to examine the correlation relations among conflicting segments.

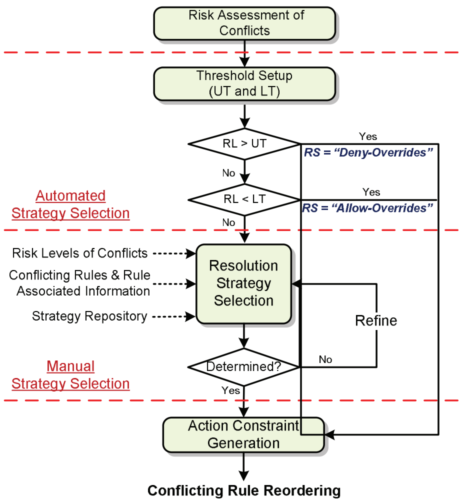

Risk Assessment of Conflicts
Threshold Setup (UT and LT)
RL > UT
Yes
RS = "Deny-Overrides"
No
Automated Strategy Selection
RL < LT
Yes
RS = "Allow-Overrides"
No
Risk Levels of Conflicts
Conflicting Rules & Rule Associated Information
Strategy Repository
Resolution Strategy Selection
Refine
Manual Strategy Selection
Determined?
No
Yes
Action Constraint Generation
Conflicting Rule Reordering

Fig. 5. Strategy-based conflict resolution.

## 4.2 Conflict Resolution

Each conflicting segment indicates a policy conflict as well as a set of conflicting rules involved in the conflict. Once conflicts are identified, a possible way for a system administrator to resolve conflicts is to manually change the conflicting rules. However, as we addressed in Section 1, resolving all conflicts manually is a tedious task and even impractical due to the complicated nature of policy conflicts. Thus, a practical and effective method to resolve a policy conflict is to determine which rule should take precedence when a network packet is matched by a set of rules involved in the conflict. In order to utilize the existing first-match conflict resolution mechanism implemented in common firewalls, the rule expected to take precedence needs to be moved to the first-match rule.

Our conflict resolution mechanism introduces that an *action constraint* is assigned to each conflicting segment. An action constraint for a conflicting segment defines a desired action (either `Allow` or `Deny`) that the firewall policy should take when any packet within the conflicting segment comes to the firewall. Then, to resolve a conflict, we only assure that the action taken for each packet within the conflicting segment can satisfy the corresponding action constraint. A key feature of this solution is that we do not need to move a rule expected to take precedence to the first-match rule at all times. Any rule associated with the conflict on the same action (as a rule with the precedence) can be moved to the first-match rule, guaranteeing the same effect with respect to the conflict resolution. Thus, it is doable to obtain an optimal solution for conflict resolution.

### *4.2.1 Action Constraint Generation*

To generate action constraints for conflicting segments, we propose a strategy-based conflict resolution method, which generates action constraints with the help of effective resolution strategies based on the minimal interaction with system administrators. Fig. 5 shows the main processes of this method, which incorporates both *automated* and *manual* strategy selections.

Once conflicts in a firewall policy are discovered and conflict correlation groups are identified, the risk assessment for conflicts is performed. The risk levels (RL) of conflicts are in turn utilized for both automated and manual strategy selections. A basic idea of automated strategy selection is that a risk level of a conflicting segment is used to directly determine the expected action taken for the network packets in the conflicting segment. If the risk level is very high, the expected action should deny packets considering the protection of network perimeters. On the contrary, if the risk level is quite low, the expected action should allow packets to pass through the firewall so that the availability and usage of network services cannot be affected. Thus, conflict resolution strategies ($RS$) can be generated automatically for *partial* conflict segments by comparing the risk levels with two thresholds, upper threshold ($UT$) and lower threshold ($LT$), which can be set by system administrators in advance based on the different situations of protected networks. If a risk level of a conflicting segment is between the upper threshold and the lower threshold, system administrators need to examine the characteristics of each conflict, and manually select appropriate strategies for resolving the conflict, considering both network situations (e.g., risk levels of conflicts) and contexts associated with conflicting rules (e.g., priorities, creation time, authors, and so on). Thus, a fine-grained conflict resolution can be carried out with human cognition via the interaction facility with system administrators. Since some strategies may be nondeterministic and could not generate a concrete action constraint when applying to a conflict, system administrators need to adjust the strategy assignments accordingly. As long as all conflicts within a conflict correlation group are associated with desired action constraints, these conflicts will be ultimately resolved by reordering conflicting rules to satisfy corresponding action constraints.

Risk (security) levels are determined based on the vulnerability assessment of the protected network. We have recently seen a number of attempts for qualitatively measuring risks in a network [12], [13], [14]. In our work, we adopt the Common Vulnerability Scoring System (CVSS) [15] as an underlying security metrics for risk evaluation. Two major factors, *exploitability of vulnerability* (reflecting the likelihood of exploitation) and *severity of vulnerability* (representing the potential damage of exploitation), are utilized to evaluate the risk level of a network system. Beside those two factors, another important factor in determining the criticality of an identified security problem is *asset importance value*. Normally, system administrators place a higher priority on defending critical servers than noncritical PCs. Similarly, some machines are more valuable than others. We use *asset importance value* to represent a service's inherent value to network attackers or system administrators. Since the CVSS *base score* can cover both *exploitability of vulnerability* and *severity of vulnerability* factors, we incorporate the CVSS *base score* and *asset importance value* to compute the risk value for each vulnerability as follows:

$$Risk\ Value = (CVSS\ Base\ Score) \times (Importance\ Value). \tag{1}$$

To calculate the risk level of each conflicting segment, we accumulate all risk values of the vulnerabilities covered by a conflicting segment. In practice, system administrators may mainly concern about the security risk (SR) of each vulnerability in their network. In this case, an *average* risk value needs to be calculated as the risk level of a conflicting segment. In order to accommodate both requirements for risk evaluation of a conflicting segment, we introduce a generic equation for the risk level calculation as follows:

$$RL(cs) = \frac{\sum_{v \in V(cs)} (CVSS(v) \times IV(s))}{\alpha \times |V(cs)|}, \tag{2}$$

where $V(cs)$ is a function to return all vulnerabilities that are contained in a conflicting segment $cs$; $CVSS(v)$ is a function to return the CVSS *base score* of vulnerability $v$; and $IV(s)$ is a function to return the importance value (with the range from 0.0 to 1.0) of service $s$. Also, we incorporate a coefficient factor $\alpha$ ($\frac{1}{|V(cs)|} \leq \alpha \leq 1$) that allows system administrators to express their preferences in choosing *average* or *overall* risk value to measure the risk of each conflicting segment. The value of $\alpha$ can be decreased, in which case an administrator cares more about the *overall* risk value. Otherwise, she/he can increase the value of $\alpha$ until it reaches an *average* risk value.

Some general conflict resolution strategies for access control have been introduced [16], [17], [18]. We classify conflict resolution strategies for firewall policies into two categories: network situation-aware strategy and policy-oriented strategy. Note that other user-defined strategies [19] can be implied in our conflict resolution mechanism as well.

**Network situation-aware strategy**. A system administrator adopts this strategy to resolve policy conflicts based on the results of risk assessment of the network covered by corresponding conflicting segments

- *Deny-overrides*. This strategy indicates that "deny" rules take precedence over "allow" rules. In general, a system administrator may directly take this strategy to harden his network if the risk level of a conflict is very high.
- *Allow-overrides*. This strategy states that "allow"rules take precedence over "deny" rules. A system administrator may apply this strategy to resolve a conflict, which has a lower risk level.

**Policy-oriented strategy**. This strategy considers the factors related to the policy definition, such as when was the rule defined? what was the rule defined for? and who defined the rule?

- *Recency-overrides*. This strategy indicates that rules take precedence over rules specified earlier. As the security requirements may change over a period of time, an administrator may define new rules along with his evolving security requirements which may be in conflict with previous security requirements. Obviously, in this case, newer rules should take precedence over older rules.
- *Specificity-overrides*. This strategy states that a more specific rule overrides more general rules. In a firewall policy, shadowing and generalization conflicts can be identified by conflict detection tools. In our solution, we treat shadowing and generalization conflicts as the same case, since we resolve them through rule reordering.
- *High-majority-overrides*. This strategy allows (denies, respectively) a packet if the number of rules taking

TABLE 2
Constraint Generation from Conflict Resolution Strategy

| Strategy | Action Constraint |
|---|---|
| Deny-overrides | Action = "deny" |
| Allow-overrides | Action = "allow" |
| Recency-overrides | Action of the newest rule |
| Specificity-overrides | Action of the most specific rule |
| High-majority-overrides | Action of the rules with greater number than the opposite rules |
| First-match-overrides | Action of the first-matched rule |
| High-authority-overrides | Action of the rule with the highest authority level |

"allow" ("deny," respectively) action is greater than the number of rules taking "allow" ("deny," respectively) action.

- *First-match-overrides*. This strategy states that the first matched rule takes precedence over others. This strategy can be directly converted to the existing firewall implementation.
- *High-authority-overrides*. This strategy states that a rule defined by an administrator with a higher authority level takes precedence.

Since some strategies are nondeterministic and adopting only one strategy may not guarantee solving the conflict, it is desirable to combine multiple strategies together to fulfil the requirements of conflict resolution posed by a policy. Our previous work in [20], [21] proposed a strategy chain to address this issue and we incorporate such a strategy chain in our firewall conflict resolution mechanism. Once each conflicting segment has been assigned to appropriate conflict resolution strategies, action constraints can be generated based on the assigned strategies. Table 2 summarizes the constraint generation from those strategies.

### *4.2.2 Rule Reordering*

The most ideal solution for conflict resolution is that all action constraints for conflicting segments can be satisfied by reordering conflicting rules. In other words, if we can find out conflicting rules in order that satisfies all action constraints, this order must be the optimal solution for the conflict resolution. Unfortunately, in practice action constraints for conflicting segments can only be satisfied partially in some cases.

Fig. 6 illustrates a scenario representing that action constraints cannot be fully satisfied. In this scenario, four rules intersect with each other in different conflicting segments. The existing order of rules cannot satisfy the fourth action constraint. In order to make the fourth action constraint satisfied, $r_4$ needs to be moved in front of $r_1$. However, this situation causes the violation against the

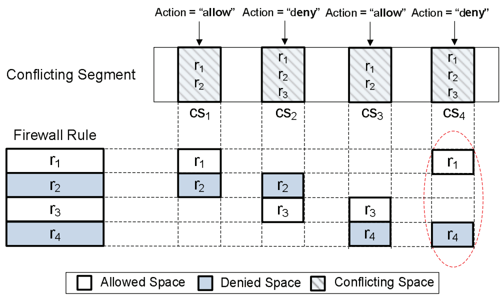


Fig. 6. Partial satisfaction of action constraints.

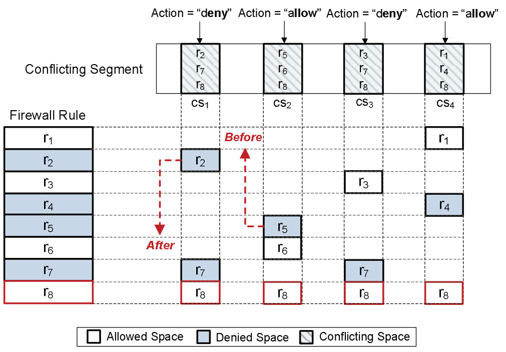


Fig. 7. Example of position indicators for a rule.

third action constraint. We can easily observe that all action constraints cannot be satisfied simultaneously by any permutation of conflicting rules in this scenario.

A naive way to find an optimal solution is to exhaustively search all permutations of correlated conflicting rules. We then compute a resolving score for each permutation by counting how many action constraints can be satisfied, and select the permutation with the maximum resolving score as the best solution for a conflict resolution. However, a key limitation of using the permutation algorithm is its computational complexity which is $O(n!)$. Even though the search space can be significantly reduced by applying our correlation scheme, the number of correlated conflicting rules may still be large, leading to the permutation algorithm unapplicable. Since the permutation algorithm is time intensive and can be only used to identify an optimal reordering for a small set of correlated conflicting rules, an approximation algorithm is more desirable. Although approximation algorithms can only find a near-optimal solution, they are more efficient in finding a solution comparing to the permutation algorithm. To address this issue, we introduce a greedy algorithm, which can be employed to resolve the conflicts containing a larger number of correlated conflicting rules.

A greedy algorithm makes the locally optimal choice at each stage with the hope of finding the global optimum. For all conflicting rules in a correlation group, our greedy conflicting resolution algorithm first calculates a resolving score for each conflicting rule individually. Then, the rule with the greatest resolving score is selected to solve the conflicts: a position range with the best conflict resolution is identified for the selected rule; and moving the selected rule to the new position achieves a locally optimal conflicting resolution. Applying the same processes to the remaining rules recursively until all rules in the correlation group are processed, the final order of the correlated conflicting rules is a solution of the conflict resolution. Note that the resolving scores for the remainder of rules should be recalculated, since moving a rule may change the original conflicting situation of a policy.

In our greedy algorithm, a critical process is to calculate the resolving score for each conflicting rule within a conflict correlation group. This process contains four steps as follows:

1. *Generating position indicators for each conflicting segment*. A position indicator of a rule for a conflicting segment indicates a position range in which this rule can stay so that the action constraint of the conflicting segment is satisfied. Fig. 7 gives an example, which

demonstrates the generation of position indicators for a rule $r_8$. The rule $r_8$ in this example involves in four conflicts. In order to fulfill the action constraint of the first conflicting segment ($cs_1$), $r_8$ should stay after $r_2$. Thus, a position indicator ($pi_1$) can be generated as: $Position(r_8) > 2$. To satisfy the action constraint of the second conflicting segment, $r_8$ needs to be moved in front of $r_5$. Another position indicator ($pi_2$) is generated as: $Position(r_8) \leq 5$.

2. *Generating position ranges*. Based on the position indicators generated for a rule, we sort the range bounds in an ascending order, and then employ a plane sweeping technique [22] to obtain the disjoint position ranges ($pr$) for the rule. Using the same example in Fig. 7, two position indicators $pi_1$ and $pi_2$ can identify three ranges for the selected rule: $pr_1 : [1, 2]$, $pr_2 : [2, 5]$, and $pr_3 : [5, 8]$.
3. *Calculating a resolving score for each position range*. The resolving score of a position range represents the impact of resolving conflicts when moving the selected rule to this position range. Moving the selected rule to a position range satisfies some action constraints, but may also violate some other action constraints. Thus, the resolving score for a position range with respect to the selected rule can be calculated by subtracting the number of violated action constraints from the number of satisfied action constraints. For example, if we move $r_8$ to the range $pr_1$. The resolving score in this case is equal to 0, since the action constraint of $cs_1$ will be satisfied but the action constraint of $cs_2$ will be violated in such a particular situation.
4. *Choosing the maximum range score as the resolving score of selected rule*. Since one rule may have multiple associated position ranges. In order to achieve the effectiveness of conflict resolution, it is necessary to move the rule to a position in the range with the maximum resolving score. Thus, we use the maximum resolving score of a position range to represent the resolving score of selected rule.

In order to achieve the objective of resolving conflicts effectively and efficiently, our conflict resolution mechanism adopts a combination algorithm[4] incorporating features from both permutation and greedy algorithms. A threshold $\mathcal{N}$ for selecting a suitable rule reordering algorithm to resolve a conflict can be predefined in the combination algorithm. When the number of conflicting rules is less than $\mathcal{N}$, the permutation algorithm is utilized for resolving conflicts. Otherwise, the greedy algorithm is applied to resolve conflicts.

### 4.3 Redundancy Elimination

In this step, every rule subspace covered by a policy segment is assigned with a property. Four property values, *removable* (R), *strong irremovable* (SI), *weak irremovable* (WI), and *correlated* (C), are defined to reflect different characteristics of each rule subspace. *Removable* property is used to indicate that a rule subspace is removable. In other words, removing such a rule subspace does not make any impact

4. The pseudocode of conflict resolution algorithm is omitted in this paper due to the space limitation.

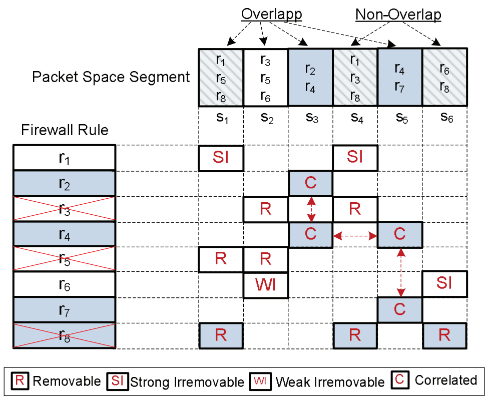


Fig. 8. Example of property assignment.

on the original packet space of an associated policy. *Strong irremovable* property means that a rule subspace cannot be removed because the action of corresponding policy segment can be decided only by this rule. *Weak irremovable* property is assigned to a rule subspace when any subspace belonging to the same rule has *strong irremovable* property. That means a rule subspace becomes irremovable due to the reason that other portions of this rule cannot be removed. *Correlated* property is assigned to multiple rule subspaces covered by a policy segment, if the action of this policy segment can be determined by any of these rules. We next introduce three processes to perform the property assignments to all of rule subspaces within the segments of a firewall policy, considering different categories of policy segments discussed in Section 3.1

1. *Property assignment for the rule subspace covered by a nonoverlapping segment*. A nonoverlapping segment contains only one rule subspace. Thus, this rule subspace is assigned with *strong irremovable* property. Other rule subspaces associated with the same rule are assigned with *weak irremovable* property, except for the rule subspaces that already have *strong irremovable* property.
2. *Property assignment for rule subspaces covered by a conflicting segment*. The first rule subspace covered by the conflicting segment is assigned with *strong irremovable* property. Other rule subspaces in the same segment are assigned with *removable* property. Meanwhile, other rule subspaces associated with the first rule are assigned with *weak irremovable* property except for the rule subspaces with *strong irremovable* property.
3. *Property assignment for rule subspaces covered by a nonconflicting overlapping segment*. If any rule subspace has been assigned with *weak irremovable* property, other rule subspaces without *any irremovable* property are assigned with *removable* property. Otherwise, all subspaces within the segment are assigned with *correlated* property.

Fig. 8 illustrates the result of applying our property assignment approach, which performs three property assignment processes in sequence, to a firewall policy with eight rules. We can easily identify that three rules, $r_3$, $r_5$, and $r_8$, are *removable* rules, where the *removable* property is assigned to

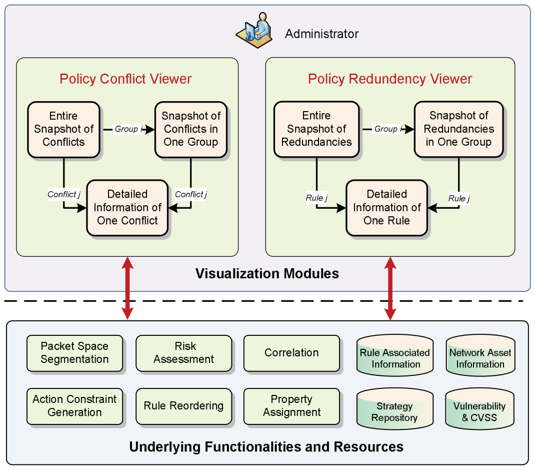

Administrator
Policy Conflict Viewer
Entire Snapshot of Conflicts
Group
Snapshot of Conflicts in One Group
Conflict j
Detailed Information of One Conflict
Conflict j
Policy Redundency Viewer
Entire Snapshot of Redundancies
Group
Snapshot of Redundancies in One Group
Rule j
Detailed Information of One Rule
Rule j
Visualization Modules
Packet Space Segmentation
Risk Assessment
Correlation
Rule Associated Information
Network Asset Information
Action Constraint Generation
Rule Reordering
Property Assignment
Strategy Repository
Vulnerability & CVSS
Underlying Functionalities and Resources

Fig. 9. Architecture of FAME.

all subspaces. In addition, by examining the *correlated* rules $r_2$, $r_4$, and $r_7$, which contain some subspaces with *correlated* property, $r_2$ and $r_7$ can be identified as redundant rules and removed from the policy. However, if we leverage *traditional* redundancy detection method [1], [6], which is limited to detect *pairwise* redundancies, to this example, only two redundant rules $r_2$ and $r_7$ can be discovered.

## 5 Implementation and Evaluation

Our framework is realized as a proof-of-concept prototype called Firewall Anomaly Management Environment. Fig. 9 shows a high-level architecture of FAME with two levels. The upper level is the visualization layer, which visualizes the results of policy anomaly analysis to system administrators. Two visualization interfaces, *policy conflict viewer* and *policy redundancy viewer*, are designed to manage policy conflicts and redundancies, respectively. The lower level of the architecture provides underlying functionalities addressed in our policy anomaly management framework and relevant resources including rule information, strategy repository, network asset information, and vulnerability information.

### 5.1 Implementation of Anomaly Management Framework in FAME

FAME was implemented in Java. Based on our policy anomaly management framework, it consists of six components: segmentation module, correlation module, risk assessment module, action constraint generation module, rule reordering module, and property assignment module. The segmentation module takes firewall policies as an input and identifies the packet space segments by partitioning the packet space into disjoint subspaces. FAME utilizes Ordered Binary Decision Diagrams[5] to represent firewall rules and perform various set operations, such as unions ($\cup$), intersections ($\cap$), and set differences ($\setminus$), required by the segmentation algorithm. A BDD library called JavaBDD [24], which is based on BuDDy package [25], is employed by FAME. Once the segmentation of packet space is identified, FAME further identifies different kinds of segments and corresponding correlation groups. In risk assessment module, Nessus [26] is utilized as a *vulnerability scanner* to identify the vulnerabilities within a conflicting segment. Network address space of each conflicting segment is fed into Nessus to get the vulnerability information of a given address space. Nessus produces the vulnerability information in a “nbe” format. The risk assessment module utilizes tissynbe script [27] to parse the Nessus results and store the vulnerability information to a *vulnerability database*. A *risk calculator* retrieves vulnerability information, such as CVSS base score and asset importance value, to calculate the risk level of each conflicting segment.

### 5.2 Implementation of Visualization Interfaces in FAME

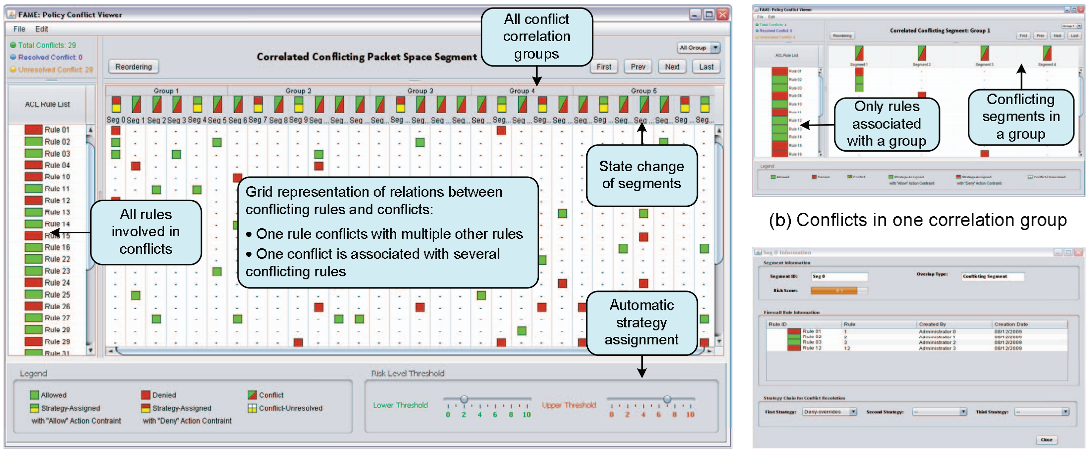
FAME provides two policy viewers to visualize the outputs of policy conflict analysis and policy redundancy analysis. Each viewer offers two kinds of visualization interfaces: one interface shows an entire snapshot of all anomalies; another interface shows a partial snapshot only containing anomalies within one correlation group.

Fig. 10 depicts interfaces of FAME conflict viewer. The grid representation shows accurately how a set of rules interacts with each other. FAME conflict viewer has the ability to show an overview of the entire conflicts as well as portions of the policy conflicts, that need to be examined in depth for conflict resolution, based on correlation groups. As illustrated in Fig. 10a, all conflicting segments and conflict correlation groups are displayed along the horizontal axis at the top of the interface. All conflicting rules are shown along the vertical axis at the left of the interface. Each grid cell represents a rule’s subspace. In our interface, the icons for conflicting segments indicate four different states with respect to conflicting resolution. One icon represents a conflicting segment with the state of strategy unassigned. Two other icons indicate conflicting segments with the state of strategy assigned with “Allow” action constraint and strategy assigned with “Deny” action constraint, respectively. The fourth icon indicates a conflicting segment with the state of conflict unresolved. In addition, this interface allows an administrator to set the risk level thresholds for automatically assigning strategies.

Clicking on a group name box of the interface in Fig. 10a, another window as shown in Fig. 10b is displayed with the targeted conflicts that an administrator needs to examine and resolve. In this interface, the number of visible entities is reduced to only display conflicting segments in one correlation group and a list of conflicting rules associated with this group. This significantly eliminates administrators’ workloads in resolving conflicts by highlighting conflicts within a group. For resolution strategy selection, the administrator needs to further examine rule information for selecting suitable strategies for each conflicting segment. When the administrator clicks the icon of a conflicting segment, the detailed information related to the conflict is displayed in a window as shown in Fig. 10c.[6]

5. BDD has been demonstrated as an efficient data structure to deal with a variety of network configuration analysis [5], [23].

6. FAME redundancy viewer was also developed in a similar fashion. We elide the discussion of redundancy viewer in this paper.


All conflict correlation groups
All rules involved in conflicts
Grid representation of relations between conflicting rules and conflicts:
• One rule conflicts with multiple other rules
• One conflict is associated with several conflicting rules
State change of segments
Automatic strategy assignment
Only rules associated with a group
Conflicting segments in a group

(b) Conflicts in one correlation group

(a) Conflict Viewer for all conflicts

(c) Information of one conflict

Fig. 10. Interface of FAME conflict viewer.

## 5.3 Evaluation of FAME

For FAME evaluation, we utilized a number of firewall policies and associated information required by our tool from different resources. Most of them are from campus networks and some are from major ISPs. Our experiments were performed on Intel Core 2 Duo CPU 3.00 GHz with 3.25 GB RAM running on Linux kernel 2.6.16.

### *5.3.1 Evaluation of Conflicting Segment Generation and Correlation*

Table 3 shows the evaluation results generated by the segmentation and correlation engine of FAME. The number of conflicting segments, the number of conflict correlation groups, the number of large conflict correlation groups (the rule number is greater than *six*) and the number of conflicting rules in the largest correlation group are given in this table, which also contains the execution time required by the segmentation module of FAME for identifying conflicting segments (i.e., detecting conflicts), as well as the one required by the correlation module of FAME for identifying correlation groups among conflicting segments. Note that all measurements were based on the system time stamps in our experiments.

In Table 3, the number of large conflict correlation groups and the number of conflicting rules in the largest correlation group give us the evidences that manual conflict resolution for a large size of firewall policies is almost impossible. Also, we can observe that the segmentation and correlation processes are efficient enough to handle a larger size of firewall policies, such as policy $G$ and policy $H$ in the table.

### *5.3.2 Evaluation of Conflicting Rule Reordering Algorithm*

We have addressed that permutation and greedy algorithms can be used for reordering conflicting rules, and our conflict resolution mechanism utilizes a combination algorithm incorporating the features of both permutation and greedy algorithms to achieve a more effective and efficient conflict resolution. In order to evaluate our proposed method, we measured the effectiveness and efficiency of three algorithms implemented in the rule reordering module of FAME using two metrics, *resolved conflicts* (RC) and *resolving time*.

Table 4 summarizes our evaluation results. It shows that the permutation algorithm can always achieve an optimal conflict resolution for all policies except policy $H$. We were unable to resolve the conflicts in policy $H$ using the permutation algorithm, because there exist a larger size of conflicting rules in some correlation groups. From Table 3, we can notice that the number of the largest group member of policy $H$ is *eighteen*. Also, it shows that the resolving time required by the permutation algorithm increases exponentially as the number of conflicting segments increases. Hence, the permutation algorithm is infeasible to the policies with a large size of conflicting rules, although it can achieve an optimal solution.

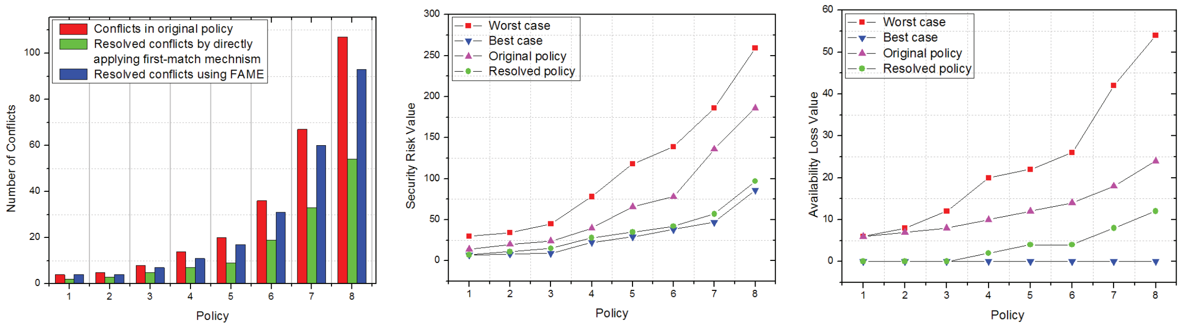
Regarding the greedy algorithm, Table 4 shows that it can only achieve a near-optimal conflict resolution for all firewall policies. However, as the size of conflicting rules increases, the time taken by the greedy algorithm increases almost linearly as opposed to an exponential increase in case of using the permutation algorithm.

TABLE 3
Segmentation and Correlation Evaluation

| Policy | Rules (#) | Segmentation | | Correlation | | Large Group (#) ($n >= 6$) | Largest Group Member (#) |
|---|---|---|---|---|---|---|---|
| | | CS (#) | Time (s) | CG (#) | Time (s) | | |
| 1 (A) | 12 | 4 | 0.134 | 2 | 0.056 | 0 | 0 |
| 2 (B) | 18 | 5 | 0.186 | 3 | 0.073 | 0 | 0 |
| 3 (C) | 25 | 8 | 0.233 | 3 | 0.081 | 1 | 6 |
| 4 (D) | 52 | 14 | 0.377 | 7 | 0.094 | 1 | 7 |
| 5 (E) | 83 | 20 | 0.427 | 9 | 0.118 | 2 | 7 |
| 6 (F) | 132 | 36 | 0.518 | 10 | 0.143 | 3 | 9 |
| 7 (G) | 354 | 67 | 0.854 | 10 | 0.189 | 5 | 12 |
| 8 (H) | 926 | 107 | 2.386 | 13 | 0.437 | 8 | 18 |

TABLE 4
Rule Reordering Algorithm Evaluation

| Policy | Conflicting Segments (#) | Permutation | | Greedy | | Combination | |
|---|---|---|---|---|---|---|---|
| | | RC | Time (s) | RC | Time (s) | RC | Time (s) |
| 1 (A) | 4 | 4 | 0.759 | 3 | 0.498 | 4 | 0.743 |
| 2 (B) | 5 | 4 | 0.789 | 3 | 0.528 | 4 | 0.789 |
| 3 (C) | 8 | 7 | 0.923 | 6 | 0.758 | 7 | 0.897 |
| 4 (D) | 14 | 12 | 1.976 | 9 | 1.257 | 11 | 1.754 |
| 5 (E) | 20 | 18 | 3.734 | 15 | 1.538 | 17 | 2.897 |
| 6 (F) | 36 | 33 | 10.979 | 24 | 7.567 | 31 | 9.239 |
| 7 (G) | 67 | 62 | 973.653 | 57 | 38.289 | 60 | 96.756 |
| 8 (H) | 107 | $\infty$ | $\infty$ | 86 | 103.307 | 93 | 253.76 |



(a) Resolution rate (b) Risk reduction (c) Availability improvement

Fig. 11. Evaluation of conflict resolution.

For the combination algorithm with the default threshold ($\mathcal{N} = 6$), the results in Table 4 show that the number of resolved conflicts by the combination algorithm is greater than the greedy algorithm and almost equal to the optimal solution achieved by the permutation algorithm. The computation time is acceptable for all policies as well. Therefore, it represents higher efficiency and effectiveness in conflict resolution.

### *5.3.3 Evaluation of the Effectiveness of Conflict Resolution Approach*

Three metrics, *resolution rate*, *risk reduction*, and *availability improvement*, were adopted to evaluate the quality of conflict-resolved policies generated by our conflict resolution approach. We adopted *average* risk value (from 0.0 to 10.0) as the risk level of each conflict segment, and fixed upper and lower thresholds as 8.0 and 2.0, respectively, in our experiments.

First, we evaluated the conflict resolution rate of our strategy-based approach, which is reflected by the number of resolved conflicts (i.e., satisfied action constraints). We compared the results of applying our strategy-based approach with the results of directly applying the existing first-match mechanism for conflict resolution. As shown in Fig. 11a, we could observe that directly applying the existing first-match mechanism can only solve an average 63 percent of conflicts. However, when applying our strategy-based approach, an average 92 percent of conflicts could be resolved in our experiments. Moreover, for some small-scale policies, we noticed that FAME was capable of resolving all policy conflicts.

In general, when conflicts in a policy are resolved, the risk value of the resolved policy should be reduced and the availability of protected network should be improved comparing with the situation prior to conflict resolution. To evaluate the risk reduction and availability improvement of our conflict resolution approach, we compared the results of *conflict-resolved* policies with the *original* policies as well as the *best case* and *worst case* with respect to the conflict resolution. The *best case* of a conflict resolution is achieved when all action constraints assigned to the conflicting segments can be satisfied. The *worst case* considering the security risk is that all packets covered by conflicting segments are allowed to pass through a firewall. And the *worst case* considering the availability is that all packets covered by conflicting segments assigned with "allow" action constraints are denied.

We evaluated each policy described in Table 3 based on the risk values caused by corresponding conflicts, considering four situations: *worst case*, *best case*, *original* policy, and *resolved* policy. The *security risk* value of a policy ($p$) can be calculated by aggregating the risk levels of conflicting segments that take "allow" action to any matched packets as follows:

$$SR(p) = \sum_{cs_i \in CS(p).isAllowed()} RL(cs_i). \tag{3}$$

Note that *CS(p)* returns all conflicting segments of a policy ($p$) and the function *isAllowed()* is used to identify all conflicting segments taking "allow" actions. From Fig. 11b, we noticed that the security risk values of the *conflict-resolved* policies are always reduced, compared to the security risk value of the *original* policies. Our experiments showed that FAME could achieve an average 45 percent of risk reduction. We could also notice that the security risk values of the *conflict-resolved* policies are very close to the security risk values of the *best case*. It further indicates that FAME can achieve a higher rate of conflict resolution.

To measure the impact of conflict resolution on network availability, we were able to calculate an *availability loss* (*AL*) value for each policy. Computing the availability loss value for a policy only need to consider the conflicting segments of the policy whose action constraints are "allow," but carrying out "deny" action to all matched network packets. Suppose these conflicting segments can be returned by a function *isForceDenied()* and we utilize the equation $(10 - RL(cs))$ to derive the availability value of a conflicting segment $cs$. The availability loss value of a policy is calculated as follows:

$$AL(p) = \sum_{cs_i \in CS(p).isForceDenied()} (10 - RL(cs_i)). \tag{4}$$

Evaluation results depicted in Fig. 11c clearly show that the availability loss value for each *conflict-resolved policy* is lower than that of corresponding *original* policy, which supports our hypothesis that resolving policy conflicts can always improve the availability of protected network.

### *5.3.4 Evaluation of the Effectiveness of Redundancy Removal Approach*

We also evaluated our redundancy analysis approach based on those experimental firewall policies. We conducted the evaluation of effectiveness by comparing our redundancy analysis approach with *traditional* redundancy analysis

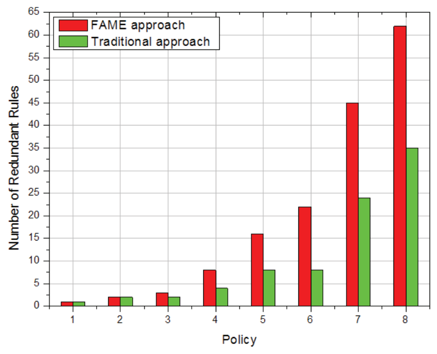


Fig. 12. Evaluation of redundancy removal.

approach [1], [6], which can only identify redundancy relations between *two* rules. Fig. 12 depicts the results of our comparison experiments. From Fig. 12, we observed that FAME could identify an average of 6.5 percent redundant rules from the whole rules. However, *traditional* redundancy analysis approach could only detect an average 3.8 percent of total rules as redundant rules. Therefore, the enhancement for redundancy elimination was clearly observed by our redundancy analysis approach compared to *traditional* redundancy analysis approach in our experiments.

## 5.REFERENCES

[1] A. Acquisti and R. Gross. Imagined communities: Awareness, information sharing, and privacy on the facebook. In Privacy Enhancing Technologies, pages 36{58, 2006.
[2] A. Acquisti and J. Grossklags. Privacy and rationality in individual decision making. IEEE Security and Privacy, 3(1):26{33, 2005.
[3] A. Besmer, J. Watson, and H. R. Lipford. The impact of social navigation on privacy policy conÿguration. In SOUPS, 2010.
[4] J. Bonneau and S. Preibusch. The privacy jungle: On the market for data protection in social networks. In The Eighth Workshop on the Economics of Information Security (WEIS 2009), 2009.
[5] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. Physical Review E, pages 1{ 6, 2004.
[6] E. Cutrell, M. Czerwinski, and E. Horvitz. Notiÿcation, disruption, and memory: Eÿects of messaging interruptions on memory and performance. pages 263{269. IOS Press, 2001.
[7] R. Dhamija and A. Perrig. Deja vu: A user study using images for authentication. In Proceedings of the 9th conference on USENIX Security Symposium - Volume 9, pages 4{4, Berkeley, CA, USA, 2000. USENIX Association.
[9] C. Dwyer, S. R. Hiltz, and K. Passerini. Trust and privacy concern within social networking sites: A comparison of facebook and myspace. In Proceedings of the Thirteenth Americas Conference on Information Systems ( AMCIS 2007), 2007. Paper 339.
[10] D. Ferraiolo and R. Kuhn. Role-based access control. In In 15th NIST-NCSC National Computer Security Conference, pages 554{563, 1992.
[11] L. Hubert and P. Arabie. Comparing partitions. Journal of classiÿcation, 2(1):193{218, 1985.
[12] S. T. Iqbal and B. P. Bailey. Investigating the eÿectiveness of mental workload as a predictor of opportune moments for interruption. In CHI '05 extended abstracts on Human factors in computing systems, CHI EA '05, pages 1489{1492, New York, NY, USA, 2005. ACM.
[13] Q. Jones, S. A. Grandhi, S. Whittaker, K. Chivakula, and L. Terveen. Putting systems into place: a qualitative study of design requirements for location-aware community systems. In In Proceedings of CSCW, pages 202{211. ACM, 2004.
[14] S. Jones and E. O'Neill. Feasibility of structural network clustering for group-based privacy control in social networks. In SOUPS, 2010.
[15] H. Krasnova, O. Gÿunther, S. Spiekermann, and K. Koroleva. Privacy concerns and identity in online social networks. Identity in the Information Society, 2:39{63, 2009.
[16] P. Kumaraguru and L. F. Cranor. Privacy indexes: A survey of westin's studies. ISRI Tech. Report, 2005.
[17] S. Lederer, J. I. Hong, A. K. Dey, and J. A. Landay. Personal privacy through understanding and action: ÿve pitfalls for designers. Personal and Ubiquitous Computing, 8(6):440{454, 2004.
[18] K. Lewis, J. Kaufman, and N. Christakis. The taste for privacy: An analysis of college student privacy settings in an online social network. Journal of Computer-Mediated Communication, 14(1), 2008.