

# Management and Translation of Filtering Security Policies

Ehab S. Al-Shaer and Hazem H. Hamed

Multimedia Networking Research Laboratory

School of Computer Science, Telecommunications and Information Systems

DePaul University, Chicago, USA

Email: {ehab, hhamed}@cs.depaul.edu

**Abstract**—Firewalls are essential elements for security policy enforcement in modern networks. However, managing a filtering security policy, especially for enterprise networks, has become complex and error-prone. Filtering rules have to be carefully written and organized in order to correctly implement the security policy and avoid policy anomalies. In this paper, we present a set of techniques and algorithms that provide (1) automatic anomaly discovery for rule conflicts and potential problems in legacy firewalls, (2) anomaly-free policy editing for rule insertion, modification and removal, and (3) concise translation of filtering rules to high-level textual description for user visualization and verification. These techniques significantly simplify the management of any generic firewall policy written as filtering rules, while minimizing network vulnerability due to filtering policy misconfiguration.

## I. INTRODUCTION

Although deployment of firewall technology is an important step toward securing our networks, the complexity of managing firewall rule policy might limit the effectiveness of firewall security. When the filtering rules are defined, serious attention has to be given to rule relations and interactions in order to determine the proper rule ordering and guarantee correct security policy semantics. As the number of filtering rules increases, the difficulty of writing a new rule or modifying an existing one also increases. It is very likely, in this case, to introduce conflicting rules that result in a different security policy. In addition, a typical large-scale enterprise network might involve hundreds of rules that might be written by different administrators in various times. This significantly increases the potential of conflicts in filtering rules and makes the network more vulnerable.

Therefore, the effectiveness of firewall security is dependent on providing policy management techniques and tools that enable network administrators to analyze, purify and verify the correctness of written firewall legacy rules. In this paper, we define a formal model for filtering rule relations and their filtering representation. The proposed model is simple and visually comprehensible. We use this model to develop an anomaly discovery algorithm to report any anomaly that exists among the filtering rules. We then develop anomaly-free filtering rule editor, which greatly simplifies adding and modifying rules into a filtering policy. We finally develop a policy translator that gives a concise textual description of the entire policy rules for user verification.

Although firewall security has been given strong attention in the research community, the emphasis was mostly on the filtering performance and hardware support issues [5], [6], [10]. On the other hand, few related work [2], [5] present a resolution for the correlation conflict problem only. Other approaches [4], [8], [7], [9] propose using a high-level policy language to define and analyze firewall policies and then map this language to filtering rules. However, they do not target legacy firewall policies that already exist. Firewall query-based languages based on filtering rules are also proposed in [3], [6], [9]. Though useful to verify the firewall policy, only highly experienced users can practically use these techniques. So in general, we consider our work a new progress in this area because it offers new techniques to automate anomaly discovery, rule editing and translation, and can be applied on legacy filtering policies of low-level filtering rule representation.

This paper is organized as follows. In Section II, we formally define filtering rule relations, and we present our proposed model of rule relations and the policy tree representation. In Section III, we classify and define filtering policy anomalies, and then we describe the anomaly discovery algorithm. In Section IV, we present techniques for anomaly-free editing of filtering rules. In Section V, we present the filtering policy translator for textual translation of filtering policies. Finally, in Section VI, we conclude our work and provide directions for future work.

## II. FILTERING POLICY MODELLING

As a basic requirement for any filtering policy management solution, we first model the relations and the representation of filtering rules in the policy. This model is complete (i.e. includes all rules possible in any filtering policy) and efficient (i.e. easy to implement and easy to use). In this section, we formally describe our model of packet filtering rule relations.

*Packet filtering rule format:* a filtering rule is composed of filtering fields (also called network fields), an order and an action. It is possible to use any field in IP, UDP or TCP headers in the filtering rule, however, practical experience shows that the most commonly used network fields are: protocol type, source IP address, source port, destination IP address and destination port. The common format of packet filtering rules is as follows:

```
<order><protocol><s_ip><s_port><d_ip><d_port><action>
```

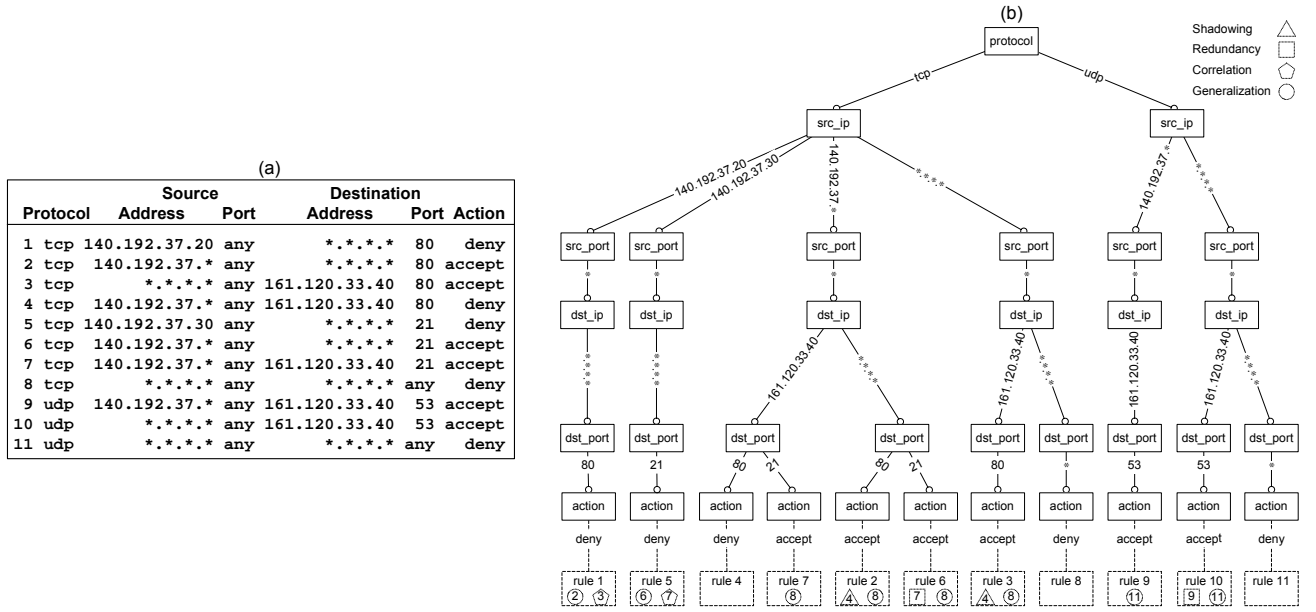


Fig. 1. Example: (a) filtering policy, (b) policy tree.

### A. Formalization of Filtering Rule Relations

To be able to build a useful model for filtering rules, we need to determine all the relations that may relate packet filters. In this section we define all the possible relations that may exist between any two filtering rules  $R_x$  and  $R_y$ .

*Def. 1:*  $R_x, R_y$  are completely disjoint, or  $R_x \mathcal{R}_{CD} R_y$ , iff

$$\forall i : R_x[i] \not\bowtie R_y[i]$$

where  $\bowtie \in \{<, >, =\}$ ,

$i \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$

*Def. 2:*  $R_x$  exactly matches  $R_y$ , or  $R_x \mathcal{R}_{EM} R_y$ , iff

$$\forall i : R_x[i] = R_y[i]$$

where  $i \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$

*Def. 3:*  $R_x$  inclusively matches  $R_y$ , or  $R_x \mathcal{R}_{IM} R_y$ , iff

$$\forall i : R_x[i] \subseteq R_y[i]$$

and  $\exists j$  such that  $:R_x[j] \neq R_y[j]$

where  $i, j \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$

$R_x$  is called the *subset match*, while  $R_y$  is the *superset match*.

*Def. 4:*  $R_x$  partially matches  $R_y$ , or  $R_x \mathcal{R}_{PM} R_y$ , iff

$\exists i, j$  such that  $R_x[i] \bowtie R_y[i]$  and  $R_x[j] \not\bowtie R_y[j]$

where  $\bowtie \in \{<, >, =\}$ ,

$i, j \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}, i \neq j$

*Def. 5:*  $R_x$  and  $R_y$  are correlated, or  $R_x \mathcal{R}_C R_y$ , iff

$$\forall i : R_x[i] \bowtie R_y[i] \text{ and}$$

$\exists i, j$  such that  $:R_x[i] \subset R_y[i]$  and  $R_x[j] \supset R_y[j]$

where  $\bowtie \in \{<, >, =\}$ ,

$i, j \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}, i \neq j$

### B. Filtering Policy Representation

We represent the filtering policy by a single rooted tree that we name the *policy tree*. The tree model provides a simple and apprehensible representation of the filtering rules and at the same time allows for easy discovery of relations and anomalies among the rules. Each node in a policy tree represents a field of the filtering rule, and each branch at this node represents a possible value of the associated field. The root node of the policy tree represents the protocol field, and the leaf nodes represent the action field, intermediate nodes represent other network fields in order. Every tree path starting at the root and ending at a leaf represents a rule in the policy and vice versa. Rules that have the same field value at a specific node, will share the same branch representing that value.

Fig. 1 illustrates the policy tree model for an example filtering policy. The dotted box below each leaf indicates the rule represented by the branch, in addition to other rules that are in anomaly with it as described in the following section.

## III. FILTERING POLICY ANOMALIES

In practical security policies, it is very common to have inter-related filtering rules. In this case, relative rule ordering must be carefully assigned. The ordering of rules is insignificant only if the rules are disjoint. A *filtering policy anomaly* is defined as the existence of two or more different filtering rules that match the same packet. In this section, we define different types of anomalies that may exist among filtering rules and then describe a technique to discover these anomalies.

### A. Filtering Policy Anomaly Classification

Here, we define a number of possible anomalies in filtering policies. These include errors for definite conflicts, or warnings for potential conflicts between filtering rules.

1) *Shadowing anomaly*: Rule  $R_y$  is shadowed by rule  $R_x$  if the following condition holds:

$$R_x[\text{order}] < R_y[\text{order}], R_x \mathfrak{R}_{EM} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

$$R_x[\text{order}] < R_y[\text{order}], R_x \mathfrak{R}_{IM} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

In Fig. 1, Rule 4 is shadowed by Rule 2. Shadowing is a critical error in the policy, as the shadowed rule never takes effect. This might cause an accepted traffic to be blocked or a denied traffic to be permitted. Therefore, as a general guideline, if there is an inclusive or exact match relationship between two rules, the subset (or specific) rule should precede the superset (or general) rule.

2) *Correlation anomaly*: Rule  $R_x$  and rule  $R_y$  have a correlation anomaly if the following condition holds:

$$R_x \mathfrak{R}_C R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

Referring to Fig. 1, Rule 1 is in correlation with Rule 3. The two rules with this ordering imply that all HTTP traffic that is coming from the address 140.192.37.20 and going to 161.120.33.40 is denied. However, if their order is reversed, the same traffic will be accepted. Correlation is considered an anomaly warning because the correlated rules imply an action that is not explicitly handled by the filtering rules.

3) *Generalization anomaly*: Rule  $R_y$  is a generalization of rule  $R_x$  if the following condition holds:

$$R_x[\text{order}] < R_y[\text{order}], R_y \mathfrak{R}_{IM} R_x, R_x[\text{action}] \neq R_y[\text{action}]$$

In Fig. 1, Rule 2 is a generalization of Rule 1. This implies that all HTTP traffic that is coming from the address 140.192.37.\* will be accepted, except the traffic that comes from 140.192.37.20. Generalization is often used to exclude a specific part of the traffic from a general filtering action. It is considered only an anomaly warning because the specific rule makes an exception of the general rule.

4) *Redundancy anomaly*: Rule  $R_y$  is redundant to rule  $R_x$  if one of the following conditions holds:

$$R_x[\text{order}] < R_y[\text{order}], R_x \mathfrak{R}_{EM} R_y, R_x[\text{action}] = R_y[\text{action}]$$

$$R_x[\text{order}] < R_y[\text{order}], R_x \mathfrak{R}_{IM} R_y, R_x[\text{action}] = R_y[\text{action}]$$

Whereas rule  $R_x$  is redundant to rule  $R_y$  if:

$$R_x[\text{order}] < R_y[\text{order}], R_y \mathfrak{R}_{IM} R_x, R_x[\text{action}] = R_y[\text{action}]$$

provided that  $R_x$  is not involved in any generalization or correlation anomalies with other rules in between  $R_x$  and  $R_y$ .

Referring to Fig. 1, Rule 7 is redundant to Rule 6, and Rule 9 is redundant to Rule 10. Redundancy is considered an error in the filtering policy because the redundant rule adds to the size of the filtering rule table, and might increase the search time and space requirements. In general, to avoid redundant rules, a superset rule following a subset rule should have an opposite filtering action.

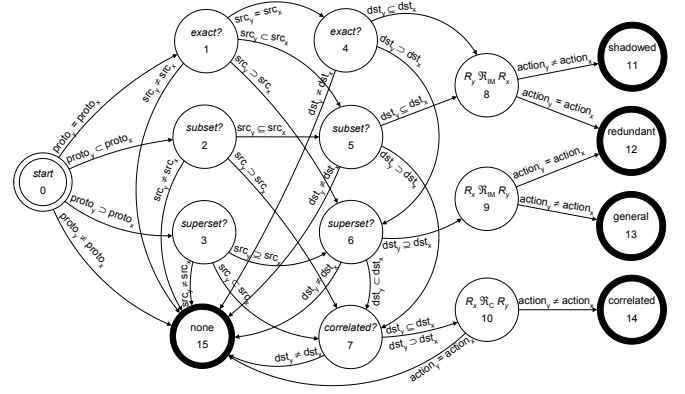


Fig. 2. State diagram for anomaly discovery among rules  $R_x$  and  $R_y$ , where rule  $R_x$  comes after  $R_y$ .

## B. Anomaly Discovery Algorithm

The state diagram shown in Fig. 2 illustrates the anomaly discovery states for any two rules,  $R_x$  and  $R_y$ , where  $R_y$  comes after  $R_x$  in the order.

Initially no relationship is assumed. Each field in  $R_y$  is compared to the corresponding field in  $R_x$ . The relationship between the two rules is determined based on the result of subsequent comparisons. If every field of  $R_y$  is a subset or equal to the corresponding field in  $R_x$  and both rules have the same action,  $R_y$  is redundant to  $R_x$ , while if the actions are different,  $R_y$  is shadowed by  $R_x$ . If every field of  $R_y$  is a superset or equal to the corresponding field in  $R_x$  and both rules have the same action,  $R_x$  is potentially redundant to  $R_y$ , while if the actions are different,  $R_y$  is a generalization of  $R_x$ . If some fields of  $R_x$  are subsets or equal to the corresponding fields in  $R_y$ , and some fields of  $R_x$  are supersets to the corresponding fields in  $R_y$ , and their actions are different, then  $R_x$  is in correlation with  $R_y$ . If none of the preceding cases occur, then the two rules do not involve any anomalies.

The basic idea for discovering anomalies is to determine if any two rules coincide in the same policy tree path. Two rules coincide in a policy tree path if they are exactly matching, inclusively matching or correlated. When a rule is inserted in the policy tree, each field in the rule is compared with existing branches at the tree node resembling this field. At a given node, if the field is subset or equal to a branch, the rule is inserted in this branch. If the field is superset of a branch, the rule is inserted in the subset branch, and a new branch is created at this node. If the rule did not match any branches, a new branch is created at this node. If the rule coincides with another rule on any of these paths, there is a potential anomaly that can be determined based on the anomaly definitions in Section III. The algorithms for building the policy tree and discovering policy anomalies are fully described in [1].

Applying the algorithm on the rules in Fig. 1, the discovered anomalies are marked in the dotted boxes at the bottom of the policy tree. Shadowed rules are marked with a triangle, redundant rules with a square, correlated rules with a pentagon and generalization rules with a circle.

## IV. FILTERING POLICY EDITOR

Filtering policies are often written by different network administrators and occasionally updated (by inserting, modifying or removing rules) to accommodate new security requirements and network topology changes. In this section, we present policy editing techniques that significantly simplify rule editing, and avoid introducing anomalies due to policy updates.

### A. Rule Insertion

Since the ordering of rules in the filtering rule list directly impacts the semantics of the filtering security policy, a new rule must be inserted in the proper order in the policy such that no shadowing, correlation or redundancy is created. The policy editor helps the user to determine the correct position of the new rule to be inserted. It also identifies anomalies that may occur due to improper insertion of the new rule.

To insert a new rule, the order of this rule in the local policy is determined based on its relation with other existing rules. In general, a new rule should be inserted before any rule that is a superset match, and after any rule that is a subset match of this rule. The local policy tree is used to keep track of the correct ordering of the new rule, and discover any potential anomalies. We start by searching for the correct rule position in the policy tree by comparing the fields of the new rule with the corresponding tree branch values. If the field value is a subset of the branch, then the order of the new rule so far is smaller than the minimum order of all the rules in this branch. If the field value is a superset of the branch, the order of the new rule so far is greater than the maximum order of all the rules in this branch. On the other hand, if the rule is disjoint, then it can be given any order in the policy. Similarly, the tree browsing continues evaluating the next fields in the rule recursively as long as the field value is an exact match or a subset match of the branch. When the action field is reached, the rule is inserted and assigned the order determined during tree browsing. A new branch is created for the new rule any time a disjoint or superset match is found. If the new rule is redundant because it is an exact match or a subset match and it has the same action of an existing rule, the policy editor rejects it and prompts the user with an appropriate message.

### B. Rule Removal and Modification

In general, removing a rule has much less impact on the filtering policy than insertion. A removed rule does not introduce an anomaly but it might change the policy semantics and this change should be highlighted and confirmed. To remove a rule, the user enters the rule number to retrieve the rule from the rule list and selects to remove it. To preview the effect of rule removal, the policy editor gives a textual translation of the affected portion of the policy before and after the rule is removed. The user is able to compare and inspect the policy semantics before and after removal, and re-assure correctness of the policy changes.

Modifying a rule in a filtering policy is also a critical operation. However, this editing action can be easily managed as rule removal and insertion as described before.

## V. FILTERING POLICY TRANSLATOR

It is difficult to comprehend the policy semantics by reading through a set of filtering rules. Rules that have common or related fields values such as source IP address or destination port number may be widely scattered within the rule table, making it even harder to have a concise and complete description of the filtering policy. In this section, we implement a filtering policy translation tool that describes, in a natural textual language, the meaning and the interactions of all filtering rules in the policy, revealing the complete semantics of the policy in a very concise fashion.

### A. Maximum Rule Aggregation for Tree-based Translation

To achieve a concise translation, we model the rules such that partially matched rules are aggregated together based on related field values. Let us start by translating Rules 2 and 5 in Fig. 3 separately, we get the following translation: “*accept tcp traffic from address 140.192.37.\* and to port 80*” for Rule 2 and “*accept tcp traffic from address 140.192.37.\* and to port 21*” for Rule 5. Both rules have common protocol, source address, source port and destination address but different destination ports. The policy tree aggregates the two rules in one branch starting from the protocol node down to the destination address node, and then the two rules split in two branches at the destination port node. By using depth-first traversal to translate the branch that aggregates the two rules, we get the following concise translation that exploits the commonality of the two rules: “*accept tcp traffic from address 140.192.37.\* and to port 80 or port 21.*”

However, the described technique provides concise rule translation only if the fields of common values in the policy tree are followed by the fields of different values. Otherwise, if the fields of common values come after the fields of different values, depth-first traversal does not generate a concise translation, as separate tree branches will be created at a higher tree level. For example, Rules 8 and 9 have different source addresses but they have a common destination port number. Since the source address field is inserted in the policy tree before the destination port field, each rule will have a separate branch starting at the source address node. If we attempt to translate these branches, we will get a separate translation statement for each rule, although these two rules have a common field value. From the previous discussion, we realize that a fixed-order policy tree may not give us a concise translation for all the rules in the policy. Therefore, in order to have the most concise translation, we construct a more optimized tree that provides maximum aggregation of all the rules regardless of the order of common fields, and at the same time preserves the semantics of the security policy. We call this optimized tree the *translation tree*.

### B. Translation Tree Construction

The idea of the translation tree is basically to find the best field ordering that provides maximum aggregation of a set of related rules. This is accomplished by building five policy trees, each rooted with one of the filter fields. Each branch

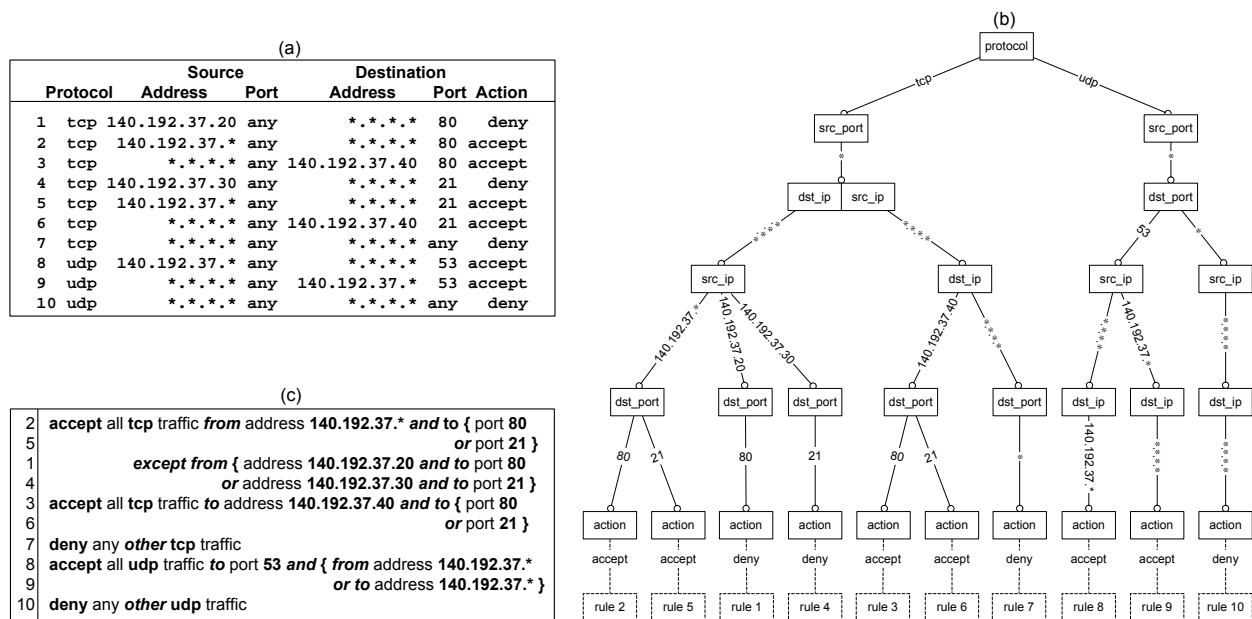


Fig. 3. Example: (a) filtering rules, (b) translation tree, (c) policy translation.

at the root node of every policy tree is an aggregation of a number of rules. The branch that has the highest number of aggregated rules is picked from its policy tree and inserted as a new branch in the translation tree, and all the aggregated rules in this branch are removed from all the policy trees. If there are any remaining rules in the policy trees, another branch is picked in the same manner and inserted in the translation tree until all rules are completely covered by the translation tree branches. This process is recursively applied at all levels of all the translation tree branches. Eventually, a single rooted tree is created such that each branch at the root node represents a field value and aggregates the maximum number of rules that have this field value in common. Finally, the rules are translated by running a depth-first tree traversal algorithm on the translation tree. Fig. 3 illustrates the translation tree and the resulting translation text of an example filtering policy.

## VI. CONCLUSIONS AND FUTURE WORK

Firewall security, like any other technology, requires proper management to provide proper security service. Thus, just having a firewall on the boundary of a network may not necessarily make the network any secure. One reason of this is the complexity of managing firewalls rules and the potential network vulnerability due to rule conflicts. The work presented in this paper provides a number of techniques for purifying and protecting the filtering policy from anomalies. These techniques can be used to manage a general filtering security policy without prior analysis of filtering rules. In this work, we formally defined all possible filtering rule relations and we used this to classify filtering policy anomalies. We then model the filtering rule information and relations in a tree-based representation. Based on this model and formalization, we developed a set of tools to manage filtering policies, namely: a Policy Anomaly Detector that discovers conflicting rules, a

Policy Editor that automatically determines the proper order for any inserted or modified rule, and a Policy Translator that provides a concise natural language translation of low-level rules in a filtering policy.

We believe that there is more to do in the area of security policy management. Our future research plan includes extending the proposed techniques to handle distributed filtering policies, extending the translator with a query-based interface, and enhancing the visualization of filtering rules.

## REFERENCES

- [1] E. Al-Shaer and H. Hamed. "Design and Implementation of Firewall Policy Advisor Tools." *Technical Report CTI-techrep0801*, School of Computer Science Telecommunications and Information Systems, DePaul University, August 2002.
- [2] D. Eppstein and S. Muthukrishnan. "Internet Packet Filter Management and Rectangle Geometry." *Proceedings of the 12<sup>th</sup> Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2001.
- [3] P. Eronen and J. Zitting. "An Expert System for Analyzing Firewall Rules." *Proceedings of the 6<sup>th</sup> Nordic Workshop on Secure IT-Systems (NordSec 2001)*, November 2001.
- [4] Z. Fu, F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine and C. Xu. "IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution." *Proceedings of Policy'2001 Workshop*, January 2001.
- [5] B. Hari, S. Suri and G. Parulkar. "Detecting and Resolving Packet Filter Conflicts." *Proceedings of IEEE INFOCOM'2000*, March 2000.
- [6] S. Hazelhurst. "Algorithms for Analyzing Firewall and Router Access Lists." *Technical Report TR-WitsCS-1999*, Department of Computer Science, University of the Witwatersrand, South Africa, July 1999.
- [7] E. Lupu and M. Sloman. "Model-Based Tool Assistance for Packet-Filter Design." *Proceedings of Workshop on Policies for Distributed Systems and Networks (POLICY'2001)*, January 2001.
- [8] I. Lck. C. Schfer and H. Krumm. "Conflict Analysis for Management Policies." *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM'1997)*, May 1997.
- [9] A. Mayer, A. Wool and E. Ziskind. "Fang: A Firewall Analysis Engine." *Proceedings of IEEE Symposium on Security and Privacy*, May 2000.
- [10] L. Qiu, G. Varghese, and S. Suri. "Fast Firewall Implementations for Software and Hardware-based Routers." *Proceedings of the 9<sup>th</sup> International Conference on Network Protocols (ICNP'2001)*, November 2001.