FIREWALL POLICY ADVISOR FOR ANOMALY DISCOVERY AND RULE EDITING

Ehab S. Al-Shaer and Hazem H. Hamed
Multimedia Networking Research Laboratory
School of Computer Science, Telecommunications and Information Systems
DePaul University, Chicago, USA
{ehab,hhamed}@cs.depaul.edu

Abstract: Firewalls are core elements in network security. However, managing firewall rules, especially for enterprize networks, has become complex and error-prone. Firewall filtering rules have to be carefully written and organized in order to correctly implement the security policy. In addition, inserting or modifying a filtering rule requires thorough analysis of the relationship between this rule and other rules in order to determine the proper order of this rule and commit the updates. In this paper, we present a set of techniques and algorithms that provide (1) automatic discovery of firewall policy anomalies to reveal rule conflicts and potential problems in legacy firewalls, and (2) anomaly-free policy editing for rule insertion, removal and modification. This is implemented in a user-friendly tool called "Firewall Policy Advisor." The Firewall Policy Advisor significantly simplifies the management of any generic firewall policy written as filtering rules, while minimizing network vulnerability due to firewall rule misconfiguration.

Keywords: Firewall, security management, security policy, policy conflict.

1. Introduction

With the global Internet connection, network security has gained significant attention in the research and industrial communities. Due to the increasing threat of network attacks, firewalls have become important integrated elements not only in enterprize networks but also in small-size and home networks. Firewalls have been the frontier defense for secure networks against attacks and unauthorized traffic by filtering out unwanted network traffic coming into or going from the secured network. The filtering decision is taken according to a set of ordered filtering rules defined based on predefined security policy requirements [4].

Although deployment of firewall technology is an important step toward securing our networks, the complexity of managing firewall policy might limit the effectiveness of firewall security. A firewall policy may include *anomalies*, where a packet may match with two or more different filtering rules. 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 such as one general rule shadowing another specific rule, or correlated rules whose relative ordering determines different actions for the same packet. In addition, a typical large-scale enterprize network might involve hundreds of rules

that might be written by different administrators in various times. This significantly increases the potential of anomaly occurrence in the firewall policy, jeopardizing the security of the protected network.

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 firewall 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 may exist among the filtering rules. We finally develop an anomaly-free firewall rule editor, which greatly simplifies adding, removing and modifying rules into firewall policy. We used the Java programming language to implement these algorithms in one graphical user-interface tool called the "Firewall Policy Advisor."

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, 8, 10, 11, 17]. On the other hand, few related work [6, 10] present a resolution for the correlation conflict problem only. Other approaches [2, 9, 12, 14, 18] propose using a high-level policy language to define and analyze firewall policies and then map this language to filtering rules. Firewall query-based languages based on filtering rules are also proposed in [7, 11]. So in general, we consider our work a new progress in this area because it offers new techniques for complete anomaly discovery and rule editing that can be applied on legacy firewall policies of low-level filtering rule representation.

This paper is organized as follows. In Section 2, we give an introduction to firewall operation and filtering rule format. In Section 3, we formally define filtering rule relations, and we present our proposed model of filtering rule relations and the policy tree representation. In Section 4, we classify and define firewall policy anomalies, and then we describe the anomaly discovery algorithm and implementation. In Section 5, we present the design and implementation of anomaly-free firewall rule editor. In Section 6, we give a summary of related work. Finally, in Section 7, we show our conclusions and our future work plan.

2. Firewall Background

A firewall is a network element that controls the traversal of packets across the boundaries of a secured network based on a specific security policy. A firewall security policy is a list of ordered filtering rules that define the actions performed on matching packets. A rule is composed of filtering fields (also called network fields) such as protocol type, source IP address, destination IP address, source port and destination port, and an action field. Each network field could be a single value or range of values. Filtering actions are either to *accept*, which passes the packet into or from the secure network, or to *deny*, which causes the packet to be discarded. The packet is accepted or denied by a specific rule if the packet header information matches all the network fields of this rule. Otherwise, the following rule is examined and the process is repeated until a matching rule is found or the default policy action is performed [3]. In this paper, we assume a "deny" default policy action.

order	protocol	src_ip	src_port	dst_ip	dst_port	action
1:	tcp,	140.192.37.20,	any,	*.*.*.*,	80,	deny
2:	tcp,	140.192.37.*,	any,	*.*.*.*,	80,	accept
3:	tcp,	*.*.*.*,	any,	161.120.33.40,	80,	accept
4:	tcp,	140.192.37.*,	any,	161.120.33.40,	80,	deny
5:	tcp,	140.192.37.30,	any,	*.*.*.*,	21,	deny
6:	tcp,	140.192.37.*,	any,	*.*.*.*,	21,	accept
7:	tcp,	140.192.37.*,	any,	161.120.33.40,	21,	accept
8:	tcp,	*.*.*.*,	any,	*.*.*.*,	any,	deny
9:	udp,	140.192.37.*,	any,	161.120.33.40,	53,	accept
10:	udp,	*.*.*.*,	any,	161.120.33.40,	53,	accept
11:	udp,	*.*.*.*,	any,	*.*.*.*,	any,	deny

Figure 1. A firewall policy example.

Filtering Rule Format It is possible to use any field in IP, UDP or TCP headers in the rule filtering part, however, practical experience shows that the most commonly used matching fields are: protocol type, source IP address, source port, destination IP address and destination port. Some other fields, like TTL and TCP flags, are occasionally used for specific filtering purposes [5]. The following is the common format of packet filtering rules in a firewall policy:

```
<order> <protocol><src_ip><src_port><dst_ip><dst_port> <action>
```

In this paper, we refer to the network fields as the "5-tuple filter." The order of the rule determines its position relative to other filtering rules. IP addresses can be a host (e.g. 140.192.37.120), or a network address (e.g. 140.192.37.*). Ports can be either a single specific port number, or any port number indicated by "any." Some firewall implementations allow the usage of non-wildcard ranges in specifying source and destination addresses or ports. However, it is always possible to split a filtering rule with a multi-value field into several rules each with a single-value field [15]. An example of typical firewall rules is shown in Figure 1.

3. Firewall Policy Modelling

As a basic requirement for any firewall policy management solution, we first model the relations between the rules in a firewall policy. Rule relation modelling is necessary for analyzing the firewall policy and designing management techniques such as anomaly discovery and policy editing. In this section, we formally describe our model of firewall rule relations.

3.1 Formalization of Firewall Rule Relations

To be able to build a useful model for filtering rules, we need to determine all the relations that may relate two or more packet filters. In this section we define all the possible relations that may exist between filtering rules, and we show that there is no other relation exists. We determine the relations based on comparing the network fields of filtering rules as follows.

DEFINITION 1 *Rules R_x and R_y are completely disjoint if every field in R_x is not a subset and not a superset and not equal to the corresponding field in R_y.*

Formally, R_x and R_y are completely disjoint iff
$\forall i : R_x[i] \not\bowtie R_y[i]$
where $\bowtie \in \{\subset, \supset, =\}, i \in \{\texttt{protocol}, \texttt{src_ip}, \texttt{src_port}, \texttt{dst_ip}, \texttt{dst_port}\}$

DEFINITION 2 *Rules R_x and R_y are exactly matched if every field in R_x is equal to the corresponding field in R_y.*

Formally, R_x exactly matches R_y iff
$\forall i : R_x[i] = R_y[i]$ where $i \in \{\texttt{protocol}, \texttt{src_ip}, \texttt{src_port}, \texttt{dst_ip}, \texttt{dst_port}\}$

DEFINITION 3 *Rules R_x and R_y are inclusively matched if they do not exactly match and if every field in R_x is a subset or equal to the corresponding field in R_y. R_x is called the subset match while R_y is called the superset match.*

Formally, R_x inclusively matches 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 $i, j \in \{\texttt{protocol}, \texttt{src_ip}, \texttt{src_port}, \texttt{dst_ip}, \texttt{dst_port}\}$

For example, rule 1 inclusively matches rule 2 in Figure 1. Rule 1 is the subset match of the relation while rule 2 is the superset match.

DEFINITION 4 *Rules R_x and R_y are partially disjoint (or partially matched) if there is at least one field in R_x that is a subset or a superset or equal to the corresponding field in R_y, and there is at least one field in R_x that is not a subset and not a superset and not equal to the corresponding field in R_y.*

Formally, R_x and R_y are partially disjoint (or partially matched) iff
$\exists i, j$ such that $R_x[i] \bowtie R_y[i]$ and $R_x[j] \not\bowtie R_y[j]$ and $i \neq j$
where $\bowtie \in \{\subset, \supset, =\}, i, j \in \{\texttt{protocol}, \texttt{src_ip}, \texttt{src_port}, \texttt{dst_ip}, \texttt{dst_port}\}$

For example, rule 2 and rule 6 in Figure 1 are partially disjoint (or partially matched).

DEFINITION 5 *Rules R_x and R_y are correlated if some fields in R_x are subsets or equal to the corresponding fields in R_y, and the rest of the fields in R_x are supersets of the corresponding fields in R_y.*

Formally, R_x and R_y are correlated iff
$\forall i : R_x[i] \bowtie R_y[i]$ and
$\exists i, j$ such that : $R_x[i] \subset R_y[i]$ and $R_x[j] \supset R_y[j]$ and $i \neq j$
where $\bowtie \in \{\subset, \supset, =\}, i, j \in \{\texttt{protocol}, \texttt{src_ip}, \texttt{src_port}, \texttt{dst_ip}, \texttt{dst_port}\}$

For example, rule 1 and rule 3 in Figure 1 are correlated.

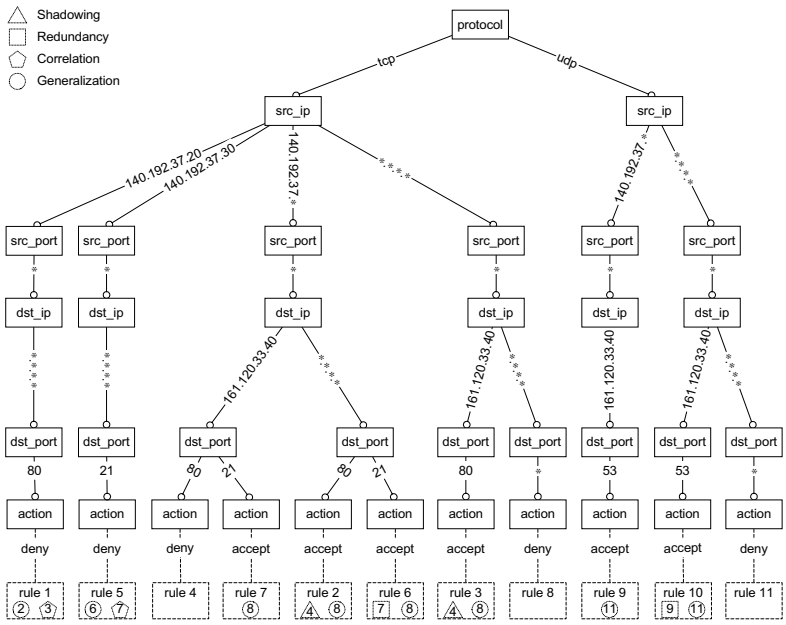

Figure 2. The policy tree for the firewall policy in Figure 1.

The following theorems show that these relations are distinct, i.e. only one relation can relate R_x and R_y, and complete, i.e. there is no other relation between R_x and R_y could exist. A complete proof of the theorems is presented in [1].

THEOREM 1 *The relations defined above are distinct; i.e. any two k-tuple filters in a firewall policy are related by only one of the defined relations.*

THEOREM 2 *The union of these relations represents the universal set of relations between any two k-tuple filters in a firewall policy.*

3.2 Firewall Policy Representation

We represent the firewall 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 5-tuple filter 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.

Figure 2 illustrates the policy tree model of the security policy in Figure 1. Notice that every rule should have an action leaf in the tree. The dotted box below each leaf indicates the rule represented by that branch in addition to other rules that are in anomaly with it as described in the following section. The tree shows that rules 1 and 5 each has a separate source address branch as they have different field values, whereas rules 2, 4, 6 and 7 share the same source address branch as they all have the same field value. Also notice that rule 8 has a separate branch and also appears on other rule branches of which it is a superset, while rule 4 has a separate branch and also appears on other rule branches of which it is a subset.

The basic idea for building the policy tree is to insert the filtering rule in the correct tree path. When a rule field is inserted at any tree node, the rule branch is determined based on matching the field value with the existing branches. If a branch exactly matches the field value, the rule is inserted in this branch, otherwise a new branch is created. The rule also propagates in superset or superset branches to preserve the relations between the policy rules.

4. Firewall Policy Anomaly Discovery

The ordering of filtering rules in a firewall policy is very crucial in determining the security policy because the firewall packet filtering process is performed by sequentially matching the packet against filtering rules until a match is found. If filtering rules are completely disjoint, the ordering of the rules is insignificant. However, it is very common to have filtering rules that are inter-related. In this case, if the relative rule ordering is not carefully assigned, some rules may be always screened by other rules producing an incorrect security policy. Moreover, when a security policy contains a large number of filtering rules, the possibility of writing conflicting or redundant rules is relatively high. A firewall policy anomaly is defined as the existence of two or more different filtering rules that match the same packet. In this section, we classify different anomalies that may exist among filtering rules and then describe a technique for discovering these anomalies.

4.1 Firewall Policy Anomaly Classification

Here, we describe and then define a number of possible firewall policy anomalies. These include errors for definite conflicts that cause some rules to be always suppressed by other rules, or warnings for potential conflicts that may be implied in related rules.

1. **Shadowing anomaly** A rule is shadowed when a previous rule matches all the packets that match this rule, such that the shadowed rule will never be activated. Rule R_y is shadowed by rule R_x if R_y follows R_x in the order, and R_y is a subset match of R_x, and the actions of R_x and R_y are different. As illustrated in the rules in Figure 1, rule 4 is a subset match of rule 3 with a different action. We say that rule 4 is shadowed by rule 3 as rule 4 will never get activated.

   Shadowing is a critical error in the policy, as the shadowed rule never takes effect. This might cause a permitted traffic to be blocked and vice versa. It is important to discover shadowed rules and alert the administrator who might correct this error by reordering or removing the shadowed rule.

2. **Correlation anomaly** Two rules are correlated if the first rule in order matches some packets that match the second rule and the second rule matches some packets that match the first rule. Rule R_x and rule R_y have a correlation anomaly if R_x and R_y are correlated, and the actions of R_x and R_y are different. As illustrated in the rules in Figure 1, rule 1 is in correlation with rule 3; if the order of the two rules is reversed, the effect of the resulting policy will be different.

   Correlation is considered an anomaly warning because the correlated rules imply an action that is not explicitly handled by the filtering rules. Consider rules 1 and 3 in Figure 1. The two rules with this ordering imply that all HTTP traffic coming from address 140.192.37.20 and going to address 161.120.33.40 is denied. However, if their order is reversed, the same traffic will be accepted. Therefore, in order to resolve this conflict, we point out the correlation between the rules and prompt the user to choose the proper order that complies with the security policy requirements.

3. **Generalization anomaly** A rule is a generalization of another rule if this general rule can match all the packets that match a specific rule that precedes it. Rule R_y is a generalization of rule R_x if R_y follows R_x in the order, and R_y is a superset match of R_x, and the actions of R_y and R_x are different. As illustrated in the rules in Figure 1, rule 2 is a generalization of rule 1; if the order of the two rules is reversed, the effect of the resulting policy will be changed, and rule 1 will not be effective anymore, as it will be shadowed by rule 2. Therefore, as a general guideline, if there is an inclusive match relationship between two rules, the superset (or general) rule should come after the subset (or specific) rule.

   Generalization is considered only an anomaly warning because the specific rule makes an exception of the general rule, and thus it is important to highlight its action to the administrator for confirmation.

4. **Redundancy anomaly** A redundant rule performs the same action on the same packets as another rule such that if the redundant rule is removed, the security policy will not be affected. Rule R_y is redundant to rule R_x if R_x precedes R_y in the order, and R_y is a subset or exact match of R_x, and the actions of R_x and R_y are similar. If R_x precedes R_y in the order, and R_x is a subset match of R_y, and the actions of R_x and R_y are similar, then Rule R_x is redundant to rule R_y provided that R_x is not involved in any generalization or correlation anomalies with other rules preceding R_y. As illustrated in the rules in Figure 1, rule 7 is redundant to rule 6, and rule 9 is redundant to rule 10, so if rule 7 and rule 9 are removed, the effect of the resulting policy will not be changed.

   Redundancy is considered an error. A redundant rule may not contribute in making the filtering decision, however, it adds to the size of the filtering rule table, and might increase the search time and space requirements. It is important to discover redundant rules so that the administrator may modify its filtering action or remove it altogether.

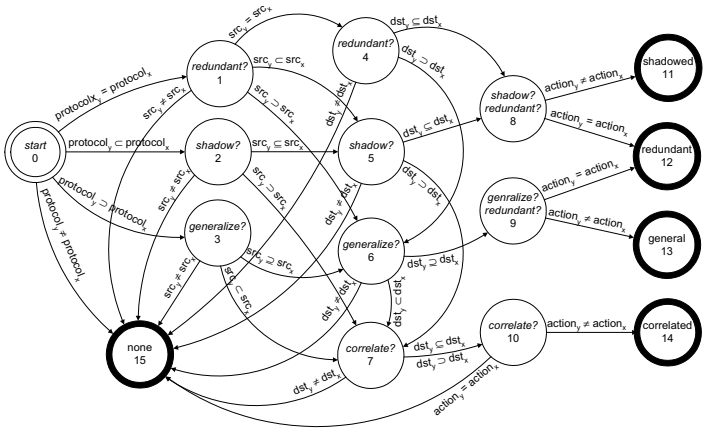

Figure 3. State diagram for detecting anomalies for rules R_x and R_y, R_y comes after R_x.

4.2 Anomaly Discovery Algorithm

The state diagram shown in Figure 3 summarizes anomaly discovery for any two rules, R_x and R_y where R_y comes after R_x in the order. For simplicity, the source address and source port and integrated into one field, and the same with the destination address and port. This simplification reduces the number of states and simplifies the explanation of the diagram. A similar state diagram can be produced for the real case of five fields with a substantially larger number of states involved.

Initially no relationship is assumed. Each field in R_y is compared to the corresponding field in R_x starting with the protocol then source address and port, and finally destination address and port. 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, the two rules do not involve any anomalies.

The basic idea for discovering anomalies is by determining if two rules coincide in their policy tree paths. If the tree path of a rule coincides with the tree path of another rule, there is a potential anomaly that can be determined based on the previous definitions of anomalies. If rule paths do not coincide, these rules are disjoint and they have no anomalies. The algorithm for building the policy tree and determining the anomalies among the filtering rules is shown in Figures 4 and 5. The algorithm is divided into two main parts: an anomaly discovery routine, `DiscoverAnomaly`, which represents the transition states in the state diagram, and an anomaly decision routine, `DecideAnomaly`, which represents the termination states.

```
function DiscoverAnomaly(rule, field, node, anomaly_state)
  if field = ACTION then
    value_found = FALSE
    for each branch in node.branch_list do
      if branch.value = rule.field.value then
        value_found = TRUE
        if anomaly_state = NOANOMALY then anomaly_state = REDUNDANT
        DiscoverAnomaly(rule, field.next, branch.node, anomaly_state)
      else if rule.field.value is superset of branch.value then
        if anomaly_state = GENERALIZATION then
          DiscoverAnomaly(rule, field.next, branch.node, CORRELATION)
        else
          DiscoverAnomaly(rule, field.next, branch.node, SHADOWING)
      else if rule.field.value is subset of branch.value then
        if anomaly_state = SHADOWING then
          DiscoverAnomaly(rule, field.next, branch.node, CORRELATION)
        else
          DiscoverAnomaly(rule, field.next, branch.node, GENERALIZATION)
      end if
    end for
    if value_found = FALSE then
      new_branch = new TreeBranch(rule, rule.field, rule.field.value)
      node.branch_list.add(new_branch)
      DiscoverAnomaly(rule, field.next, new_branch.node, NOANOMALY)
    end if
  else /* action field reached */
    call DecideAnomaly(rule, field, node, anomaly_state)
  end if
end function
```

Figure 4. Algorithm for building the policy tree with anomaly discovery.

In the discovery routine, the previous anomaly state is checked if there is a value match between the field of the new rule and the already existing field branch. The next anomaly state is determined based on the shown state diagram and the algorithm is executed recursively to let the rule propagate in existing branches and check the remaining fields. As the rule propagates, the anomaly state is updated until the final state is reached. If there is no exact match for the value of a field, a new branch is created at the current node to represent the inserted field value, and the anomaly state is initialized to no anomaly. The decision routine is activated once all the rule fields have been inserted in the tree and the action field is reached. If the rule action coincides with the action of another rule, an anomaly is discovered. At that point the final anomaly state is determined and reported. If an anomaly is discovered and decided, the user is reported with the type of anomaly and the rules involved.

Applying the algorithm on the rules in Figure 1, the discovered anomalies are marked in the dotted boxes at the bottom of the policy tree in Figure 2. Shadowed rules are marked with a triangle, redundant rules with a square, correlated rules with a pentagon and generalization rules with a circle.

Figure 6 shows the graphical user interface for the Firewall Policy Advisor. The bottom panel shows a tabular list of filtering rules. The top-left panel displays the policy tree showing aggregated rules. The top-right panel displays the anomalies discovered along with highlighting redundant and shadowed rules in a different color.

```
function DecideAnomaly(rule, field, node, anomaly)
  if node has branch_list then
    branch = node.branch_list.first()
    if anomaly = CORRELATION then
      if not rule.action = branch.value then
        branch.rule.anomaly = CORRELATION
        report rule rule.id is in correlation with rule branch.rule.id
      else anomaly = NONE
    else if anomaly = GENERALIZATION and not rule.action = branch.value then
      branch.rule.anomaly = SPECIALIZATION
      report rule rule.id is a generalization of rule branch.rule.id
    else if anomaly = GENERALIZATION and rule.action = branch.value then
      if branch.rule.anomaly = NONE then
        anomaly = NONE; branch.rule.anomaly = REDUNDANCY
        report rule branch.rule.id is redundant to rule rule.id
      end if
    else if rule.action = branch.value then
      anomaly = REDUNDANCY
      report rule rule.id is redundant to rule branch.rule.id
    else if not rule.action = branch.value then
      anomaly = SHADOWING
      report rule rule.id is shadowed by rule branch.rule.id
    end if
  end if
  rule.anomaly = anomaly
end function
```

Figure 5. Algorithm for making the anomaly decision.

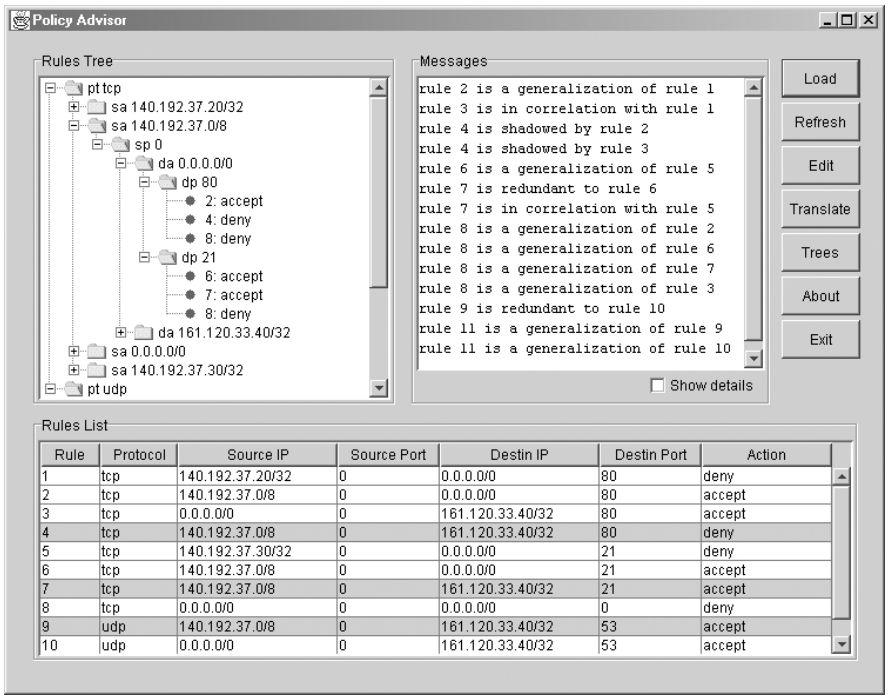

Figure 6. Policy Advisor anomaly discovery user interface.

5. Firewall Policy Editor

Firewall 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. Editing a security policy can be far more difficult than creating a new one. As rules in firewall policy are ordered, a new rule must be inserted in a particular order to avoid creating anomalies. The same applies if any network field in a rule is modified. In this section, we present a policy editor tool that simplifies the rule editing task significantly, and avoids introducing anomalies due to policy updates. The policy editor (1) prompts the user with the proper position(s) for a new or modified rule, (2) shows the changes in the security policy semantic before and after removing a rule, and (3) provides visual aids for users to track and verify policy changes. Using the policy editor, administrators require no prior knowledge or understating of the firewall policy in order to insert, modify or remove a rule.

5.1 Rule Insertion

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

The general idea is that the order of a new rule is determined based on its relation with other existing rules in the firewall policy. In general, a new rule should be inserted before any rule that is its superset match, and after any rule that is its subset match. The policy tree is used to keep track of the correct order of the new rule, and discover any potential anomalies. The algorithm implementing the mechanism to insert a new rule is fully described in [1].

The algorithm is organized into two phases: the browsing phase and the insertion phase. In the browsing phase, the fields of the new rule are compared with the corresponding tree branch values one at a time. If the field value of the new rule is a subset of an existing branch, then the new rule must be inserted before the minimum order of all the rules in this branch. If the field value is a superset of an existing branch, the rule must be inserted after the maximum order of all the rules in this branch. In addition, if the field value is an exact match or a subset match of a branch, evaluating the next field continues recursively by browsing through the branch sub-tree until correct position of the rule within the sub-tree is determined. Otherwise, if disjoint or superset match occurs, a branch is created for the new rule.

The algorithm enters into the insertion phase when the action field of a new rule is to be inserted. If an action branch is created for the new rule, then the rule will be inserted and assigned the order determined in the browsing phase. If there is more than one possible order for this rule, the user is asked to select an order from within a valid range of orders as determined in the browsing phase. However, if the order state of the new rule remains undetermined then policy editor rejects this new rule and prompts the user with the appropriate message. If the rule is inserted, the anomaly discovery algorithm is invoked to alert the administrators with any generalization or correlation cases as a possible source of anomalies in the firewall policy.

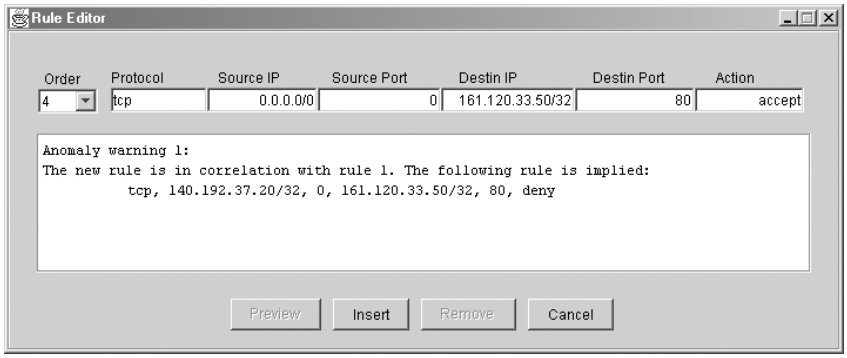

Figure 7. Rule editor user interface.

5.2 Rule Removal and Modification

In general, removing a rule has much less impact on the firewall 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 firewall policy is also a critical operation. However, this editing action can be easily managed as rule removal and insertion as described before.

Figure 7 shows the graphical user interface for the rule editor tool. The figure shows the final step in inserting a rule in the filtering rule table. The tool alerts the user for any anomalies that may be introduced by inserting the new rule.

6. Related Work

A significant amount of work has been reported in the area of firewall and policy-based security management. In this section, we focus our study on related work that intersects with our work in three areas: packet filter modelling, conflict discovery and rule analysis.

Several models have been proposed for filtering rules. Ordered binary decision diagram is used as a model for optimizing packet classification in [11]. Another model using tuple space is developed in [16], which combines a set of filters in one tuple and stored in a hash table. The model in [17] uses bucket filters indexed by search trees. Multi-dimensional binary tries are also used to model filters [15]. In [6] a geometric model is used to represent 2-tuple filtering rules. Because these models were designed particularly to optimize packet classification in high-speed networks, we found them too complex to use for firewall policy analysis. We can confirm from experience that the tree-based model is simple and powerful enough for this purpose.

Research in policy conflict analysis has been actively growing for many years. However, most of the work in this area addresses general management policies rather than firewall-specific policies. For example, authors in [13] classify possible policy conflicts in role-based management frameworks, and develop techniques to discover them. A policy conflict scheme for IPSec is presented in [8]. Although this work is very useful as a general background, it is not directly applicable in firewall anomaly discovery. On the other hand, few research projects address the conflict problem in filtering rules. Both [6] and [10] provide algorithms for detecting and resolving conflicts among general packet filters. However, they only detect what we defined as correlation anomaly because it causes ambiguity in packet classifiers. In conclusion, we could not find any published research work that uses low-level filtering rules to perform a complete anomaly analysis and guided editing of firewall policies.

7. Conclusions and Future Work

Firewall security, like any other technology, requires proper management to provide the 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 firewall rules and the potential network vulnerability due to rule conflicts. The Firewall Policy Advisor presented in this paper provides a number of user-friendly tools for purifying and protecting the firewall policy from anomalies. The administrator can use the firewall policy advisor to manage a general firewall security policy without prior analysis of filtering rules. In this work, we formally defined all possible firewall rule relations and we used this to classify firewall policy anomalies. We then model the firewall rule information and relations in a tree-based representation. Based on this model and formalization, the firewall policy advisor implements two management tools:

- **Policy Anomaly Detector** for identifying conflicting, shadowing, correlated and redundant rules. When a rule anomaly is detected, users are prompted with proper corrective actions. We intentionally made the tool not to automatically correct the discovered anomaly but rather alarm the user because we believe that the administrator is the one who should do the policy changes.

- **Policy Editor** for facilitating rules insertion, modification and deletion. The policy editor automatically determines the proper order for any inserted or modified rule. It also gives a preview of the changed parts of the policy whenever a rule is removed to show the affect on the policy before and after the removal.

The firewall policy advisor is shown to be very useful and effective when used on real firewall rules in different academic and industrial environments [1]. However, we believe that there is more to do in firewall policy management area. Our future research plan includes extending the proposed techniques to handle distributed firewall policies with centralized or distributed repositories, classifying different semantics in firewall policies and extracting them from the filtering rules, translating low-level filtering rules into high-level textual description, providing a query-based policy analysis algorithms to enhance our visualization of the underlying firewall security policy.

Acknowledgments

We gratefully thank Iyad Kanj for his feedback on the theory work in this paper. We would also like to thank Lopamudra Roychoudhuri and Yongning Tang for their useful comments on an earlier version of this paper.

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] Y. Bartal., A. Mayer, K. Nissim and A. Wool. "Firmato: A Novel Firewall Management Toolkit." *Proceedings of 1999 IEEE Symposium on Security and Privacy*, May 1999.

[3] D. Chapman and E. Zwicky. *Building Internet Firewalls, Second Edition*, Orielly & Associates Inc., 2000.

[4] W. Cheswick and S. Belovin. *Firewalls and Internet Security*, Addison-Wesley, 1995.

[5] S. Cobb. "ICSA Firewall Policy Guide v2.0." NCSA Security White Paper Series, 1997.

[6] D. Eppstein and S. Muthukrishnan. "Internet Packet Filter Management and Rectangle Geometry." *Proceedings of 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2001.

[7] P. Eronen and J. Zitting. "An Expert System for Analyzing Firewall Rules." *Proceedings of 6th Nordic Workshop on Secure IT-Systems (NordSec 2001)*, November 2001.

[8] 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.

[9] J. Guttman. "Filtering Posture: Local Enforcement for Global Policies." *Proceedings of 1997 IEEE Symposium on security and Privacy*, May 1997.

[10] B. Hari, S. Suri and G. Parulkar. "Detecting and Resolving Packet Filter Conflicts." *Proceedings of IEEE INFOCOM'00*, March 2000.

[11] S. Hazelhusrt. "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.

[12] S. Hinrichs. "Policy-Based Management: Bridging the Gap." *Proceedings of 15th Annual Computer Security Applications Conference (ACSAC'99)*, December 1999.

[13] E. Lupu and M. Sloman. "Conflict Analysis for Management Policies." *In Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM'1997)*, May 1997.

[14] A. Mayer, A. Wool and E. Ziskind. "Fang: A Firewall Analysis Engine." *Proceedings of 2000 IEEE Symposium on Security and Privacy*, May 2000.

[15] L. Qiu, G. Varghese, and S. Suri. "Fast Firewall Implementations for Software and Hardware-based Routers." *Proceedings of 9th International Conference on Network Protocols (ICNP'2001)*, November 2001.

[16] V. Srinivasan, S. Suri and G. Varghese. "Packet Classification Using Tuple Space Search." *Computer ACM SIGCOMM Communication Review*, October 1999.

[17] T. Woo. "A Modular Approach to Packet Classification: Algorithms and Results." *Proceedings of IEEE INFOCOM'00*, March 2000.

[18] A. Wool. "Architecting the Lumeta Firewall Analyzer." *Proceedings of 10th USENIX Security Symposium*, August 2001.

[19] "Cisco Secure Policy Manager 2.3 Data Sheet." http://www.cisco.com/warp/public/cc/pd/sqsw/sqppmn/prodlit/spmgr_ds.pdf

[20] "Check Point Visual Policy Editor Data Sheet." http://www.checkpoint.com/products/downloads/vpe_datasheet.pdf