Optimization of Network Firewall Policies using Directed Acyclical Graphs*

Errin W. Fulp
Department of Computer Science
Wake Forest University
Winston-Salem, NC, USA
nsg.cs.wfu.edu
fulp@wfu.edu

Abstract

This paper introduces a new method to improve the performance of list oriented firewall systems. Specifically, the paper addresses reordering a firewall rule set to minimize the average number of comparisons to determine the action, while maintaining the integrity of the original policy. Integrity is preserved if the reordered and original rules always arrive at the same result given a packet. To maintain integrity, this paper will model the rule set as a Directed Acyclical Graph (DAG), where vertices are firewall rules and edges indicate precedence relationships. Given this representation, any linear arrangement of the policy DAG (which is a list of rules) is shown to maintain the original policy integrity. Unfortunately, determining the optimal rule order from all the possible linear arrangements is shown to be $\mathcal{NP}$-hard, since it is equivalent to sequencing jobs with precedence constraints for a single machine. Although determining the optimal order is $\mathcal{NP}$-hard, this paper will introduce a simple heuristic to order firewall rules that reduces the average number of comparisons while maintaining integrity. Simulation results show the proposed reordering method yields rule orders that are comparable to optimal (11% difference); thus, provides a simple means to significantly improve firewall performance and lower packet delay.

Keywords

security management, firewalls, policy representation, performance optimization

1. Modeling Firewall Security Policies

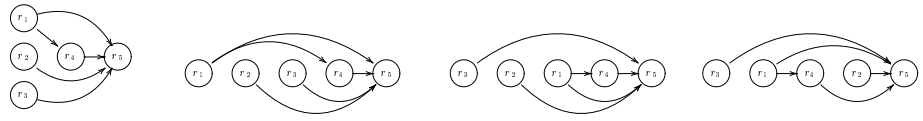
The policy model introduced in this paper is designed for firewall performance optimization and integrity. Firewall performance refers to reducing the average number of comparisons required to determine an action, while integrity refers to maintaining the original policy intent. Although improving the worst case performance is important, it is not possible without changing the list-based representation.

A rule r is modeled as an ordered tuple of sets, $r = (r[1], r[2], ..., r[k])$. Order is necessary among the tuples since comparing rules and packets requires the comparison

*This work was supported by the U.S. Department of Energy MICS (grant DE-FG02-03ER25581). The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DOE or the U.S. Government.

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action	Prob.
1	UDP	1.1.*	*	*	80	deny	0.01
2	TCP	1.*	*	1.*	90	accept	0.02
3	TCP	2.*	*	2.*	20	accept	0.25
4	UDP	1.*	*	*	*	accept	0.22
5	*	*	*	*	*	deny	0.50

Table 1 Example security policy consisting of multiple ordered rules.



(a) Policy DAG. (b) Original order. (c) Sorted order. (d) Optimal order.

Figure 1: Different policy DAG representations of the firewall rules given in table 1.

of corresponding tuples. Each tuple $r[l]$ is a set that can be fully specified, be a range, or contain wildcards '*' in standard prefix format. In addition to the prefixes, each filter rule has an action, which is to accept or deny. However, the action will not be considered when comparing packets and rules. Using this rule model, a standard security policy can be modeled as an ordered set (list) of n rules, denoted as $R = \{r_1, r_2, ..., r_n\}$. A packet d is sequentially compared against each rule r_i starting with the first, until a match is found ($d \Rightarrow r_i$) then the associated action is performed. A match is found between a packet and rule when every tuple of the packet is a subset of the corresponding tuple in the rule.

A rule list has an implied precedence relationship where certain rules must appear before others if the integrity of the policy is to be maintained. For example consider the rule list in table 1. Rule r_1 must appear before rule r_4, likewise rule r_5 must be the last rule in the policy. If for example, rule r_4 was moved to the beginning of the policy, then it will *shadow* [1] the original rule r_1. However, there is no precedence relationship between rules r_1, r_2, or r_3 given in table 1. The relative ordering of these three rules will not impact the policy integrity, and can be changed to improve performance. Therefore, a model is needed to represent precedence if performance is to be improved via reordering.

1.1 Modeling Precedence Relationships

Let $G = (R, E)$ be a *policy DAG* for a rule list R, where vertices are rules and edges E are the precedence relationships (constraint). A precedence relationship, or edge, exists between rules r_i and r_j, if $i < j$ and the rules intersect. The intersection of two rules results in an ordered set of tuples that collectively describes the packets that match both rules. Rules r_i and r_j intersect if the intersection of every corresponding tuple is non-empty. In contrast, the rules r_i and r_j do not intersect, denoted as $r_i \not\cap r_j$, if at least one tuple is the empty set. An example policy DAG is given in figure 1(a).

Using the policy DAG representation a linear arrangement is sought that improves the firewall performance. As depicted in figures 1(b), 1(c), and 1(d), a linear arrangement (permutation or topological sort) is a list of DAG vertices where all the successors of a vertex appear in sequence after that vertex. Therefore it follows that a linear arrangement of a policy DAG represents a rule order, if the vertices are read from left to right. Furthermore, it is proven in [2] that any linear arrangement of a policy DAG maintains integrity.

2. Rule List Optimization

Using the policy DAG to maintain policy integrity, a linear arrangement is sought that minimizes the average number of comparisons required. Certain firewall rules have a higher probability of matching a packet than others. As a result, it is possible to develop a *policy profile* over time that indicates frequency of rule matches (similar to cache hit ratio). Let $P = \{p_1, p_2, ..., p_n\}$ be the policy profile, where p_i is the probability that a packet will match rule i (first match in the policy). Using this information, the average number of rule comparisons required is $A[n] = \sum_{i=1}^{n} i \cdot p_i$

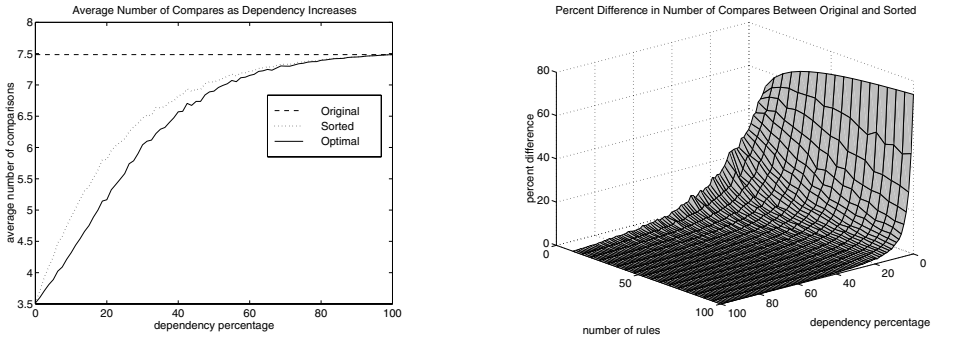
Given a policy DAG $G = (R, E)$ and policy profile $P = \{p_1, p_2, ..., p_n\}$ a linear arrangement π of G is sought that minimizes average number of comparisons. In the absence of precedence relationships, the average number of comparisons is minimized if the rules are sorted in non-increasing order according to the probabilities (Smith's algorithm). Precedence constraints cause the problem to be more realistic; however, it also makes determining the optimal permutation more problematic. Determining the optimal rule list permutation can be viewed as job scheduling problem for a single machine with precedence constraints. Lawler [3] proved such problems to be $\mathcal{NP}$-hard, therefore determining the optimal firewall rule order is as well [2].

3. A Simple Rule Sorting Algorithm

Although finding the optimal rule order is $\mathcal{NP}$-hard, reducing the average number of comparisons remains the objective. A simple algorithm sorts neighboring rules based on non-increasing probabilities. However, an exchange of neighbors should never occur if the rules intersect. This test preserves any precedence relationships in the policy. For example, the following sorting algorithm uses such a comparison to determine if neighboring rules should be interchanged.

```
done = false
while(!done)
        done = true
        for(i = 1; i < n; i++)
                if(p_i < p_{i+1} AND r_i ⋔ r_{i+1})then
                        interchange rules and probabilities
                        done = false
                endif
        endfor
endwhile
```

Consider the match probabilities for the rule list given in table 1. Applying the sorting algorithm to this rule list results in the ordering depicted in figure 1(c), which has 11% fewer comparisons on average. However when using the algorithm, it is possible that



(a) Average number of comparisons for 15 rule policy.

(b) Comparison percent difference for larger policies.

Figure 2: Experimental results for sorting rule lists.

one rule can prevent another rule from being reordered (for example rule r_1 prevents the reorder of rule r_4). The optimal order for the 5 rule list is depicted in figure 1(d).

4. Experimental Results

In this section, the performance of the sorting algorithm is measured using simulation. For the experiments, the percent dependency of the rules varied from completely independent (no policy DAG edges) to completely dependent. The first experiment considered policies consisting of 15 rules. As seen in the figure 2(a), the sorting algorithm remains close to the optimal value with a maximum 11% difference. Although not optimal, the results indicate the proposed sorting algorithm can provide a significant improvement when the dependency percentage is low. The benefit of rule sorting on larger policies is also of interest; however, it was not possible to determine the optimal ordering once the number of rules approached 15. Therefore, the second experiment used larger rule lists, but only compared the original order and the sorted order. The number of rules ranged from 10 to 100. As seen in figure 2(b), the sorted rule list always performed equal to or better than the original order. As noted in the previous experiment, the percent difference is very large given few dependencies (e.g. 70% decrease for 100 rules with 0% dependency), but approaches zero as the rules are completely dependent. As the number of rules increases, sorting is increasingly beneficial; although only at low dependency percentages. Therefore, a large rule list can benefit from sorting if the dependency percentage is low.

References

[1] Ehab Al-Shaer and Hazem Hamed. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management*, 1(1), 2004.

[2] Errin W. Fulp. Firewall policy models using ordered-sets and directed acyclical graphs. Technical report, Wake Forest University Computer Science Department, 2004.

[3] E. L. Lawler. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2:75 – 90, 1978.