This paper was presented as part of the 14th IEEE Global Internet Symposium (GI) 2011 at IEEE INFOCOM 2011

Fast and Scalable Method for Resolving Anomalies in Firewall Policies

Hassan Gobjuka
Verizon
919 Hidden Ridge
Irving, TX 75038
hasan.gobjuka@verizon.com

Kamal A. Ahmat
Department of Information Technology
City University of New York
New York, NY 11101
kamal.ahmat@live.lagcc.cuny.edu

Abstract

In this paper, we investigate the problem of improving the performance and scalability of large firewall policies that comprise thousands of rules by detecting and resolving any potential conflicts among them. We present a novel, highly scalable data structure that requires $O(n)$ space where n is the number of rules in the policy to represent the dependency among rules. After that, we describe a practical heuristic that utilizes our data structure to find conflicting rules, and consequently find an optimal ordering of consistent ones. Our algorithm has time complexity $O(n^2 \log n)$, making it the fastest to-date known algorithm for firewall rule anomaly discovery and resolution. We validate the practicality of our algorithm through real-life firewall policies and synthetic firewall policies of large data. Performance results show that our heuristic algorithm achieves from 40% to 87% improvement in the number of comparisons overhead, comparatively with the original policies.

I. Introduction

The proliferation and advancement of the network technologies have not only facilitated the work of legitimate organizations, but also the activity of criminals and other malefactors. The plenty of opportunities, provided by the World Wide Web, make the network attacks a daily routine. The nature of firewall makes it the only tool to effectively control the flow of traffic and detect suspicious behavior of the installed applications. Firewalls are the bastions, protecting the workstations and networks from the outside intrusion.

Unlike other security software, firewalls have no database of virus signatures, which makes them effective protection against the applications disguised as the legitimate ones. As a rule, firewalls have at least two modes, perhaps more, which either blocks the connection or selectively allows the traffic going in and out. When filtering the traffic firewall acts in accordance with the custom rules added to its configuration, which practically describe the packets to be blocked and the ones to be let in. The assembly of firewall rules is referred to as policy.

Firewall policies are mainly created on the individual basis, often the rules have different moments: some grant access for certain applications, others for IPs and protocols, third ones make accessible the certain ports, etc. Once written, the policy is experiencing a constant modification, which becomes painstaking with the course of time. The number of rules in a policy increases constantly which doesn't only raise serious scalability and performance issues, but also highly increases the probability of inconsistency among these rules [2], [12].

The anomalies have double effect: on the one hand, they can compromise organization for letting confidential information out, on the other hand, they expose the network to external attacks, the consequences of which are hard to predict [2], [5]. Despite the importance of automating the process of discovering rule conflicts, there are very few commercial platforms available on the market today that offer tools for discovering and eliminating anomalies in firewall policies.

In this paper, we investigate the problem of discovering the set of troublesome rules in a large firewall policy and consequently eliminating or resolving them so that all the rules in the policy are consistent and can be reordered to make them effectively and optimally functional. We first present a novel data structure that requires $O(n)$ space where n is the number of rules in the policy to represent the dependency among rules. After that, we describe a practical heuristic that utilizes our data structure to find conflicting rules, and consequently find an optimal ordering of consistent rules. Our algorithm has time complexity $O(n^2 \log n)$, making it not only very scalable, but also the fastest to-date known algorithm for firewall rule anomaly discovery and resolution. We validate the practicality of our algorithm through real-life firewall policies and synthetic firewall policies of large data and demonstrate that our algorithm achieves from 40% to 87% improvement in the number of comparisons overhead comparatively with the original policies.

A. Related Work

Analyzing firewall policies has been extensively studied in the research community. In this section, we focus our literature study on related research that is close to our work in the areas of firewall rule conflict detection and optimization analysis [1], [4], [6], [8] and [10].

In [6], the authors study Optimal Rule Ordering (ORO) in a firewall policy. They prove that the ORO problem is NP-hard. Then, they present a heuristic algorithm that utilizes Internet traffic characteristics to optimize the organization of firewall policies. Their algorithm takes the rule list to be optimized as an input and an optimization limit that represents an upper bound on the total weight of the active rules. The algorithm then sequentially selects rules based on the descending order of weights. Each selected rule along with all dependent rules are sorted and inserted in the sorted list while maintaining the integrity of the policy. However, this approach considers only partial set of rules that have higher probability, which limits its practicality.

Al-Shaer *et al.* [1] identified the anomalies that could exist in both single- and multi-firewall environments. They also presented several algorithms based on state diagrams to detect anomalies between each pair of rules in centralized and distributed firewalls. A tool named "Firewall Policy Advisor" was also implemented based on their methods. However, this work doesn't address the problem of reordering rules.

Fulp [4] proposed a new method named Simple Rule Sorting (SRS) for the optimal ordering of independent neighboring firewall rules based on the non-increasing probabilities. Their method is based on Direct Acyclic Graph (DAG). They also showed that their proposed methods yield in considerably better performance comparing to unordered policies. However, this paper considers independent rules and thus, it can be more applicable to firewalls that handle smaller set of rules.

An all-match based method for discovering and eliminating identical redundant rules was described in [8]. The authors also described a formal proof that their method removes all such redundant rules. The work presented in [8] closes, generally, the problem of discovering redundant rules.

Results presented here are most closely associated with the techniques developed in [10]. However, there are important differences between their and our methods. First, the algorithm presented in [10] has time complexity $O(n^3)$ while the complexity of our algorithm is $O(n^2 \log n)$. Second, our algorithm also returns a set of correlating rules.

B. Organization of the Paper

The rest of the paper is organized as follows. The next section describes the system model used in this paper and problem formulation. We subdivide Section III into two subsections. First, in Subsection III-A we describe a data structure that we designed to model the rule dependencies. Then, in Subsection III-B we present an algorithm to find the correct ordering of rules and to discover and eliminate conflicting ones. Our environment setup and experimental results are described in Section IV. Finally, Section V concludes the paper.

II. System Model and Problem Formulation

In this section, we describe the firewall policy model we adopt and present formal description of the problem addressed in this paper. Then, we formally present the problem addressed in this paper.

A. Firewall Rules and Policies

Firewalls control traffic by matching incoming packets against a set of rules grouped in policy files. Each rule defines the parameters against which each connection is compared, resulting in a decision on what action to take for each connection. As soon as a network packet matches a rule, that rule is applied, and processing stops [11], [12].

Definition 2.1: [FIREWALL-RULE]
A firewall rule r is an ordered set of seven attributes r = (Protocol, source IP, source port, destination IP, destination port, action, probability). □

While the first six attributes are self-descriptive, the probability of each rule indicates the ratio of hits of that rule against incoming packets. To improve the performance and scalability in firewalls that handle large number of incoming packets and maintain very large policies, rules with higher probability are usually listed on top of the policy so that the number of comparisons is kept minimum.

Definition 2.2: [FIREWALL-POLICY]
A firewall policy P is an ordered list of rules $P = (r_1, r_2, \ldots, r_n)$. □

The two major anomalies in firewall rules are *Shadowing* and *Correlating* [1]. A rule is shadowed by another rule when a preceding rule matches all the packets that match the succeeding rule, and thus the succeeding rule is never activated [1]. Consequently, there is an implied precedence relationship - also known as rule dependency - between rules to prevent the occurrence

Id	Protocol	Source		Destination		Action	Probability
		IP address	Port	IP address	Port		
r_1	TCP	71.123.10.*	any	10.0.0.1	21	permit	0.3
r_2	TCP	*.*.*.*	any	10.0.0.1	21-23	deny	0.25
r_3	TCP	71.*.*.*	any	10.0.0.1	21	permit	0.15
r_4	TCP	71.123.*.*	any	10.0.0.1	21	deny	0.1
r_5	TCP	*.*.*.*	any	10.0.0.1	23-25	permit	0.1

TABLE I
EXAMPLE SECURITY POLICY CONSISTING OF MULTIPLE ORDERED RULES.

of shadowing [4]. If rule r_j shadows rule r_i (denoted $r_i \subset r_j$), then we use the notation $r_i \prec r_j$ to denote the implied precedence of rule r_i to rule r_j. Ordering rules based on their dependencies may conflict with the goal of minimizing the number of comparisons [4]. It is worth to note here that the shadowing relationship is *transitive*. That is, if rule $r_k \subset r_j$ and $r_j \subset r_i$, then $r_k \subset r_i$. For example, consider the list of rules in Table I[1]. It can be seen that rule r_2 shadows rule r_3 and rule r_3 shadows rule r_4. Consequently, rule r_2 shadows rule r_4 ($r_4 \subset r_2$).

Two rules are correlated if some packets match both of the rules but none of the rules shadows the other, and the action taken by these rules is different [1]. If two rules are correlated, then one of them must be removed from the policy or edited. We use the notion $r_i \bigcap r_j$ to indicate the correlation between rules r_i and r_j. Note that if $r_i \bigcap r_j$, then both $r_i \prec r_j$ and $r_j \prec r_i$ hold. For example, rules r_2 and r_5 from Table I are correlated.

In an optimal firewall policy, neither any pair of rules correlate nor shadow each other. Consequently, we define a *valid policy* as follows.

Definition 2.3: [VALID-POLICY]
A policy $P = (r_1, r_2, r_3, \ldots, r_n)$ is valid if and only if, for any two rules $\{r_i, r_j\} \in P$:

1) Rule r_i doesn't shadow rule r_j, and vice versa; and
2) Rules r_i and r_j do not correlate.

□

B. Problem Formulation

Given a policy comprising a set of rules, our goal is to discover and eliminate troublesome rules and find an ordered list of consistent ones while performing the minimum number of comparisons. Formally, the AUTOMATIC-ANOMALY-RESOLVING problem is defined as follows.

[1]To simplify the presentation, we don't include the default rule. However, that doesn't affect the results presented in this paper.

Definition 2.4: [AUTOMATIC-ANOMALY-RESOLVING]
Given a policy that comprises a list of rules $P = (r_1, r_2, \ldots, r_n)$, construct a policy $\grave{P} \subseteq P$ such that $\grave{P}$ is valid. □

III. Policy Conflict Detection Algorithm

In this section we first describe the data structure that we use to represent dependencies among policy rules. Then, we present our algorithm Policy-Anomaly-Discovery that takes a policy and utilizes the dependency data structure to find and eliminate anomalies returning a list of validated policy.

A. Data Structure

The data structures that are used by the Policy-Anomaly-Discovery algorithm are a directed graph D and an undirected graph U. In the undirected graph U, each rule is represented as a node and each link represents the presence of dependency between connected nodes (i.e. rules.) That is, if U doesn't contain any links, there is no dependency between any pair of rules, and consequently, rules can be arbitrarily ordered. The forehead mentioned situation is very unlikely in practice but we stated it to clarify the context of the graph. The directed graph D is used to represent the actual dependency among rules. For each pair of rules r_i and r_j, if $r_i \prec r_j$, then there is node $(j, i) \in D$. Furthermore, for any two nodes (i, j) and (j, k) in graph D, there is link $((i, j), (j, k)) \in D$. Graphs U and D are formally described as follows.

- $D = (\grave{V}, A)$ where for each $r_k \prec r_j$ and $r_j \prec r_i$, $\{r_i, r_j, r_k\} \in P$, nodes (r_i, r_j) and (r_j, r_k) are in $\grave{V}$ and edge $(r_i, r_j) \rightarrow (r_j, r_k)$ is in A.
- $U = (V, E)$ with the vertex set $V = \{1, 2, \ldots, n\}$ and where, for each rule node $(r_i, r_j) \in D$, nodes r_i and r_j are in V, and edge (r_i, r_j) is in E.

Since the dependency relationship is transitive, if graph D contains edge $(r_i, r_j) \rightarrow (r_j, r_k)$, then the dependency of rule r_i on rule r_k is represented by that edge and consequently, node (r_i, r_k) can be removed from D. Thus, in such cases, edge (i, k) is not included in graph U. We call nodes that has outdegree 0 in D as $terminal$ nodes, and all other nodes as $non-terminal$ nodes.

To illustrate the structures of U and D, consider again the rules listed in Table I. It can be easily seen that the set R of rule dependencies is as follows $R = \{(r_1 \prec r_2)$, $(r_1 \prec r_3)$, $(r_1 \prec r_4)$, $(r_1 \prec r_5)$, $(r_4 \prec r_2)$, $(r_4 \prec r_3)$, $(r_4 \prec r_5)$, $(r_3 \prec r_2)$, $(r_3 \prec r_5)$, $(r_2 \prec r_5)$, $(r_5 \prec r_2)\}$. The graphs D and U are shown in Figure 3(a). Observe that, node $(2,1) \notin D$ and also edge $(1,2) \notin U$. This is due to the transitivity relationship between rules r_1 and r_2 (e.g. $r_1 \prec r_4 \prec r_2$). Excluding transitive dependencies simplifies the graph but it doesn't affect the anomaly discovery process.

During the anomaly detection and resolution course, we use the term "Red Nodes" to refer to the terminal nodes in graph D. Intuitively, we use the term "Red Edges" to refer to links from graph U that correspond to red nodes from D. Every non-red-edge from U is a "Black Edge". A "Black Path" is a set of consequent black edges. A Connected Component CC is a subset $\grave{U} \subset U$ such that every two nodes i and j from $\grave{U}$ are connected by a "Black Path". For instance, in the graph U in Fig. 3(b) there are two connected components; node 1 is a connected component, and nodes 2, 3, 4, and 5 form another connected component.

B. Anomaly Detection Algorithm

1) The Discover-Connected-Components *Procedure:* The Discover-Connected-Components procedure receives as an input an instance of graph U. It then systematically finds a red link (u, v) in U, assigns nodes u and v to connected components C_i and C_j, respectively. It then recursively traverses the nodes that are adjacent to both u and v and add them to C_i and C_j, respectively, until the traversing process ends. At this stage, either C_i and C_j are disjoint, which means at least one new connected component has been created, or C_i and C_j are in the same connected component. If C_i and C_j are disjoint connected components, then the red links are deleted from U and the set $\mathcal{C}$ of connected components is updated. The formal description of the Discover-Connected-Components procedure is given in Fig. 1.

2) The Policy-Anomaly-Discovery *Algorithm:* The algorithm Policy-Anomaly-Discovery starts by constructing graphs D and U as described in Section III-A. After that, it iteratively finds terminal nodes in D and colors them along with their corresponding edges in U in red. Then, it calls the Discover-Connected-Components procedure to find whether any new connected components are created. If there is a newly

```
Algorithm Discover-Connected-Components($U$)
Input: An undirected graph $U$
Output: Set of connected components $\mathcal{C} \in U$
(* Returns connected components in $U$ *)
1.  set $\mathcal{C} \leftarrow \phi$;
2.  for every red link $(i, j) \in U$
3.          set $C_i \leftarrow \{i\}$;
4.          set $C_j \leftarrow \{j\}$;
5.          for every node $k \in U$
6.                  if there is a black path $p(i, k)$
7.                          set $C_i \leftarrow C_i \bigcup \{k\}$;
8.                  if there is a black path $p(j, k)$
9.                          set $C_j \leftarrow C_j \bigcup \{k\}$;
10.         if $C_i = \{i\}$
11.                 set $\mathcal{C} \leftarrow \mathcal{C} \bigcup C_i$;
12.         if $C_j = \{j\}$
13.                 set $\mathcal{C} \leftarrow \mathcal{C} \bigcup C_j$;
14. delete red links from $U$;
15. return $\mathcal{C}$;
```

Fig. 1. A formal description of the Discover-Connected-Components procedure.

created connected component C_i such that C_i consists of a single node u, then the rule corresponds to node u is appended to end of validated policy $\grave{P}$. If more than one connected component is created such that each of them comprises one node, then all these nodes are appended arbitrarily to the policy $\grave{P}$. After that, red nodes (along with their incident links) and red links are deleted from graphs D and U, respectively. This process is repeated until D is empty and U has no edges, or until D doesn't contain any terminal nodes and the validated policy $\grave{P}$ is returned. If the graph D is not empty but it doesn't contain any terminal nodes, then graph D contains a set of correlating rules and the network manager must take an action and edit some of them to eliminate the anomaly. The formal description of algorithm Policy-Anomaly-Discovery is given in Fig. 2.

To derive the time complexity of the Policy-Anomaly-Discovery algorithm we first observe that the construction of the data structures U and D is proportional to the number of rules in the policy (i.e. $|P|$). To derive the complexity of Discover-Connected-Components procedure, we observe that each non-separable edge e is connected to vertices in the same component. Since each new component is smaller than previous ones, each vertex v is in a smaller connected component for at most log n times. Thus, each edge e is considered for at most 2log n times. Consequently, the total cost of considering all non-separable edges is $O(n^2 \log n)$ such that n is the total number of rules.

Example 3.1: Consider the policy P that contains the set of five rules defined in Table I. That is, $P = \{r_1$,

```
Algorithm Policy-Anomaly-Discovery($P$)
Input: A set $P$ of rules $r_1$, $r_2$, ..., $r_n$
Output: A set $\grave{P}$ of ordered rules, and $P$ of correlated
    rules
(* Orders rules according to their dependencies *)
1.  construct graphs $U$ and $D$;
2.  set $\grave{P} \leftarrow \varnothing$;
3.  while $D$ contains terminal nodes
4.      for every terminal node $v$=($i$, $j$) $\in D$
5.          do color $v$ red;
6.          do color link $e$=($i$, $j$) $\in U$ red;
7.      $\mathcal{C} \leftarrow$ Discover-Connected-Components($U$);
8.      if $\mathcal{C}$ doesn't contain new components
9.            return "Rules conflict: ", $P$;
10.     while $\mathcal{C} \neq \varnothing$
11.           select node $u$ randomly from $\mathcal{C}$;
12.           set $\grave{P} \leftarrow \grave{P} \cup \{r_u\}$;
13.           set $P \leftarrow P \setminus \{r_u\}$;
14. return $\grave{P}$;
```

Fig. 2. A formal description of the Policy-Anomaly-Discovery algorithm.

r_2, r_3, r_4, $r_5\}$. The algorithm will first construct the initial graphs D and U as depicted in Figure 3(a). Nodes (3, 1) and (4, 1) are terminals in D. Consequently, they are colored along with their corresponding links in U in red. The set $\mathcal{C}$ of connected components will be $\mathcal{C}$ = $\{C_1, C_2\}$ such that C_1 = $\{1\}$ and C_2 = $\{2, 3, 4, 5\}$. Since C_1 is a newly formed component, the procedure Discover-Connected-Components will traverse the graph U and find that the component C_1 contains one node (i.e. 1). Thus, the rule r_1 will be added as the first rule to the validated policy $\grave{P}$. Then, nodes (3, 1) and (4, 1) and their corresponding links will be deleted from D and U, prospectively. At this stage, graph D has three terminal nodes (2, 4), (3, 4), and (5, 4). Consequently, these nodes will be colored in red along with their corresponding links in U as shown in Fig. 3(b). Since node 4 comprises a new connected component in U, it will be appended to the list $\grave{P}$ and all red nodes and links will be deleted from D and U, respectively. After that, graph D contains two terminal nodes (2, 3) and (5, 3) so the algorithm will color these nodes along with edges (2, 3) and (5, 3) in U in red and append rule 3 to $\grave{P}$ as depicted in Fig. 3(c). Now, there is no terminal node in D and graph U is not empty (Fig. 3(d)), which indicates the presence of correlation anomaly between rules 2 and 5. Thus, the algorithm will return partial list of validated rules $\grave{P}$ = $\{r_1, r_4, r_3\}$, and set of correlating rules P = $\{r_2, r_5\}$. □

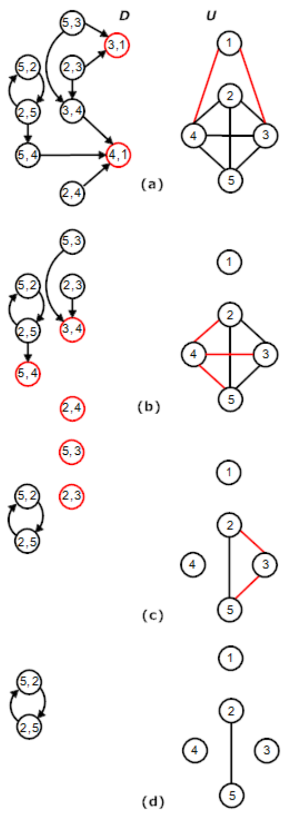

Fig. 3. The data structures used in Example 3, (a) The initial graphs D and U, (b) and (c) The data structures after the first and second iterations, respectively, and (d) The correlation relationship between rules 2 and 5.

IV. Implementation and Experimental Results

We implemented the algorithm described in this paper in the context of Verizon Internet Security Suite's Firewall system using platform-independent C++. In this section, we present our evaluation study of the scalability and performance of our method. Given any firewall policy, our algorithm builds the data structure presented in this paper and computes the average dependency and correlation values of each rule for analysis purposes. To evaluate the performance of this algorithm, we conducted two sets of experiential tests. In the first set, we obtained measurements of original policies for thirty-day period on five Verizon Internet Security Suite's firewalls and then ran our algorithm on the same firewalls and obtained measurements for thirty days and calculated average performance results for both sets. In the second set, we conducted further stress tests on synthetic firewall policies that have very large number of rules. During the experiments, the correlating rules returned by our algorithm were further sorted in non-

Test	No. of Rules	Avg. Base Comp.	Avg. Correlation	Avg. Dependency	Imp. Ratio
1	107	43.1	2.7	8.5	63.3%
2	361	87.2	1.4	2.2	47.2%
3	647	381.1	3.1	7.9	62.7%
4	881	341.6	3.3	6.4	71.2%
5	1385	715.3	3.8	6.7	74.8%

TABLE II
PERFORMANCE RESULTS FOR REAL-LIFE POLICIES.

Test	No. of Rules	Avg. Base Comp.	Avg. Correlation	Avg. Dependency	Imp. Ratio
1	10K	4224	121.3	13.5	68.6 %
2	12.5K	5584	389.6	11.6	40.5 %
3	15K	8054	274.7	12.0	76.4 %
4	25.5K	14263	649.2	15.2	79.3 %
5	30K	17714	712.4	20.7	87.6 %

TABLE III
PERFORMANCE RESULTS FOR SYNTHETIC POLICIES.

increasing order according to their probabilities.

The implementation of our data structure is based on the heap structure implementation described in [7], which has $O(n)$ space complexity, with substantial modifications. The details are omitted due to space constraints and will appear in the full version of the paper.

To test the performance of our methods on real-life policies, we deployed our implementation component on five firewalls that were installed on distinct networks. The five firewall policies consisted of 107, 361 , 647, 881, and 1316 rules, respectively. For the stress test, we ran five test sets on very large policies. For each test set, we generated the 50 policies of the same size using Rovniagin and Wools [9] model of generating firewall synthetic rules. Then, we submitted the policy files to our component and calculated the average for each test set. The performance results for both real real-life policies and synthetic policies are shown in Tables II and III, respectively. Obviously, the average comparison numbers depend on the traffic type and the complexity of rules. We observed that our method improves the performance as the policy size increases as well as when the average number of rule dependency and correlation increase. It can be also clearly seen that the performance ratio of our method is not affected by the increasing number of anomalies, which validates the high scalability and efficiency of our approach.

In general, we have discovered that our algorithm is sufficiently fast and highly scalable for all practical purposes. As can be seen in our results, the performance improvement ratio was quite high and our method helped in discovering many anomalies that were undiscovered by network administrators in real-life policies.

V. Conclusion

Firewall rule anomaly discovery has been an active research area in the past several years. However, the increasing complexity of enterprise network applications and services dictates a manifested need to find new directions and methods to improve the performance, reliability and scalability of firewalls. This paper presents a fast and highly scalable approach for discovering anomalies in firewall policies and resolving them. We have implemented our method and validated its practicality on both real-life as well as synthetically generated firewall policies of very large sizes. The experimental results support our theoretical analysis and show that our proposed method doesn't only scale very well but also improves the performance of firewall performance significantly.

References

[1] E. Al-Shaer and H. Hamed, *Firewall Policy Advisor for Anomaly Discovery and Rule Editing*, IFIP/IEEE Eighth International Symposium on Integrated Network Management, 2003, pp. 17-30.

[2] C. Benecke, *A Parallel Packet Screen for High Speed Networks*, In Proc. Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC '99), 1999.

[3] F. Chen, A. X. Liu , J. Hwang , T. Xie, *First Step Towards Automatic Correction of Firewall Policy Faults*, In Proceedings of the 24th USENIX Large Installation System Administration Conference (LISA), 2010.

[4] E. W. Fulp, *Optimization of Network Firewall Policies using Directed Acyclical Graphs*, In Proc. IEEE Internet Management Conference (IM '05), 2005.

[5] S Goddard, *An Unavailability Analysis of Firewall Sandwich Configurations*, In Proc. IEEE International Symposium on High Assurance Systems Engineering, 2001.

[6] H. Hamed and E. Al-Shaer, *On autonomic optimization of firewall policy organization*, J. High Speed Networks, Vol. 15, Issue 3, 2006, pp. 209-227.

[7] C. Lattner, V. Adve , *Data Structure Analysis: A Fast and Scalable Context-Sensitive Heap Analysis*, Tech. Report UIUCDCS-R-2003-2340, Computer Science Dept., Univ. of Illinois at Urbana-Champaign, 2003.

[8] A. Liu, C. R. Meiners, Y. Zhou, *All-Match Based Complete Redundancy Removal for Packet Classifiers in TCAMs*, In Proc. IEEE INFOCOM, 2008, pp. 574-582.

[9] D. Rovniagin, A. Wool, *The geometric efficient matching algorithm for firewalls*, In Proc. of IEEE Convention of Electrical and Electronics Engineers in Israel (IEEEI), 2004, pp. 153156.

[10] A. Tapdiya, E. W. Fulp, *Towards Optimal Firewall Rule Ordering Utilizing Directed Acyclical Graphs*, In Proc. IEEE ICCCN, 2009.

[11] R. L. Ziegler, *Linux Firewalls*, New Riders, Second Edition, 2002.

[12] E. D. Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls* , O'Reilly Media, Second Edition, 2000.