Reducing the Size of Rule Set in a Firewall

MyungKeun Yoon Shigang Chen Zhan Zhang
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611, USA
352 392 2713 (phone), 352 392 1220 (fax), {myoon, sgchen, zzhan}@cise.ufl.edu

***Abstract*— A firewall's complexity is known to increase with the size of its rule set. Complex firewalls are more likely to have configuration errors which cause security loopholes. Until now, two rules can be merged into one only when they are exactly same for all the dimensions except one for which each value of two rules should be adjacent to each other. In this paper, we propose a new and aggressive reduction algorithm which finds a group of rules and replace it with a smaller new group so that the total size of rule set can be reduced. This can not be achievable by any previous work because all of them eliminate rules only when these rules are redundant by other rules in the same rule set. The proposed algorithm is also orthogonal to the previous works so that it can be used to supplement them.**

Key Words: firewall, rule management, network security

I. Introduction

Firewalls are the cornerstones of corporate intranet security. Once a company acquires a firewall, a system administrator must configure and manage it according to a security policy that meets the company's needs. Configuration is a crucial task, probably the most important factor to determine a security level [1]. A firewall rule set consists of a large number of rules. A single rule typically includes a group of source addresses, a group of destination addresses, a group of source ports, a group of destination ports, a group of services and an appropriate action. The action can be either "accept" or "deny." For the sake of brevity, we consider only "accept" or "deny," while recent firewalls can support other types of action such as writing a log record, applying a proxy, and implementing a virtual private network (VPN) [20]. In many firewalls, the rule set is order-sensitive: the firewall checks if the first rule in the rule set applies to a new session. If so, the packets are either dropped or let through according to the action of the first rule. Otherwise, the firewall checks if the second rule applies, and so forth [11].

Analyzing firewall rule sets from various organizations in the telecommunications, financial institutes, etc., Wool quantified the complexity of a rule set in terms of the numbers of rules, network objects, and interfaces [2]. While firewall managers intuitively classify a rule set as "complicated" or "simple," Wool quantifies this intuition into a concrete measure of rule complexity (RC) as follows [2]:

$$RC = Rules + Objects + \frac{Interfaces \times (Interfaces - 1)}{2} \tag{1}$$

where $Rules$ is the raw number of rules in the rule set, $Objects$ is the number of network objects, and $Interfaces$ is the number of interfaces on the firewall. As complex rule sets easily cause mistakes and mal-configurations in firewall security, it is very important to configure the rule set as small as possible for enhancing firewall security, or the security of enterprise networks [2]. In most real world networks, because the numbers of $Interfaces$ and $Objects$ are relatively small compared to the number of $Rules$, it is important to make $Rules$ small.

In this paper, we propose how to reduce the size of rule set so that network security can be enhanced. To reduce rule set, previous works rely on merging two rules into one only when they are all the same except one dimension for which the values of the two rules should be adjacent to each other. To the contrary, our approach is new and aggressive in that it finds a group of rules and replace it with a smaller new group. Our algorithm is also orthogonal to the previous works like [4] so that it can be used together with them to achieve more optimal result. The proposed algorithm can be applied to a general classification problem as well as firewall's rule set.

The rest of paper is organized as follows. Section II surveys the related work. Section III proposes *simple substitutional optimization (SSO)*, which covers the basic idea of the substituting approach to reduce the size of rule set. Section IV presents *complex substitutional optimization (CSO)*, which is based on SSO and the algorithms from [4]. Section V draws the conclusion.

II. Related Works

Gouda and Liu proposed the *firewall decision diagram (FDD)* to represent a rule set as a tree structure. This diagram makes it possible to check the consistency and completeness of the rule set systematically. They developed a sequence of five algorithms that can be applied to FDD to generate a compact sequence of rule set while maintaining the consistency and completeness of the original rule set [4]. They also proposed a method for diverse firewall design and presented algorithms to detect discrepancies between two rule sets for the same firewall [5].

Firewall rule sets have to be written, ordered and distributed carefully in order to avoid firewall policy anomalies that may expose a network to danger. Al-Shaer and Hamed identified anomalies that could exist in a single- or multi-firewall environment. They also presented a set of techniques and algorithms to discover policy anomalies in centralized and distributed legacy firewalls [6]. Smith et al. dealt with the issue of how to place a set of firewalls around a complex network, with varying security needs of the different nodes, in order to minimize cost and delay. An operations research view was proposed into the deployment of multiple firewalls [19].

Since the syntax and semantics of the rule set and their ordering depend on the firewall product/vendor, firewall management becomes difficult. This is akin to the dark age of software development, where programs were written in assembly language and thus the programmer had to know all the idiosyncrasies of the target processor. This problem gets even worse when there are many firewalls dividing a company's intranets into multiple domains [11]. Many approaches were proposed to use a high-level language to define and analyze firewall policies and then map this language to filtering rules [11], [12], [13], [14], [17].

Packet filtering of a firewall can be viewed as a special case of packet classification. The packet classification problem is to determine the first matching rule for each incoming packet at a router. A good tutorial is provided by Gupta et al. on packet classification algorithms such as basic search algorithms, geometric algorithms, heuristic algorithms, and hardware-specific algorithms [9]. Traditionally, researchers of packet classification have focused on how to find matching rules as fast as possible using sophisticated data-structures or hardware-driven approaches, most of which are beyond the scope of this paper [7], [8], [9], [10], [15], [16]. Some of them deal with how to reduce the size of a rule set. Gupta identified special types of redundant rules in his PhD thesis [8], namely backward redundant rules and forward redundant rules, by studying 793 packet classifiers from 101 different Internet Service Providers and enterprise networks with a total of 41,505 rules [3]. A rule r in a packet classifier is backward redundant iff there exists another rule r' listed above r such that all packets that match r also match r'. Gupta observed that on average 7.8% of the rules in a packet classifier are backward redundant [8]. A rule r in a packet classifier is forward redundant iff there exists another rule r' listed below r such that the following three conditions hold: (1) all packets that match r also match r', (2) r and r' have the same decision, (3) for each rule r" listed between r and r' either r and r" have the same decision, or no packet matches both r and r". Gupta observed that on average 7.2% of the rules in a packet classifier are forward redundant [8].

Liu et al. suggested how to find and eliminate upward redundant rules and downward redundant rules [3]. Upward/downward redundant rules include backward/forward redundant rules. A rule r in a packet classifier is upward redundant iff there is no packet whose first matching rule is r. Geometrically, a rule is upward redundant in a packet classifier if the rule is overlayed by some rules listed above it. A rule r in a packet classifier, where no rule is upward redundant, is downward redundant iff for each packet, whose first matching rule is r, the first matching rule below r has the same decision as r [3].

Useful tips on implementing firewalls are presented in [20]. This practical document addresses concepts relating to the design, selection, deployment, and management of firewalls and firewall environment.

TABLE I
FIREWALL RULE SET AND OPTIMIZATION

(A) Original rule set	**(B) Optimized rule set [4]**
(1)F_0∈[1,4]∧F_1∈[1,10] d	(1)F_0∈[5,6]∧F_1∈ [3,8] a
(2)F_0∈[5,8]∧F_1∈[1,2] d	(2)F_0∈[7,8]∧F_1∈ [3,4] a
(3)F_0∈[5,6]∧F_1∈[3,8] a	(3)F_0∈[7,8]∧F_1∈ [6,8] a
(4)F_0∈[7,8]∧F_1∈[3,4] a	(4)F_0∈[1,10]∧F_1∈ [1,10] d
(5)F_0∈[7,8]∧F_1∈[5,5] d	**(C) Best optimization**
(6)F_0∈[7,8]∧F_1∈[6,8] a	(1)F_0∈[7,8]∧F_1∈ [5,5] d
(7)F_0∈[5,8]∧F_1∈[9,10] d	(2)F_0∈[5,8]∧F_1∈ [3,8] a
(8)F_0∈[9,10]∧F_1∈[1,10] d	(3)F_0∈[1,10]∧F_1∈ [1,10] d

The usefulness of direction-based filtering in firewalls is shown in [21]. It shows that the discrepancy can make it very confusing for firewall managers to recognize the direction of a packet correctly. Improving traditional firewall performance is shown in [22] by reducing the number of rule comparisons required per packet. This is done by placing the most frequently matched rules around the beginning of the rule set. Directed acyclical graphs are used to represent the presence of rules efficiently. It is proved as an NP-complete problem to find an optimal graph.

III. SIMPLE SUBSTITUTIONAL OPTIMIZATION (SSO)

A. Definition

A firewall's rule set consists of rules where first coming rules have higher priority. We denote rule r_i as $r_i = (F_0 \in S_0 \wedge \ldots F_i \in S_i \wedge \ldots F_{n-1} \in S_{n-1}\ Action)$ in accordance with the notations used in [4]. For simplicity, we denote rule r_i's S_m value as $r_i.S_m$. We only consider two types of $Action$: a (accept) or d (deny). Each F_i represents for a unique dimension. In general, F_i can be IP addresses (source/destination), or port numbers (source/destination), or protocol types (TCP/UDP/ICMP). S_i is range-values. For simplicity, we assume that each dimension ranges from 1 to 10. In the real world, the dimension of IP address ranges from 0 to 2^{32}-1 while port numbers range from 0 to $2^{16} - 1$.

Table I shows three rule sets, (A), (B), and (C), all of which have the same meaning. In this example, each rule set consists of two dimensions. Table I (A) has 8 rules that can be represented by two-dimensional rectangles as in [10]. Fig. 1 illustrates table I (A) as two dimensional rectangles.

In this paper, we say that two rules r_i and r_j are adjacent when they satisfy two conditions: $r_i.S_m$ equals $r_j.S_m$ for $(0 \leq m \leq n-2)$. $r_i.S_{n-1}$ is adjacent to $r_j.S_{n-1}$, i.e., $Min(r_i.S_{n-1})$=$Max(r_j.S_{n-1}) + 1$ or $Min(r_j.S_{n-1})$=$Max(r_i.S_{n-1}) + 1$. When two rules are adjacent, they can be merged into one as $(F_0 \in S_0 \wedge \ldots F_i \in S_i \wedge \ldots F_{n-1} \in (r_i.S_{n-1} \cup r_j.S_{n-1}))$

Until now, two rules can be merged into one only if they are adjacent. In this paper, we propose to reduce rule set by replacing a group of rules with other smaller group even though they are not adjacent.

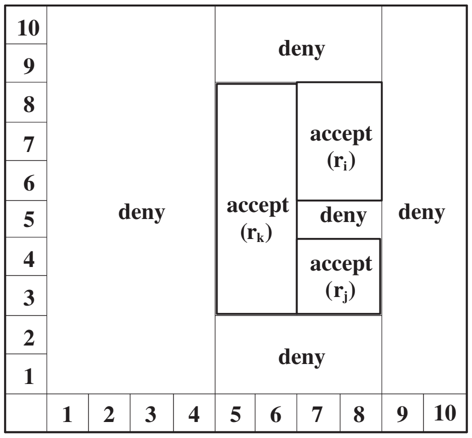
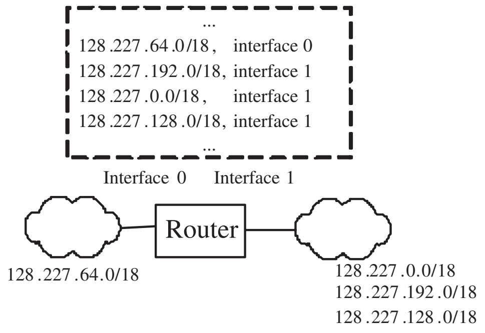


Fig. 1. Two dimensional representation for table 1 (A)



...
128.227.64.0/18, interface 0
128.227.192.0/18, interface 1
128.227.0.0/18, interface 1
128.227.128.0/18, interface 1
...
Interface 0
Interface 1
Router
128.227.64.0/18
128.227.0.0/18
128.227.192.0/18
128.227.128.0/18

Fig. 2. Substitutional optimization example for routing table

B. Basic Idea

The number of rules can be reduced by substituting a new group of rules for a given group. Fig. 2 shows an example in a routing table. The class B network of 128.227.0.0/16 is divided into four sub-networks: 128.227.0.0/18, 128.227.64.0/18, 128.227.128.0/18 and 128.227.192.0/18. To forward packets to appropriate interfaces, the router has four rules as shown in Fig. 2. However, an experienced router manager can make the routing table simpler by substituting a new group of two rules (128.227.64.0/18 interface0, 128.227.64.0/16 interface1) for the old four rules. In this paper, we propose to apply this basic substitutional optimization to firewall's rule set.

To optimize firewall's rule set, Gouda et al. propose five algorithms for designing the sequence of rules in a firewall to be consistent, complete, and compact [4]. These algorithms include $reduction$, $marking$, $generation$, $compactness$, and $simplification$. As in [4], we denote firewall's rule set as R. Table I (B) shows how R changes after applying these five algorithms to rule set (A). In this example, half the rules are eliminated while the semantics of the rule set remains same. Fig. 3 shows FDD for R in table I (B).

However, as shown in table I (C), the rule set can be reduced further, which can not be achievable by any previous work [3], [4], [7], [8]. This is because previous approaches can eliminate some rules only when they are redundant by other rules. To

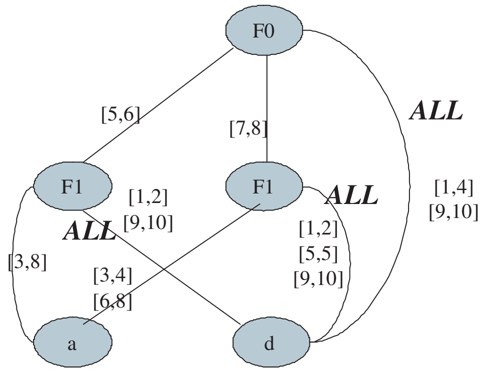


Fig. 3. FDD for table I (B)

the contrary, the rule set of table I (C) can be obtained by our proposal, $Substitutional\ Optimization\ (SO)$.

C. Algorithm

We propose a simple algorithm, *simple substitutional optimization (SSO)*. It can reduce the number of firewall's rule set aggressively when a certain condition is satisfied.

We assume that a rule set has already been converted into FDD [4] and SSO uses it as an input. Therefore, no rules are overlapped with each other when SSO starts.

We define replicable-group as a group of rules which can be replaced by other smaller group without changing the meaning of the rule set. For example, in Fig. 1, the rule set of $\{(F_0 \in [5,6] \wedge F_1 \in [3,8]\ a), (F_0 \in [7,8] \wedge F_1 \in [3,4]\ a), (F_0 \in [7,8] \wedge F_1 \in [6,8]\ a), (F_0 \in [7,8] \wedge F_1 \in [5,5]\ d)\}$ can be substituted by a group of two rules $\{(F_0 \in [7,8] \wedge F_1 \in [5,5]\ d), (F_0 \in [5,8] \wedge F_1 \in [3,8]\ a)\}$. In this case, the first three rules of the original rule set form a replicable-group: $\{(F_0 \in [5,6] \wedge F_1 \in [3,8]\ a), (F_0 \in [7,8] \wedge F_1 \in [3,4]\ a), (F_0 \in [7,8] \wedge F_1 \in [6,8]\ a)\}$. In general, a replicable-group is defined as a group of three rules (r_i, r_j, r_k) that has the following properties (we use the same notations of Fig. 1):

(1) $r_i.S_m = r_j.S_m = r_k.S_m, (0 \leq m \leq n-3)$
(2) $r_i.S_{n-2} = r_j.S_{n-2}, r_i.S_{n-2} \neq r_k.S_{n-2}$
(3) $r_i.S_{n-2}\ are\ adjacent\ to\ r_k.S_{n-2}$
(4) $r_i.action = r_j.action = r_k.action$
(5) $r_i.S_{n-1} \subset r_k.S_{n-1}, Max(r_i.S_{n-1}) = Max(r_k.S_{n-1})$
(6) $r_j.S_{n-1} \subset r_k.S_{n-1}, Min(r_j.S_{n-1}) = Min(r_k.S_{n-1})$

The first property means that S_m values should be same except two dimensions: $m = (n-2)$ and $(n-1)$. In Fig. 1, x and y-axes represent for dimension $(n-2)$ and $(n-1)$ respectively. The second property says that only two of the three rules have the same value for S_{n-2}. The other properties mean that r_i, r_j and r_k forms a bracket-like shape as in Fig. 1.

For a replicable-group of (r_i, r_j, r_k), we define an inscribed-rule r_{id} and an inscribing-rule r_{ig} as follows:

$$r_{id} = (F_0 \in S_0 \wedge \ldots F_{n-3} \in S_{n-3} \wedge F_{n-2} \in r_i.S_{n-2} \wedge F_{n-1} \in (r_k.S_{n-1} - r_i.S_{n-1} - r_j.S_{n-1})\ \neg r_i.action) \quad (2)$$

```
R={};
SSO(R) {
  Find $r_i$, $r_j$, and $r_k$ satisfying following conditions;
  (1) $r_i.S_m = r_j.S_m = r_k.S_m, (0 \leq m \leq n-3)$
  (2) $r_i.S_{n-2} = r_j.S_{n-2}, r_i.S_{n-2} \neq r_k.S_{n-2}$
  (3) $r_i.S_{n-2}$ are adjacent to $r_k.S_{n-2}$
  (4) $r_i.action = r_j.action = r_k.action$
  (5) $r_i.S_{n-1} \subset r_k.S_{n-1}, Max(r_i.S_{n-1}) = Max(r_k.S_{n-1})$
  (6) $r_j.S_{n-1} \subset r_k.S_{n-1}, Min(r_j.S_{n-1}) = Min(r_k.S_{n-1})$
  $R = R - \{r_i, r_j, r_k\} + \{r_{ig}\};$
  $R' = R' \cup \{r_{id}\};$
}
```

Fig. 4. SSO Algorithm

$$r_{ig} = (F_0 \in S_0 \wedge \ldots F_{n-3} \in S_{n-3} \wedge F_{n-2} \in (r_i.S_{n-2} \cup r_k.S_{n-2}) \wedge F_{n-1} \in r_k.S_{n-1}\ r_i.action) \quad (3)$$

where $\neg r_i.action$ is the opposite of r_i's action.

Geometrically, a replicable-group and r_{id} form a rectangle in which S_{n-2} and S_{n-1} are the ranges for the lower and the upper bases, and r_{id} is the inscribed rectangle. Every pair of a replicable-group and r_{id} can be replaced with a smaller rule set by the following theorem 1.

Theorem 1: A replicable-group consisting of r_i, r_j and r_k can be replaced with r_{id} and r_{ig} to decrease the number of rules for a given rule set.

$Proof$: If r_{id} is placed above r_{ig}, the semantics of the original rule set R remains same over the area of $(F_0 \in S_0 \wedge \ldots F_{n-3} \in S_{n-3} \wedge F_{n-2} \in (r_i.S_{n-2} \cup r_k.S_{n-2}) \wedge F_{n-1} \in (r_k.S_{n-1} \cup r_i.S_{n-1} \cup r_j.S_{n-1}))$. If R includes r_{id} explicitly, the number of rules will decrease to $|R| - 2$ after substitution. If not, it will be $|R| - 1$. In any case, substitution decreases the number of rules.

Fig. 4 shows the algorithm for SSO. Besides R, we use temporary rule set R' to maintain high priority rules separately. In fact, R' is a set of $r'_{id}s$, and will be merged with R at the end of the whole algorithm.

D. Example

For an example, we apply SSO to Fig. 1. A replicable-group of r_i, r_j and r_k is found as follows:

- $r_i = \{F_0 \in [7,8] \wedge F_1 \in [6,8]\ a\}$
- $r_j = \{F_0 \in [7,8] \wedge F_1 \in [3,4]\ a\}$
- $r_k = \{F_0 \in [5,6] \wedge F_1 \in [3,8]\ a\}$

By equation 2 and 3, r_{id} and r_{ig} become

- $r_{id} = \{F_0 \in [7,8] \wedge F_1 \in [5,5]\ d\}$
- $r_{ig} = \{F_0 \in [5,8] \wedge F_1 \in [3,8]\ a\}$

Table II shows R and R' after SSO has been applied to table I (A). The number of rules has been decreased by two.

TABLE II
AFTER SSO(R) IS EXECUTED

R	(1)F_0∈[1,4]∧F_1∈[1,10] d
	(2)F_0∈[5,8]∧F_1∈[1,2] d
	(3)F_0∈[5,8]∧F_1∈[3,8] a
	(4)F_0∈[5,8]∧F_1∈[9,10] d
	(5)F_0∈[9,10]∧F_1∈[1,10] d
R'	(1)F_0∈[7,8]∧F_1∈ [5,5] d

IV. COMPLEX SUBSTITUTIONAL OPTIMIZATION (CSO)

A. Basic Idea

We can combine SSO and the five algorithms proposed in [4], which we call *complex substitutional optimization (CSO)*. After a replicable-group in R is substituted by r_{id} and r_{ig}, some rules in the changed rule set may become adjacent to each other. This can happen because CSO can eliminate some rules that have prevented other rules from being merged together and abbreviated. If then, we can use again the ideas of [4], [7] to merge two adjacent rules so that the total size of rule set decreases further. In this way, CSO can reduce rule set dramatically when used together with the algorithms from [4].

B. Algorithm

SSO can be used together with the five algorithms from [4]: $reduction$, $marking$, $generation$, $compactness$, and $simplification$. More specifically, SSO is executed repetitively together with $reduction$. So, the input to SSO does not have "ALL" marking as in [4]. We also assume that no dimension is allowed to disappear until the end of the whole algorithm.

Fig. 5 shows CSO algorithm in which SSO is mixed up with the five algorithms from [4]. During the while loop, $reduction$ and SSO are repeated alternatively until no substitution happens. $Reduction$ should be executed before SSO is executed since $reduction$ can make adjacent rules merged into one so that SSO does not need to take care of it. For example, if rule (2) in table I (A) were two rules of $\{F_0 \in [5,6] \wedge F_1 \in [1,2]\ d\}$ and $\{F_0 \in [7,8] \wedge F_1 \in [1,2]\ d\}$, $reduction$ would put them together into $\{F_0 \in [5,8] \wedge F_1 \in [1,2]\ d\}$ before SSO is applied to.

Once the while loop exits, we can benefit the five algorithms from [4]. Readers who want to know details about $reduction$, $marking$, $generation$, $compactness$, and $simplification$ should refer [4].

At the end of the whole algorithm, we concatenate R to R'. All rules from R' precede any of R. In this way, R' has higher priority to R since we assumed that a first-matching rule has high priority when conflict happens.

C. Example

We show how efficiently CSO optimizes the firewall's rule set in table III. The original rule set has three dimensions each of which ranges from 1 to 10.

According to CSO algorithm in Fig. 5, $reduction$ is executed first of all, but it does not reduce any rule. Next,

```
CSO(R) {
    R'={};
    While (any change at SSO subroutine) {
        reduction(R);
        SSO(R);
    }
    reduction(R);
    marking(R);
    generation(R);
    compactness(R);
    simplification(R);
    R=merge(R,R');
}
```

Fig. 5. CSO Algorithm

SSO is applied to the rule set. After this first substitution, we obtain two rule sets, R and R', which decrease the size of rule set by four rules. This is shown in table IV. For example, rules (1), (4) and (5) in table III forms a replicable-group. In Table IV, they are replaced by rule (1) in R and rule (1) in R'. Each of them corresponds to r_{ig} and r_{id}. In this case, rule (7) in table III equals r_{id}, i.e., explicitly defined in the original rule set. Similarly, rules (9), (10), (12), and (14) in table III are replaced with rule (6) in R and rule (2) in R' in table IV.

Since there was a change in SSO, we reenter the while loop. After second $reduction$ in the while loop is applied, rules of table IV (R) are reduced to five rules in table V: three belongs to R and two belongs to R'. For example, rules (1) and (6) in table IV (R) are reduced to rule (1) in table V (R). Rules (2), (4), (7) and (9) in table IV (R) are reduced to rule (2) in table V (R). Similarly, rules (3), (5), (8) and (10) in table IV (R) are reduced to rule (3) in table V (R). R' remains same since this is used by only SSO.

Since no more change happens during the next SSO, we exit the while loop. From now, we can benefit the algorithms from [4], and just need to merge R and R' to make a new optimized rule set. Fig. 6 is FDD for R of table V (R) before $reduction$ is applied. After applying $reduction$, $marking$, $generation$, $compactness$ and $simplification$ to this FDD, we get the third and the fourth rules shown in table VI (A). After $merge$, the first and the second rules in table VI (A) are added up. The final rule set consists of only four rules as in the table VI (A).

Fig. 7 and table VI (B) show FDD and R when the algorithms from [4] are executed without SO. In conclusion, for this simple example, CSO reduces the size of rule set from 16 to 4, resulting in 75% reduction, while the state-of-the-art algorithm eliminated 6 rules, that is just 37.5% reduction.

V. Conclusion

In this paper, we proposed new algorithms for optimizing firewall's rule set: SSO and CSO. We observe that previous works could merge two rules into one only when they are adjacent. Our approach expands this one dimensional merger to two dimensional substitution. In CSO, we augment SSO

TABLE III
Original rule set

(1) $F_0 \in [1,5] \wedge F_1 \in [1,3] \wedge F2 \in [2,7]$ a
(2) $F_0 \in [1,5] \wedge F_1 \in [1,3] \wedge F2 \in [1,1]$ d
(3) $F_0 \in [1,5] \wedge F_1 \in [1,3] \wedge F2 \in [8,10]$ d
(4) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [5,7]$ a
(5) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [2,3]$ a
(6) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [1,1]$ d
(7) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [4,4]$ d
(8) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [8,10]$ d
(9) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [2,3]$ a
(10) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [5,7]$ a
(11) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [1,1]$ d
(12) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [4,4]$ d
(13) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [8,10]$ d
(14) $F_0 \in [6,10] \wedge F_1 \in [1,4] \wedge F2 \in [2,7]$ a
(15) $F_0 \in [6,10] \wedge F_1 \in [1,4] \wedge F2 \in [1,1]$ d
(16) $F_0 \in [6,10] \wedge F_1 \in [1,4] \wedge F2 \in [8,10]$ d

TABLE IV
rule set after the first SSO applied

R	(1) $F_0 \in [1,5] \wedge F_1 \in [1,10] \wedge F2 \in [2,7]$ a
	(2) $F_0 \in [1,5] \wedge F_1 \in [1,3] \wedge F2 \in [1,1]$ d
	(3) $F_0 \in [1,5] \wedge F_1 \in [1,3] \wedge F2 \in [8,10]$ d
	(4) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [1,1]$ d
	(5) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [8,10]$ d
	(6) $F_0 \in [6,10] \wedge F_1 \in [1,10] \wedge F2 \in [2,7]$ a
	(7) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [1,1]$ d
	(8) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [8,10]$ d
	(9) $F_0 \in [6,10] \wedge F_1 \in [1,4] \wedge F2 \in [1,1]$ d
	(10) $F_0 \in [6,10] \wedge F_1 \in [1,4] \wedge F2 \in [8,10]$ d
R'	(1) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [4,4]$ d
	(2) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [4,4]$ d

TABLE V
Rule set after while loop finished

R	(1) $F_0 \in [1,10] \wedge F_1 \in [1,10] \wedge F2 \in [2,7]$ a
	(2) $F_0 \in [1,10] \wedge F_1 \in [1,10] \wedge F2 \in [1,1]$ d
	(3) $F_0 \in [1,10] \wedge F_1 \in [1,10] \wedge F2 \in [8,10]$ d
R'	(1) $F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [4,4]$ d
	(2) $F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [4,4]$ d

TABLE VI
Final rule set and comparison with [4]

A. Final rule set obtained by CSO
(1)$F_0 \in [1,5] \wedge F_1 \in [4,10] \wedge F2 \in [4,4]$ d
(2)$F_0 \in [6,10] \wedge F_1 \in [5,10] \wedge F2 \in [4,4]$ d
(3)$F_0 \in [1,10] \wedge F_1 \in [1,10] \wedge F2 \in [2,7]$ a
(4)$F_0 \in ALL \wedge F_1 \in ALL \wedge F2 \in ALL$ d

B. Final rule set obtained by [4]
(1)$F_0 \in [1,5] \wedge F_1 \in [1,3] \wedge F2 \in [2,7]$ a
(2)$F_0 \in [1,5] \wedge F_1 \in [1,3] \wedge F2 \in ALL$ d
(3)$F_0 \in [1,5] \wedge F_1 \in ALL \wedge F2 \in [2,3]$ a
(4)$F_0 \in [1,5] \wedge F_1 \in ALL \wedge F2 \in [5,7]$ a
(5)$F_0 \in [1,5] \wedge F_1 \in ALL \wedge F2 \in ALL$ d
(6)$F_0 \in ALL \wedge F_1 \in [5,10] \wedge F2 \in [2,3]$ a
(7)$F_0 \in ALL \wedge F_1 \in [5,10] \wedge F2 \in [5,7]$ a
(8)$F_0 \in ALL \wedge F_1 \in [5,10] \wedge F2 \in ALL$ d
(9)$F_0 \in ALL \wedge F_1 \in ALL \wedge F2 \in [2,7]$ a
(10)$F_0 \in ALL \wedge F_1 \in ALL \wedge F2 \in ALL$ d

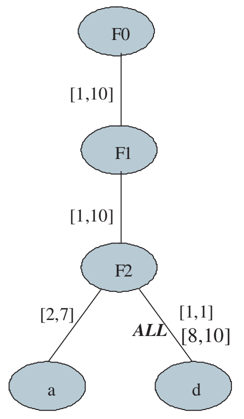


Fig. 6. FDD for table V (R) before $reduction$

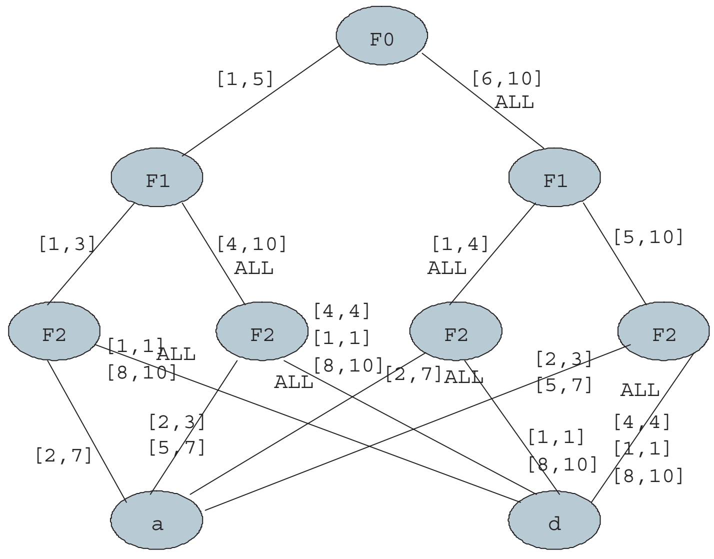


Fig. 7. FDD for table III before $reduction$

in such a way that SSO is used together with the previous work [4]. We showed that CSO can reduce rule set in a new manner, which is not achievable by any previous work.

The proposed algorithm is aggressive in reducing firewall's rule set since it can reduce the size of rule set by actively substituting a smaller group of rules for a larger existing group. This is an innovative approach since previous works depend on only adjacent relationship between two rules for reducing rule set. The proposed algorithm can also be applied to a general classification problem as well as firewall's rule set.

References

[1] A. Rubin, D. Geer, and M. Ranum, "Web Security Sourcebook," *Wiley Computer Publishing*, 1997.

[2] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer, vol. 37, no. 6, pp. 62-67*, June 2004.

[3] A. X. Liu and M. G. Gouda, "Removing Redundancy from Packet Classifiers," *Proc. of ACM SIGCOMM*, 2004.

[4] M. G. Gouda and A. X. Liu, "Firewall Design: Consistency, Completeness and Compactness," *Proc. of ICDCS-04, pp. 320-327*, March 2004.

[5] A. X. Liu and M. G. Gouda, "Diverse Firewall Design," *Proc. of IEEE International Conference on Dependable Systems and Networks (DSN-04), pp. 595-604*, June 2004.

[6] E. S. Al-Shaer and H. H. Hamed, "Discovery of Policy Anomalies in Distributed Firewalls," *Proc. of IEEE Infocom 2004*, March 2004.

[7] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," *Proc. of ACM SIGCOMM*, 1999.

[8] P. Gupta, "Algorithms for Routing Lookups and Packet Classification," *PhD Thesis, Stanford University*, 2000.

[9] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network, 15(2):, pp. 24-32*, March 2001.

[10] T. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," *Proc. of ACM SIGCOMM*, 1998.

[11] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: a novel firewall management toolkit," *ACM Transactions On Computer Systems, vol. 22, issue 4, pp. 381-420*, November 2004.

[12] J. D. Guttman, "Filtering Postures: Local Enforcement for Global Policies," *Proc. of IEEE Symposium on Security and Privacy*, 1997.

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

[14] A. D. Keromytis and J. Wright, "Transparent Network Security Policy Enforcement," *Proc. of USENIX 2000 Annual Technical Conference, pp. 215 - 226*, June 2000.

[15] A. Hari, S. Suri, and G. Parulkar, "Detecting and Resolving Packet Filter Conflicts," *Proc. of IEEE Infocom*, March 2000.

[16] V.Srinivasan, G.Varghese, S.Suri and M.Waldvogel, "Fast and Scalable Layer Four Switching," *Proc. of ACM SIGCOMM*, 1998.

[17] S.Hazelhurst, A.Attar and R.Sinnappan, "Algorithms for Improving the Dependability of Firewall and Filter Rule Lists," *Proc. of International Conference on Dependable Systems and Networks (DSN 2000)*, 2000.

[18] S. Ioannidis, A. D. Keromytis, S. Bellovin, and J. M. Smith, "Implementing a Distributed Firewall," *Proc. of ACM Computer and Communications Security (CCS) 2000, pp. 190 - 199*, 2000.

[19] R. N. Smith, Y. Chen, and S. Bhattacharya, "Cascade of Distributed and Cooperating Firewalls in a Secure Data Network," *IEEE Trans. On Knowledge and Data Engineering, Vol. 15, No. 5*, 2003.

[20] J. Wack, K. Cutler, and J. Pole, "Guidelines on Firewalls and Firewall Policy," *National Institute of Standards and Technology*, January 2002.

[21] A. Wool, "The use and usability of direction-based filtering in firewalls," *Computers & Security, 23(6), pp. 459 - 468*, 2004.

[22] E. W. Fulp, "Optimization of Network Firewall Policies Using Ordered Sets and Directed Acyclical Graphs," *Proc. of IEEE Internet Management Conference*, 2005.