Modeling and Management of Firewall Policies

Ehab S. Al-Shaer and Hazem H. Hamed

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.**

I. 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 written based on predefined security policy requirements.

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 [1].

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 issues [2]–[4]. On the other hand, a few related works [5], [6] attempt to address only one of the conflict problems which is the rule correlation in filtering policies. Other approaches [7]–[9] propose using a high-level policy language to define and analyze firewall policies and then map this language to filtering rules. Although using such high-level languages might avoid rule anomalies, they are not practical for the most widely used firewalls that contain low-level filtering rules. It is simply because redefining already existing policies using high-level languages require far more effort than just analyzing existing rules using stand-alone tools such as the Firewall Policy Advisor. Therefore, we consider our work a significant progress in the area as it offers a novel and comprehensive framework to automate anomaly discovery and rule editing in legacy firewalls.

This paper is organized as follows. In Section II we give an introduction to firewall operation. In Section III we present our formalization of filtering rule relations. In Section IV, we classify and define firewall policy anomalies, and then we describe the anomaly discovery algorithm. In Section V we describe the techniques for anomaly-free rule editing. In Section VI we show the implementation and evaluation of the Firewall Policy Advisor. In Section VII, we give a summary of related work. Finally, in Section VIII, we show our conclusions and our plans for future work.

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

II. 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 rules that define the actions performed on network

```
                  Source          Destination
    Protocol    Address    Port    Address    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,    140.192.38.*,  any,  161.120.35.*, any,  accept
12: udp,        *.*.*.*,   any,      *.*.*.*, any,     deny
```
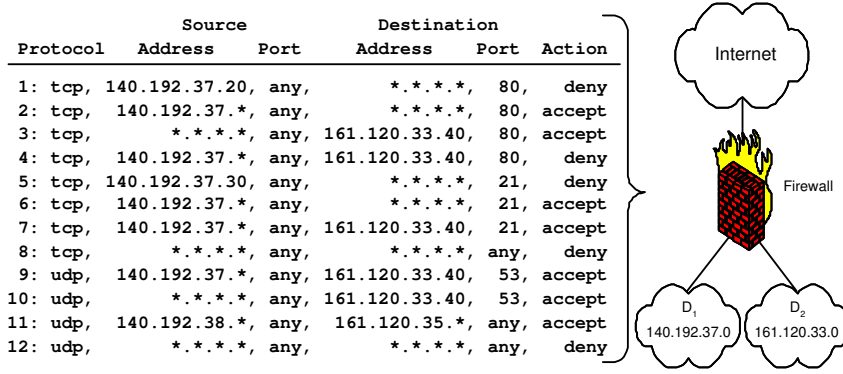
Fig. 1.   An example of a typical firewall filtering policy.

packets based on specific filtering conditions. A rule is composed of set of filtering fields (also called network fields) such as protocol type, source and destination IP addresses and ports, as well as an action field. The filtering fields of a rule represent the possible values of the corresponding fields in actual network traffic that matches this rule. Each network field could be a single value or range of values. Filtering actions are either to *accept*, which permits the packet into or from the secure network, or to *deny*, which causes the packet to be blocked. The packet is permitted or blocked 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 [10], [11]. In this paper, we assume a "deny" default policy action.

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 [12], [13]. The following is the common format of packet filtering rules in a firewall policy:

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 in the policy. 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 [2]. An example of typical firewall rules is shown in Fig. 1.

III. FIREWALL POLICY MODELING

Modeling of firewall rule relations 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.

A. 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 packet filters. In this section we define all the possible relations that may exist between filtering rules, and we show that no other relation exists. We determine these relations based on comparing the network fields of filtering rules, independent of the rule actions. In the next section, we consider these relations as well as rule actions in our study of firewall rule conflicts.

Definition 1: Rules R_x and R_y are *completely disjoint* if every field in R_x is not a subset nor a superset nor equal to the corresponding field in R_y. Formally, $R_x \Re_{CD} R_y$ iff

$$\forall i : R_x[i] \not\bowtie R_y[i]$$
$$\text{where } \bowtie \in \{\subset, \supset, =\},$$
$$i \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$$

Definition 2: Rules R_x and R_y are *exactly matching* if every field in R_x is equal to the corresponding field in R_y. Formally, $R_x \Re_{EM} R_y$ iff

$$\forall i : R_x[i] = R_y[i]$$
$$\text{where } i \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$$

Definition 3: Rules R_x and R_y are *inclusively matching* 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, $R_x \Re_{IM} R_y$ iff

$$\forall i : R_x[i] \subseteq R_y[i]$$
$$\text{and } \exists j \text{ such that } : R_x[j] \neq R_y[j]$$
$$\text{where } i, j \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}$$

For example, in Fig. 1, Rule 1 inclusively matches Rule 2. Rule 1 is the subset match while Rule 2 is the superset match.

Definition 4: Rules R_x and R_y are *partially disjoint* (or *partially matching*) 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
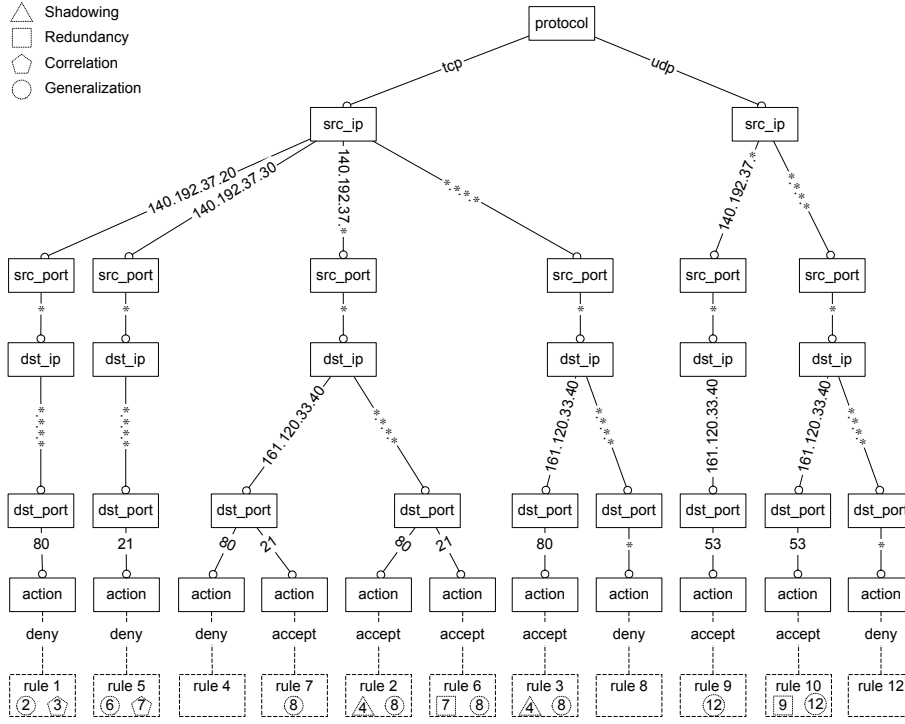
Fig. 2.   The policy tree for the firewall policy in Fig. 1.

and not a superset and not equal to the corresponding field in R_y. Formally, $R_x \Re_{PD} R_y$ iff

$$\exists i, j \text{ such that } R_x[i] \bowtie R_y[i] \text{ and } R_x[j] \not\bowtie R_y[j]$$
$$\text{where } \bowtie \in \{\subset, \supset, =\},$$
$$i, j \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}, i \neq j$$

For example, Rule 2 and Rule 6 in Fig. 1 are partially disjoint (or partially matching).

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, $R_x \Re_C R_y$ iff

$$\forall i : R_x[i] \bowtie R_y[i] \text{ and}$$
$$\exists j, k \text{ such that} : R_x[j] \subset R_y[j] \text{ and } R_x[k] \supset R_y[k]$$
$$\text{where } \bowtie \in \{\subset, \supset, =\},$$
$$i, j, k \in \{\text{protocol, s\_ip, s\_port, d\_ip, d\_port}\}, j \neq k$$

For example, Rule 1 and Rule 3 in Fig. 1 are correlated.

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.

Theorem 1: Any two k-tuple filters in a firewall policy are related by one and only one of the defined relations.

Proof: Intuitively, we can simply show that the intersection between any two relations in $\Re$ is an empty set. In [14], we prove by contradiction that there are no two rules R_x and R_y

such that $R_x \Re_1 R_y$ and $R_x \Re_2 R_y$ and $\Re_1 \neq \Re_2$.

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

Proof: Intuitively, we first prove that the relation between any two 2-tuple filters, R_x and R_y, must be in $\Re$. This is simply shown by enumerating permutations of all possible relations between the fields of R_x and R_y. Since any two fields can be related with any relation in $\{=, \subset, \supset, \neq\}$, thus there are only 16 possible combinations for R_x and R_y relations (e.g. $R_x[field_1] = R_y[field_1], R_x[field_2] \subset R_y[field_2] \Rightarrow R_y \Re_{IM} R_x$. This shows that any relation between R_x and R_y must be one of the rules relations in $\Re$. Next, we prove that adding one more field to any two filters related with one of the defined relations above will produce two filters, R'_x and R'_y, that are also related by one of the defined relations above. We can also show this by enumerating all combinations between R_x and R_y and the added field as follows: $R_x \Re_{IM} R_y$, $R_x[field_k + 1] \supset R_y[field_k + 1] \Rightarrow R'_y \Re_C R'_x$. Based on these two results, we then prove by induction that any two rules with k-tuple filters must be related with each other by one of the rule relations defined in this section.

B. Firewall Policy Representation

We represent the firewall policy by a single-rooted tree called the *policy tree*. The tree model provides a simple representation of the filtering rules and at the same time allows for easy discovery of relations and anomalies among these rules. Each node in a policy tree represents a network field,

and each branch at this node represents a possible value of the associated field. 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. 2 illustrates the policy tree model of the filtering policy given in Fig. 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 later in the following section. The tree shows that each of Rules 1 and 5 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 subset or superset branches to preserve the relations between the policy rules.

IV. Firewall Anomaly Discovery

The ordering of filtering rules in a centralized firewall policy is very crucial in determining the filtering policy within this firewall. This is because the packet filtering process is performed by sequentially matching the packet against filtering rules until a match is found. If filtering rules are 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 related rules are not carefully ordered, some rules may never be used because of other rules, resulting in an incorrect policy. Moreover, when the policy contains a large number of filtering rules, the possibility of writing conflicting or redundant rules is relatively high.

An firewall policy anomaly is defined as the existence of two or more filtering rules that may match the same packet or the existence of a rule that can never match any packet on the network paths that cross the firewall. In this section, we classify different anomalies that may exist among filtering rules in one firewall and then describe a technique for discovering these anomalies.

A. Firewall Anomaly Classification

Here, we describe and then formally define the possible firewall policy anomalies.

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. Formally, rule R_y is shadowed by rule R_x if one of the following conditions

holds:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{EM}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$
$$R_x[\text{order}] < R_y[\text{order}], R_y \Re_{\text{IM}} R_x, R_x[\text{action}] \neq R_y[\text{action}]$$

For example, Rule 4 in shadowed by Rule 3 in Fig. 1. 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 superset (or general) rule should come after the subset (or specific) rule. It is important to discover shadowed rules and alert the administrator to correct this error by reordering or removing these rules.

2) Correlation anomaly: Two rules are correlated if they have different filtering actions, and the first rule matches some packets that match the second rule and the second rule matches some packets that match the first rule. Formally, rule R_x and rule R_y have a correlation anomaly if the following condition holds:

$$R_x \Re_{\text{C}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

Rule 1 is in correlation with Rule 3 in Fig. 1. The two rules with this ordering imply that all HTTP traffic that is coming from 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 stated by the filtering rules. 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 a preceding rule if they have different actions, and if the first rule can match all the packets that match the second rule. Formally, rule R_y is a generalization of rule R_x if the following condition holds:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{IM}} R_y, R_x[\text{action}] \neq R_y[\text{action}]$$

Rule 2 is a generalization of Rule 1 in Fig. 1. These two rules imply that all HTTP traffic that is coming from the address 140.192.37.* will be accepted, except the traffic coming 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. This might cause an accepted traffic to be blocked or a denied traffic to be permitted, and thus it is important to highlight its action to the administrator for confirmation.

4) Redundancy anomaly: A rule is redundant if there is another rule that produces the same matching and action such that if the redundant rule is removed, the security policy will not be affected. Formally, rule R_y is redundant to rule R_x if one of the following conditions holds:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{EM}} R_y, R_x[\text{action}] = R_y[\text{action}]$$
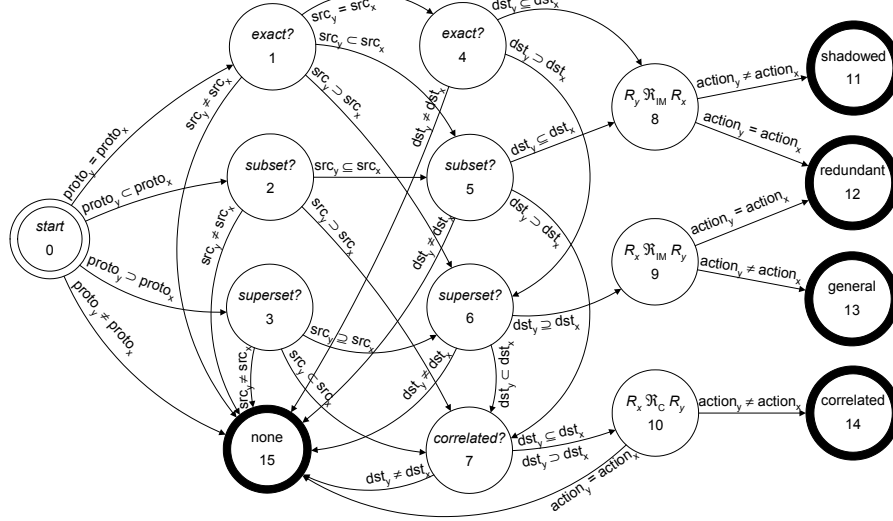$$R_x[\text{order}] < R_y[\text{order}], R_y \Re_{\text{IM}} R_x, R_x[\text{action}] = R_y[\text{action}]$$

Fig. 3. State diagram for detecting firewall anomalies for rules R_x and R_y, where R_y comes after R_x.

Whereas rule R_x is redundant to rule R_y if the following condition holds:

$$R_x[\text{order}] < R_y[\text{order}], R_x \Re_{\text{IM}} R_y, R_x[\text{action}] = R_y[\text{action}]$$
$$\text{and} \quad \nexists R_z \text{ where } R_x[\text{order}] < R_z[\text{order}] < R_y[\text{order}],$$
$$R_x\{\Re_{\text{IM}}, \Re_{\text{C}}\}R_z, R_x[\text{action}] \neq R_z[\text{action}]$$

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 firewall policy because a redundant rule adds to the size of the filtering rule list, and therefore increases the search time and space requirements of the packet filtering process [15]. In general, to avoid redundant rules, a superset rule following a subset rule should have an opposite filtering action. In some cases, redundancy might be preferred to ensure that a desired action is performed on a specific traffic, but in general it is considered an anomaly. It is important to discover redundant rules so that the administrator can decide whether to keep these rules, modify their filtering actions, or remove them from the policy.

5) Irrelevance anomaly: A filtering rule in a firewall is irrelevant if this rule does match any traffic that may flow through this firewall. This exists when both the source address and the destination address fields of the rule do not match any domain reachable through this firewall. In other words, the path between the source and destination addresses of this rule does not pass through the firewall. Thus, this rule has no effect on the filtering outcome of this firewall. Formally, rule R_x in firewall F is irrelevant if:

$$F \notin \{n : n \text{ is a node on a path from } R_x[\text{src}] \text{ to } R_x[\text{dst}]\}$$

Referring to Fig. 1, Rule 11 is irrelevant because the traffic that goes between the source (140.192.38.*) and the destination (161.120.35.*) does not pass through this firewall.

Irrelevance is considered an anomaly because it adds unnecessary overhead to the filtering process and it does not contribute to the policy semantics. It is well-understood that keeping the filtering rule table as small as possible helps in improving the overall firewall performance [13], [15]. Thus, discovering irrelevant rules is an important function for the network security administrator.

B. Firewall Anomaly Discovery Algorithm

The state diagram in Fig. 3 illustrates the firewall anomaly discovery states for any two rules, R_x and R_y, where the two rules are in the same firewall and R_y follows R_x. For simplicity, the address and port fields are integrated in one field for both the source and destination. This reduces the number of states and simplifies the explanation of the diagram.

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. Identifying irrelevant rules requires the knowledge of the network connectivity and therefore it is not included in the diagram. Irrelevance anomalies can be discovered simply by verifying that each rule in the policy matches a source and a destination address that lie on a path controlled by the firewall. If none of the preceding cases occur, then the two rules do not involve any anomalies. As an example, we match Rule 4 against Rule 3 in Fig. 1. The matching process

transits through the intermediate states 1-5-8 and terminates at state 11 because Rule 4 is a subset match of Rule 3 and has a different action.

The basic idea for discovering anomalies is to determine if any two rules coincide in their policy tree paths. If the path of a rule coincides with the path of another rule, there is a potential anomaly that can be determined based on the firewall anomaly definitions in Section IV-A. If rule paths do not coincide, then these rules are disjoint and they have no anomalies. Algorithm 1 discovers rule anomalies by implementing the state transition diagram shown in Fig. 3. The algorithm can be divided into two phases: the state transition phase (lines 2-23), which represents the transition states in the state diagram, and the state termination phase (lines 24-43), which represents the termination states.

The transition routine is invoked upon inserting every rule in the policy tree. If the field of the current rule matches an already existing rule branch, then the next relation state is determined based on the shown state diagram. The algorithm is executed iteratively to let the rule propagate in existing branches and check the remaining fields. As the rule propagates, the relation state is updated until the final state is reached. If there is no match for a field value, the relation state is set to disjoint. The termination routine is activated once all the rule fields have been matched and the action field is reached. If the rule action coincides with the action of another rule on the tree, an anomaly is discovered. At that point the final anomaly state is determined and any anomalies are reported together with 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.

V. Firewall Policy Editing

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 the rule is modified or removed. In this section, we present firewall policy editing techniques that simplify the rule editing task significantly, and avoids introducing anomalies due to policy updates. The policy editor helps the user to determine the proper order for a new or modified rule in the policy, and provides visual aids for users to track and verify policy changes. Using the policy editor, administrators need no prior analysis of the firewall policy rules in order to insert, modify or remove a rule.

Algorithm 1 Firewall anomaly discovery algorithm.

Input: $rule, branch$
Output: $anomaly$

1: **for each** $field \in rule.fields$ **do**
2:    **if** $field \neq$ ACTION **then** {find transition states}
3:      $relation \leftarrow$ UNDETERMINED
4:      **if** $branch = field$ **then** {exact match}
5:        **if** $relation =$ UNDETERMINED **then**
6:          $relation \leftarrow$ EXACT
7:        **end if**
8:      **else if** $field \supset branch$ **then** {superset match}
9:        **if** $relation \in$ {SUBSET, CORRELATED} **then**
10:          $relation \leftarrow$ CORRELATED
11:        **else if** $relation \neq$ DISJOINT **then**
12:          $relation \leftarrow$ SUPERSET
13:        **end if**
14:      **else if** $field \subset branch$ **then** {subset match}
15:        **if** $relation \in$ {SUPERSET, CORRELATED} **then**
16:          $relation \leftarrow$ CORRELATED
17:        **else if** $relation \neq$ DISJOINT **then**
18:          $relation \leftarrow$ SUBSET
19:        **end if**
20:      **else** {not matching}
21:        $relation \leftarrow$ DISJOINT
22:      **end if**
23:      $branch \leftarrow branch.next$
24:    **else** {find termination state}
25:      $anomaly \leftarrow$ NOANOMALY
26:      **if** $relation \neq$ DISJOINT **then**
27:        **if** $relation =$ CORRELATED and $field \neq branch$ **then** {different actions}
28:          $anomaly \leftarrow$ CORRELATION
29:        **else if** $relation =$ SUPERSET **then**
30:          **if** $field = branch$ **then** {similar actions}
31:            $anomaly \leftarrow$ REDUNDANCY
32:          **else** {different actions}
33:            $anomaly \leftarrow$ GENERALIZATION
34:          **end if**
35:        **else if** $relation \in$ {EXACT, SUBSET} **then**
36:          **if** $field = branch$ **then** {similar actions}
37:            $anomaly \leftarrow$ REDUNDANCY
38:          **else** {different actions}
39:            $anomaly \leftarrow$ SHADOWING
40:          **end if**
41:        **end if**
42:      **end if**
43:    **end if**
44: **end for**

Algorithm 2 Firewall rule insertion algorithm.
___
Input: $rule, tree$
Output: $min\_order, max\_order, anomaly$
1: $max\_order, max\_order \leftarrow$ UNDERTERMINED
2: $node \leftarrow tree.root$
3: **for each** $field \in rule.fields$ **do**
4:    **if** $field \neq$ ACTION **then**
5:      $target \leftarrow$ nil
6:      **for each** $branch \in node.branches$ **do**
7:        **if** $branch = field$ **then**
8:          $target \leftarrow branch$
9:        **else if** $field \subset branch$ **then**
10:          **if** $max\_order >$ MinOrder($branch$) **then**
11:            $max\_order \leftarrow$ MinOrder($branch$)-1
12:            $target \leftarrow branch$
13:          **end if**
14:        **else if** $field \supset branch$ **then**
15:          **if** $min\_order <$ MaxOrder($branch$) **then**
16:            $min\_order \leftarrow$ MaxOrder($branch$)+1
17:          **end if**
18:        **end if**
19:      **end for**
20:      **if** $target \neq$ nil **then** {browse target branch}
21:        $node \leftarrow target.next$
22:      **else** {create new branch}
23:        $node \leftarrow$ NewBranch($node, field$)
24:      **end if**
25:    **else if** $target \neq$ nil **then** {and action field reached}
26:      $anomaly \leftarrow$ NOANOMALY
27:      **if** $min\_order =$ UNDERTERMINED
        and $max\_order =$ UNDERTERMINED **then**
28:        **if** $field = branch$ **then** {similar actions}
29:          $anomaly \leftarrow$ REDUNDANCY
30:        **else** {different actions}
31:          $anomaly \leftarrow$ SHADOWING
32:        **end if**
33:      **else if** $max\_order \neq$ UNDERTERMINED
        and $field = branch$ **then** {similar actions}
34:        $anomaly \leftarrow$ REDUNDANCY
35:      **end if**
36:    **end if**
37: **end for**
___

A. 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 anomalies are 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 order of the new rule in the firewall 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. Algorithm 2 uses the local policy tree 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 (line 11). 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 (line 16). On the other hand, if the rule is disjoint, then it can be given any order in the policy. Similarly, the tree browsing continues, matching the next fields in the rule as long as a field value is an exact match or a subset match of a branch (lines 8, 12, 21). A new branch is created for the new rule any time a disjoint or superset match is found (line 23). When the action field is reached, the rule is inserted and assigned an order within the maximum and minimum range determined in the browsing phase. 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 (lines 27-35).

After inserting the rule in the proper position, the anomaly discovery algorithm in Section IV-B is activated to identify any correlation or generalization anomalies that might be introduced by the new rule.

B. Rule Removal

In general, removing a rule has much less impact on the firewall policy than insertion. A removed rule might not introduce an anomaly but may change the policy semantics. Therefore, the policy changes due to removing a rule should be highlighted and confirmed. To preview the effect of rule removal, the policy editor extracts and displays the affected portion of the policy before and after the rule is removed. This is done by extracting all the rules that match the removed rule in the policy tree. This way, the user is able to inspect and compare the policy semantics before and after removal, and re-assure the correctness of the changes. In some cases, a removed rule may introduce rule redundancy if it exists between a preceding subset or correlated rule and a succeeding superset rule. Referring to the policy example in Fig. 1, if both Rule 2 and Rule 3 are removed, Rule 1 becomes redundant in the policy. After a rule is removed, the policy editor highlights any redundant rules and prompts the user for a correcting action.

Modifying a rule in a firewall policy is also a critical operation. However, a modified rule can be easily verified and inserted based on the rule removal and insertion techniques described above.

VI. FIREWALL POLICY ADVISOR: IMPLEMENTATION AND EVALUATION

We implemented the techniques and algorithms described in Section IV in a software tool called the "Firewall Policy Advisor" or FPA [1]. The tool implements the firewall anomaly

___
[1] http://www.mnlab.cti.depaul.edu/projects/FPA

Experience	Shadowing	Redundancy	Correlation	Irrelevance
Expert	0%	5%	3%	0%
Intermediate	1%	9%	3%	0%
Beginner	4%	12%	9%	2%

Fig. 4. The average percentage of discovered anomalies in a man-written firewall policy.

discovery algorithm, as well as the firewall policy editor. The FPA was developed using Java programming language and it includes a graphical user interface. In this section, we present our evaluation study of the usability and the performance of the anomaly discovery techniques described in this paper.

To assess the practical value of our techniques, we first used the FPA tool to analyze real firewall rules in our university network as well as in some local industrial networks in the area. In many cases, the FPA has shown to be effective by discovering many firewall anomalies that were not discovered by human visual inspection. We then attempted to quantitatively evaluate the practical usability of the FPA by conducting an experiment that considers the level of network administrator expertise and the frequency of occurrence of each anomaly type. In this experiment, we created a firewall policy exercise and asked twelve network administrators with varying level of expertise to complete each exercise. The exercise includes writing the filtering rules for a network with a single firewall based on a given security policy requirements. We then used the FPA tool to analyze the rules in the answer of each one and calculated the ratio of the occurrence of each anomaly relative to total number of rules. The average total number of rules was 40 for the given firewall exercise. The results of this experiment are shown in Fig. 4.

These results show clearly that the margin of error that can be done even by an expert administrator is quite significant (about 8%). This figure is even much higher for an intermediate and beginner administrators (about 13% and 27%). Another interesting observation is the high percentage of redundant rules in all experience levels.

In the last phase of our evaluation study, we conducted a number of experiments to measure the performance and the scalability of firewall anomaly discovery under different filtering policies. Our experiments were performed on a Pentium PIII 400 MHz processor with 128 MByte of RAM.

To study the performance of the firewall anomaly discovery algorithm, we produced four sets of firewall rules. The first set includes rules that are different in the destination address only, and the second set includes rules that have distinct source addresses. These two sets resemble a realistic combination of practical firewall rules, and represent the best case scenario because they require the minimum policy-tree navigation for analyzing each rule. In the third set, each rule is a superset match of the preceding rule. This set represents the worst case scenario because each rule requires complete policy-tree navigation in order to analyze the entire rule set. The fourth set includes rules that are randomly selected from the three previous sets in order to represent the average case scenario. We used the FPA tool to run the firewall policy
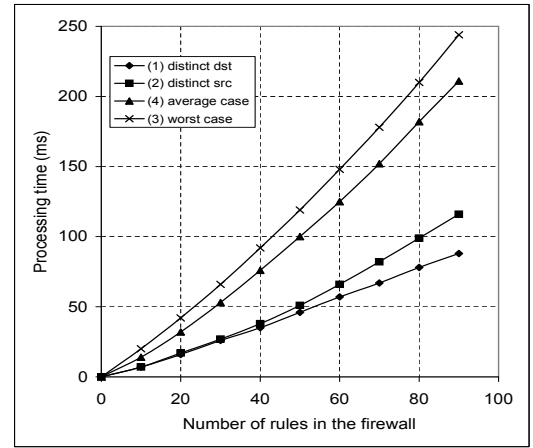

Fig. 5. Processing time for firewall anomaly discovery.

analysis algorithm on each set using various sizes of rule sets (10-90 rules). In each case, we measured the processing time needed to produce the policy analysis report.

The results we obtained are shown in Fig. 5. Set 1 shows the least processing time because all the rules are aggregated in one branch in the policy tree, which makes the number of field matches and node creations minimal. Set 2 has a slightly higher processing time, since each rule in the set has a distinct branch at a higher level in the policy tree. This requires more time to create new tree nodes for inserting each rule in the tree. Set 3 is expected to have the highest processing time since every rule in the set must be matched with all rules in the policy tree. Set 4 shows a moderate (average) processing time and represents the most practical scenario as it combines many different cases. Even in the worst case scenario (Set 3), the processing time looks very reasonable; approximately 20-240 ms for 10-90 rules. In addition, the processing time increases about 2.1-2.8 ms per rule, which is considered insignificant overhead even if hundreds of rules exist in a firewall.

VII. 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 the related work that intersects with our work in three areas: packet filter modeling, 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 [16]. Another model using tuple space is developed in [3], which combines a set of filters in one tuple stored in a hash table. The model in [4] uses bucket filters indexed by search trees. Multi-dimensional binary tries are also used to model filters [2]. In [5] 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. Interval diagrams are used in [17] to compact firewall rules. However, it requires pre-processing of firewall rules to resolve any rule overlap, and therefore it cannot be used for our anomaly analysis. We can confirm from experience that the tree-based model we use is simple and powerful enough specifically 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 [18] classify possible policy conflicts in role-based management frameworks, and develop techniques to discover them. A policy conflict scheme for IPSec is presented in [19]. 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 [5] and [6] 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. Other research work goes one step forward by offering query-based tools for firewall policy analysis. In [8] and [9], the authors developed a firewall analysis tool to perform customized queries on a set of filtering rules and extract the related rules in the policy. In [20], an expert system is used for verifying the functionality of filtering rules by performing queries. All these tools help in manually verifying the correctness of the firewall policy, however, they require high user expertise to write the proper queries to identify different firewall policy problems.

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 low-level firewall policies.

VIII. 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 the filtering rules that compose the policy. In this work, we formally defined all possible firewall rule relations and we used this to classify firewall policy anomalies. We then modeled the firewall rule information and relations in a tree-based representation. Based on this model and formalization, we implemented the Firewall Policy Advisor, which incorporates two firewall management tools:

- *Policy Analyzer* 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.

Using Firewall Policy Advisor was shown to be very effective for managing firewall policies in real-life networks. In regards to usability, the tool was able to discover filtering anomalies in rules written by expert network administrators. In regards to performance, although the policy analysis algorithm is parabolically dependant on the number of rules in the firewall policy, our experiments show that the average processing time for anomaly discovery is very reasonable for practical firewall policies. Using our Java implementation of the anomaly discovery algorithms, our results indicate that it, in the worst case, it takes 10-240 ms of processing time to analyze a firewall policy of 10-90 rules.

We believe that there is more to do in firewall policy management area. Our future research plan includes extending the proposed anomaly discovery techniques to handle distributed firewall policies. In this case, the filtering rules in different firewalls should be carefully assigned such that no desired traffic is blocked before reaching its destination, and no undesired traffic is allowed to flow in network segments. Our agenda also includes translating low-level filtering rules into high-level textual description and vice versa, and providing query-based firewall policy analysis tools to enhance our visualization of the underlying firewall security policy.

References

[1] E. Al-Shaer and H. Hamed. "Firewall Policy Advisor for Anomaly Detection and Rule Editing." *Proceedings of IEEE/IFIP Integrated Management Conference (IM'2003)*, March 2003.
[2] 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.
[3] V. Srinivasan, S. Suri and G. Varghese. "Packet Classification Using Tuple Space Search." *Computer ACM SIGCOMM Communication Review*, October 1999.
[4] T. Woo. "A Modular Approach to Packet Classification: Algorithms and Results." *Proceedings of IEEE INFOCOM'00*, March 2000.
[5] 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.

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

[7] 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.

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

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

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

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

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

[13] J. Wack, K. Cutler and J. Pole. "Guidelines on Firewalls and Firewall Policy." *NIST Recommendations, SP 800-41*, January 2002.

[14] E. Al-Shaer and H. Hamed. "Design and Implementation of Firewall Policy Advisor Tools." *DePaul CTI Technical Report, CTI-TR-02-006*, August 2002.

[15] R. Panko. *Corporate Computer and Network Security*, Prentice Hall, 2003.

[16] 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.

[17] M. Gouda and X. Liu. "Firewall Design: Consistency, Completeness, and Compactness." *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, March 2004.

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

[19] 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.

[20] 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.

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

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

Ehab Al-Shaer is an associate professor and the director of the Multimedia Networking Research Laboratory (MNLAB) in the School of Computer Science, Telecommunications and Information System at DePaul University since 1998. He received his Ph.D. in Computer Science from Old Dominion University in 1998 and M.Sc. in Computer Science from Northeastern University in 1994. His primary research areas are network monitoring, fault management, multimedia protocols, and network security. Prof. Al-Shaer has many journal and conference publications in the area of distributed monitoring, Internet measurement, multicast management, and firewall security. He was the conference co-chair for Management of Multimedia Networks and Services (MMNS 2001) and the chair and founder of the 1st Workshop on End-to-End Monitoring Workshop (E2EMON) in September 2003. Prof. Al-Shaer was a guest editor, invited speaker, panellist, session chair, steering committee and program committee member of many major IEEE/IFIP/ACM conferences and seminars. Prof. Al-Shaer is currently authoring his book on monitoring networks and distributed systems with Prentice Hall. He was awarded the best paper award in Integrated Management Conference (IM 2003) and NASA fellowship in 1997.

Hazem Hamed is a Ph.D. candidate in the School of Computer Science, Telecommunications and Information Systems at DePaul University. He received his M.Sc. degree in Computer Engineering and B.Sc. Degree in Computer and Systems Engineering from Ain-Shams University, Egypt in 1999 and 1994 respectively. He is also working as a teaching and research assistant in the Computer Science Department, DePaul University. Mr. Hamed was awarded the best paper award in Integrated Management Conference (IM 2003). His research interests include network security, differentiated services and reliable multicasting.