This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2008 proceedings.

Formal Verification of Firewall Policies

Alex X. Liu
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824-1266, U.S.A.
alexliu@cse.msu.edu

***Abstract*—Firewalls are the mainstay of enterprise security and the most widely adopted technology for protecting private networks. The quality of protection provided by a firewall directly depends on the quality of its policy (i.e., configuration). Due to the lack of tools for verifying firewall policies, most firewalls on the Internet have been plagued with policy errors. A firewall policy error either creates security holes that will allow malicious traffic to sneak into a private network or blocks legitimate traffic and disrupts normal business processes, which in turn could lead to irreparable, if not tragic, consequences.**

We propose a firewall verification tool in this paper. Our tool takes as input a firewall policy and a given property, then outputs whether the policy satisfies the property. Despite of the importance of verifying firewall policies, this problem has not been explored in previous work. Due to the complex nature of firewall policies, designing algorithms for such a verification tool is challenging. In this paper, we designed and implemented a verification algorithm using decision diagrams, and tested it on both real-life firewall policies and synthetic firewall policies of large sizes. The experimental results show that our algorithm is very efficient. In practice, our firewal verification algorithm can be used in the iterative process of firewall policy design, verification, and maintenance. Note that the firewall policy verification algorithm proposed in this paper is not limited to firewalls. Rather, they can be potentially applied to other rule-based systems as well.

Keywords: Firewall Policy, Firewalls, Network Security.

I. Introduction

Serving as the first line of defense against malicious attacks and unauthorized traffic, firewalls are cornerstones of network security and have been widely deployed in businesses and institutions. A firewall is placed at the point of entry between a private network and the outside Internet such that all incoming and outgoing packets have to pass through it. The function of a firewall is to examine every incoming or outgoing packet and decide whether to accept or discard it. This function is specified by a sequence (i.e., an ordered list) of rules, which is called the "policy", *i.e.*, the configuration, of the firewall. Each rule in a firewall policy is of the form $\langle predicate \rangle \rightarrow \langle decision \rangle$. The $\langle predicate \rangle$ of a rule is a boolean expression over some packet fields such as source IP address, destination IP address, source port number, destination port number, and protocol type. The $\langle decision \rangle$ of a rule can be *accept*, *discard*, or a combination of these decisions with other options such as a logging option. The rules in a firewall policy often conflict. To resolve such conflicts, the decision for each packet is the decision of the first (i.e., highest priority) rule that the packet matches. Table I shows an example firewall. Note that we use "a" to represent "accept" and "d" to represent "discard".

Rule	Src IP	Dest. IP	Src. Port	Dest. Port	Protocol	Action
r_1	*	192.168.0.1	*	25	TCP	a
r_2	1.2.*.*	*	*	*	*	d
r_3	*	*	*	*	*	a

TABLE I
AN EXAMPLE FIREWALL

A. Motivations

In this paper, we consider the following problem that is often raised in practice: given a firewall policy and a property, how can an administrator verify that the firewall policy satisfies the property? In this context, a property is a high-level policy that a firewall needs to enforce. An example property could be "the people in the development zone should not be able to access the database server in the accounting zone". Such a tool is helpful for firewall administrators to analyze and debug their firewalls as well as other routine duties such as demonstrating to their manager that the firewall does satisfy a set of necessary properties. This tool also can serve as a debugging tool in designing and analyzing firewall policies.

Due to the subtle and elusive nature of firewall rules, correctly verifying whether a firewall satisfies a property is by no means easy. First, the rules in a firewall policy are logically entangled because of conflicts among rules and the resulting order sensitivity. Second, a firewall policy may consist of a large number of rules. A firewall on the Internet may consist of hundreds or even a few thousand rules in extreme cases. Third, an enterprise firewall policy often consists of legacy rules that are written by different administrators, at different times, and for different reasons, which makes verifying firewall policies even more difficult. Verifying a large and complex sequence of logically related rules is certainly beyond human capability. Last but not least, an administrator can easily be deceived to believe that a property is satisfied by some rules in the middle of a firewall policy that seemingly implement the property.

Effective methods and tools for verifying firewall policies, therefore, are crucial to the success of firewalls. However, firewall administrators are woefully under-assisted due to the lack of firewall policy verification tools. Quantitative studies have shown that most firewalls on the Internet are plagued with policy errors [31]. Firewall policy errors can be dangerous and costly. On one hand, if a firewall policy error permits illegitimate communication, outside attackers may use these

978-1-4244-2075-9/08/$25.00 ©2008 IEEE

security holes to launch attacks. On the other hand, if a firewall policy error disallows legitimate communication, it may cause significant loss due to interrupted businesses. For example, if a firewall policy error prevents the communication between a web server and its supporting database server, all transactions that need such communication are interrupted.

B. Key Contributions

Despite of the importance of verifying firewall policies, this problem has not been explored in previous work. In this paper, we propose an efficient algorithm for verifying firewall policies. To our best knowledge, this paper represents the first formal study of firewall policy verification. The input of our verification algorithm includes a firewall policy and a given property, and the output is whether the policy satisfies the property. This provides firewall administrators a basis for how to fix the policy.

C. Road Map

The rest of this paper proceeds as follows. In Section II, we present our firewall verification algorithm. In Section III, we show our experimental results. In Section IV, we review previous related work. In Section V, we give concluding remarks. Because the focus of this paper is on security policies, we simply use the term "firewall" to mean "firewall policy", "firewall rule set", or "firewall configuration" unless otherwise specified.

II. Firewall Verification

In this section, we present a method for verifying firewall properties.

A. Property Representation

To verify whether a firewall satisfies a given property, we need to translate the property to a set of non-overlapping rules, which we call *property rules* in this context to distinguish them from firewall rules. For example, the property of accepting all web traffic to and from the web server except the web traffic to and from the malicious domain $1.2.*.*$ can be converted to the two property rules in Table II. Note that we use $!1.2.*.*$ to denote the set of all IP addresses that are not in the domain $1.2.*.*$. How to convert high-level descriptions of a property to a set of property rules is out of the scope of this paper. We assume that property rules are available for verification purposes.

Src IP	Dest. IP	Src Port	Dest. Port	Protocol	Action
!1.2.*.*	192.168.0.2	*	80	TCP	accept
192.168.0.2	!1.2.*.*	*	80	TCP	accept

TABLE II
Example property rules

Given a firewall and a set of property rules, the verification is successful if and only if every property rule is satisfied by the firewall. Next, we focus on how to verify whether a firewall satisfies one property rule.

B. Verification of Non-overlapping Firewalls

To make our firewall verification algorithm easy to understand, we first assume that the given firewalls are non-overlapping. A firewall is non-overlapping if and only if no rules in the firewall overlap. Two rules overlap if and only if there exists at least one packet that can match both rules. Note that real-life firewalls are most likely overlapping ones. The above unrealistic assumption is only for the purpose of introducing our verification algorithm that does not require this assumption.

Figure 1 shows an example non-overlapping firewall. In this firewall, for simplicity, we assume each packet has only two fields, F_1 and F_2, and the domain of each field is [1,100].

$$
\begin{array}{lllll}
R_1: & F_1 \in [20, & 50] & \wedge\ F_2 \in [20, & 70] & \rightarrow accept \\
R_2: & F_1 \in [20, & 50] & \wedge\ F_2 \in [1, & 19] & \rightarrow accept \\
R_3: & F_1 \in [20, & 50] & \wedge\ F_2 \in [71, & 100] & \rightarrow discard \\
R_4: & F_1 \in [1, & 19] & \wedge\ F_2 \in [1, & 39] & \rightarrow accept \\
R_5: & F_1 \in [51, & 60] & \wedge\ F_2 \in [1, & 39] & \rightarrow accept \\
R_6: & F_1 \in [1, & 19] & \wedge\ F_2 \in [40, & 100] & \rightarrow discard \\
R_7: & F_1 \in [51, & 60] & \wedge\ F_2 \in [1, & 100] & \rightarrow discard \\
R_8: & F_1 \in [61, & 100] & \wedge\ F_2 \in [1, & 100] & \rightarrow discard
\end{array}
$$

Fig. 1. A non-overlapping firewall

Suppose that we want to verify whether this firewall satisfies the following property rule:

$$F_1 \in [30,\ 40] \wedge F_2 \in [80,\ 90] \rightarrow discard$$

Comparing this property rule with each rule in the firewall, we can find that this property rule does not conflict with any rule. Two rules conflict if and only if they overlap and they have different decisions. Therefore, this property rule is satisfied by the firewall.

As another example, suppose that we want to verify whether this firewall satisfies the following property rule:

$$F_1 \in [1,\ 100] \wedge F_2 \in [20,\ 30] \rightarrow discard$$

Comparing this property rule with each rule in the firewall, we can find that this property rule conflicts with both rule R_4 and R_5. Therefore, this property rule is not satisfied, and rule R_4 and R_5 are the cause of the failure.

The algorithm for verifying non-overlapping firewalls can be directly derived from Theorem 1.

Theorem 1: A non-overlapping firewall satisfies a given property rule if and only if the property rule does not conflict with any rule in the firewall.

C. Verification of Generic Firewalls

Here we consider the verification of generic firewalls, where the given firewall can be either overlapping or non-overlapping. A firewall is called an overlapping firewall if and only if there are at least two rules in the firewall which are overlapping. Figure 2 shows an example overlapping firewall.

$$
\begin{array}{lllll}
r_1: & F_1 \in [20, & 50] & \wedge F_2 \in [20, & 70] & \rightarrow accept \\
r_2: & F_1 \in [1, & 60] & \wedge F_2 \in [40, & 100] & \rightarrow discard \\
r_3: & F_1 \in [1, & 100] & \wedge F_2 \in [1, & 100] & \rightarrow accept
\end{array}
$$

Fig. 2. An overlapping firewall

To verify whether a given firewall satisfies a property rule, one solution is to first convert the firewall to an equivalent non-overlapping firewall, then conduct the verification on the non-overlapping firewall. For example, we can convert the firewall in Figure 2 to an equivalent non-overlapping firewall as shown in Figure 1. Two firewalls are equivalent if and only if for any packet the two firewalls have the same decision.

A better solution is to convert the given firewall to an equivalent *firewall decision diagram*, then conduct the verification on the firewall decision diagram. A firewall decision diagram is a compact and conflict-free representation of firewalls [14]. A Firewall Decision Diagram (FDD) with a decision set DS and over fields $F_1, \cdots, F_d$ is an acyclic and directed graph that has the following five properties:

1) There is exactly one node that has no incoming edges. This node is called the *root*. The nodes that have no outgoing edges are called *terminal* nodes.
2) Each node v has a label, denoted $F(v)$, such that

$$F(v) \in \begin{cases} \{F_1, \cdots, F_d\} & \text{if } v \text{ is a nonterminal node,} \\ DS & \text{if } v \text{ is a terminal node.} \end{cases}$$

3) Each edge $e{:}u \rightarrow v$ is labeled with a nonempty set of integers, denoted $I(e)$, where $I(e)$ is a subset of the domain of u's label (*i.e.*, $I(e) \subseteq D(F(u))$).
4) A directed path from the root to a terminal node is called a *decision path*. No two nodes on a decision path have the same label.
5) The set of all outgoing edges of a node v, denoted $E(v)$, satisfies the following two conditions:
   a) *Consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$.
   b) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$. □

For example, we can convert the firewall in Figure 2 to a firewall decision diagram as shown in Figure 3.

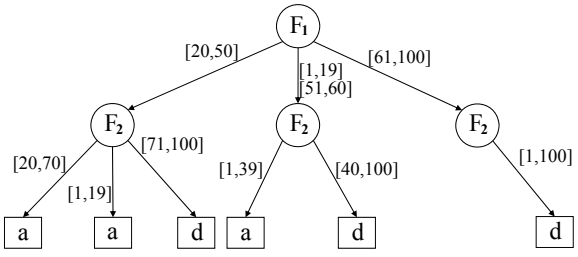

Fig. 3. A firewall decision diagram

Verifying whether a firewall decision diagram satisfies a property rule is based on the following theorem.

Theorem 2 (Firewall Verification Theorem): A firewall decision diagram satisfies a property rule if and only if the property rule does not conflict with any rule defined by a decision path of the firewall decision diagram.

A decision path in a firewall decision diagram is a path from the root to a terminal node. Each decision path defines a rule. For example, the leftmost decision path in Figure 3 defines the following rule:

$$F_1 \in [20,\ 50] \wedge F_2 \in [20,\ 70] \rightarrow accept$$

Figure 1 shows the six rules defined by the six decision paths of the firewall decision diagram in Figure 3.

The firewall verification algorithm is in Figure 4. This algorithm first converts the given firewall to an equivalent firewall decision diagram. Then it begins to traverse the diagram from its root. Let the property rule be $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec \rangle$. For any edge e in a firewall decision diagram, we use $I(e)$ to denote the label of e. For any node v in a firewall decision diagram, we use $F(v)$ to denote the label of v. Let F_1 be the label of the root. For each outgoing edge e of the root, we compute $I(e) \cap S_1$. If $I(e) \cap S_1 = \emptyset$, we skip edge e and do not traverse the subgraph that e points to. If $I(e) \cap S_1 \neq \emptyset$, then we continue to traverse the subgraph that e points to in a similar fashion. Whenever a terminal node is encountered, we compare the label of the terminal node and $\langle dec \rangle$. If they are different, we then terminate the traversal and report that the given firewall does not satisfy the given property. We use $e.t$ to denote the (target) node that edge e points to.

Firewall Verification Algorithm

Input : (1)A firewall
(2)A property rule $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec \rangle$
Output : *true* if the firewall satisfies the property rule;
false otherwise.
Steps:
1. Convert the given firewall to a firewall decision diagram;
2. **return**(Verify($root, (F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec \rangle$));

Verify(v, $(F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec \rangle$)
1. **if** (v is a terminal node) and ($F(v) = \langle dec \rangle$)
   then return true;
   if (v is a terminal node) and ($F(v) \neq \langle dec \rangle$)
   then return false;
2. **if** (v is a nonterminal node) **then**
   /*Let F_i be the label of v*/
   for each edge e in $E(v)$ **do**
   if ($I(e) \cap S_i \neq \emptyset \wedge$
   $\sim Verify(e.t, (F_1 \in S_1) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec \rangle$))
   then return false;
3. **return true**;

Fig. 4. Firewall Verification Algorithm

III. Experimental Results

We implemented our firewall verification algorithm in Java JDK 1.4. To evaluate the performance of this algorithm, first, we ran our algorithm on two real-life firewalls. Second, we stress tested our algorithms on a large number of synthetic firewalls. These experiments were carried out on a SUN workstation running Solaris 9 with 1Ghz CPU and 1 GB memory. In both cases, the experimental results show that our algorithms perform and scale well.

We obtained two real-life firewalls for this study. One firewall, with 87 rules, is from a university. The other, with 661 rules, is from a private company. Table III shows the performance of our algorithms on these two firewalls.

TABLE III
Performance of firewall verification algorithms on two real-life firewalls

	# Rules	FDD Construction	Verification
Firewall I	87	9 ms	0.1 ms
Firewall II	661	98 ms	0.3 ms

Firewall configurations are considered confidential due to security concerns. To further evaluate the performance of our algorithms on large firewalls, we run our algorithms on synthetic firewalls of large sizes. Every rule in a synthetic firewall has five fields: source IP address, destination IP address, source port number, destination port number, and protocol type. We generate a large number of rules in a recursive fashion. First, we randomly pick some source IP prefixes from our real-life firewalls. These IP prefixes together with the source IP prefix that matches all IP addresses form a set of source IP prefixes that will be used to generate new rules. Second, for each prefix in this set, we repeat the same procedure to generate a set of destination IP prefixes. This process recursively continues for source port, destination port, and protocol type. The decision for each rule is completely random. This process of generating rules is similar to the way that firewall administrators write rules. More interestingly, we have found that the firewalls generated using this process conform well to the characteristics of real-life firewalls [3], [15].

In our experiments, we generate 100 firewalls of each fixed size. For each firewall, we run our algorithm 100 times. In each run of the algorithm we generate the property rule randomly. Figure 5 shows the average running time for converting a firewall to a firewall decision diagram. Figure 6 shows the average running time for verifying a property rule on a firewall decision diagram. From Figure 6 we can see that verifying a property takes only a few milliseconds even for a large firewall. Note that the time for constructing a firewall decision is a one-time cost. Once a firewall decision diagram is constructed from a firewall, we can use it to verify multiple properties.

IV. Related Work

Previous work that is closest to ours is firewall testing [6], [19]–[21], [26], [29]. In these testing methods, test cases, *i.e.*, packets, are first generated based on a given firewall and the properties that the firewall needs to satisfy; then the generated packets are injected into the firewall to check whether the packet gets accepted or discarded. Our firewall verification technique is fundamentally different from these firewall testing techniques in two aspects. First, our firewall verification technique can verify a property accurately, while firewall testing techniques cannot because testing all possible (2^{88} IP) packets is computationally infeasible. Second, our firewall verification technique does not involve injecting packets into a firewall and inspecting the outcome of the firewall, which are often time and labor consuming. Although packet injection and inspection can be automated, such automation usually requires to take the firewall device offline, which is often not affordable.

Some interesting work has been done on firewall policy design and analysis. However, none of the previous work explored the change-impact analysis of firewall policies. Firewall decision diagrams were introduced in [13] for specifying firewall policies. The method of diverse firewall design was introduced in [23]. A Lisp-like language was introduced in [16] for specifying a high-level packet filtering policy. In a similar vein, a UML-like language was presented in [5] for specifying global filtering policies. Algorithms for detecting redundant rules were presented in [24]. Change-impact analysis of firewall policies was discussed in [22].

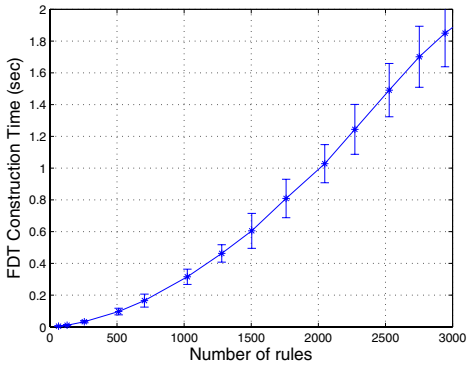

Fig. 5. Performance of firewall decision diagram construction

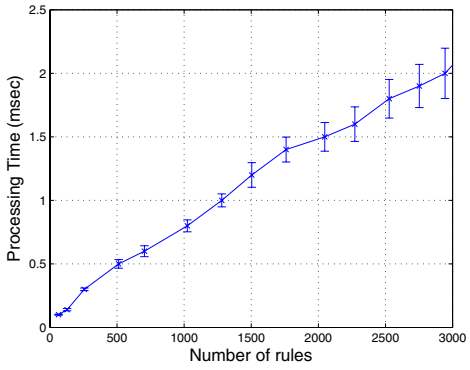

Fig. 6. Performance of firewall verification algorithm

Detecting conflicts among firewall rules was studied in [4], [8], [17]. In [9], [18], [25], [27], [30], a few firewall analysis tools were presented. In [12], algorithms for detecting firewall policy anomalies in distributed environment were proposed. This work focuses on how to efficiently detect all pairs of conflicting rules in a firewall. Some anomalies were defined and techniques for detecting anomalies were presented in [2], [32]. A good review of existing firewall technologies is in [7].

There are some tools currently available for network vulnerability testing, such as Satan [10], [11] and Nessus [28]. These vulnerability testing tools scan a private network based on the current publicly known attacks, rather than the requirement specification of a firewall. Although these tools can possibly catch some of the errors that allow illegitimate access to the private network, they cannot find the errors that disable legitimate communication between the private network and the outside Internet.

V. Conclusions

This paper presents a method for formally verifying firewall policies. Such a tool is extremely useful in many ways. For example, it can be used in firewall debugging and troubleshooting. It also can be used iteratively in the process of designing a firewall. A firewall administrator can use this tool to unambiguously demonstrate to their manager that the firewall satisfies the organization's security policies.

We have implemented our firewall verification algorithm and evaluated it on both real-life firewall policies and synthetic firewall policies of large sizes. The experimental results show that the performance of our algorithm is very efficient.

It is worth emphasizing that the methods and algorithms presented in this paper are not limited to the design and analysis of firewall policies. Rather, they can be applied to other rule based systems that can be expressed as a sequence of rules, such as access control rules specified in XACML [1].

Acknowledgement

This work is supported in part by the National Science Foundation under Grant No. CNS-0716407.

References

[1] OASIS eXtensible Access Control Markup Language (XACML). http://www.oasis-open.org/committees/xacml/, 2005.

[2] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, pages 2605–2616, March 2004.

[3] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: Is there an alternative to CAMs? In *Proceedings of IEEE INFOCOM*, 2003.

[4] F. Baboescu and G. Varghese. Fast and scalable conflict detection for packet classifiers. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, 2002.

[5] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *Proceeding of the IEEE Symposium on Security and Privacy*, pages 17–31, 1999.

[6] CERT. Test the firewall system. http://www.cert.org/security-improvement/practices/p060.html.

[7] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 2003.

[8] D. Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Symp. on Discrete Algorithms*, pages 827–835, 2001.

[9] P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, pages 100–107, 2001.

[10] D. Farmer and W. Venema. Improving the security of your site by breaking into it. *http://www.alw.nih.gov/Security/Docs/admin-guide-to-cracking.101.html*, 1993.

[11] M. Freiss. *Protecting Networks with SATAN*. O'Reilly & Associates, Inc., 1998.

[12] J. Garcła-Alfaro, F. Cuppens, and N. Cuppens. Analysis of policy anomalies on distributed network security setups. In *Proceedings of the 11th European Symposium On Research In Computer Security (Esorics)*, September 2006.

[13] M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS-04)*, pages 320–327, March 2004.

[14] M. G. Gouda and A. X. Liu. Structured firewall design. *Computer Networks Journal (Elsevier)*, 51(4):1106–1120, March 2007.

[15] P. Gupta. *Algorithms for Routing Lookups and Packet Classification*. PhD thesis, Stanford University, 2000.

[16] J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 120–129, 1997.

[17] A. Hari, S. Suri, and G. M. Parulkar. Detecting and resolving packet filter conflicts. In *Proceedings of IEEE INFOCOM*, pages 1203–1212, 2000.

[18] S. Hazelhurst, A. Attar, and R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Proceedings of the Workshop on Dependability of IP Applications, Platforms and Networks*, 2000.

[19] D. Hoffman, D. Prabhakar, and P. Strooper. Testing iptables. In *Proceedings of the 2003 conference of IBM Centre for Advanced Studies*, pages 80–91, 2003.

[20] D. Hoffman and K. Yoo. Blowtorch: a framework for firewall test automation. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 96 – 103, 2005.

[21] J. Jürjens and G. Wimmel. Specification-based testing of firewalls. In A. Ershov, editor, *Proceedings of the 4th International Conference Perspectives of System Informatics (PSI'01)*, LNCS. Springer, 2001.

[22] A. X. Liu. Change-impact analysis of firewall policies. In *Proceedings of the 12th European Symposium Research Computer Security (ESORICS), LNCS 4734, J. Biskup and J. Lopez Ed., Springer-Verlag*, page 155C170, September 2007.

[23] A. X. Liu and M. G. Gouda. Diverse firewall design. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN-04)*, pages 595–604, June 2004.

[24] A. X. Liu and M. G. Gouda. Complete redundancy detection in firewalls. In *Proceedings of 19th Annual IFIP Conference on Data and Applications Security, LNCS 3654, S. Jajodia and D. Wijesekera Ed., Springer-Verlag*, pages 196–209, August 2005.

[25] A. X. Liu, M. G. Gouda, H. H. Ma, and A. H. Ngu. Firewall queries. In *Proceedings of the 8th International Conference on Principles of Distributed Systems, LNCS 3544, T. Higashino Ed., Springer-Verlag*, pages 124–139, December 2004.

[26] M. R. Lyu and L. K. Y. Lau. Firewall security: Policies, testing and performance evaluation. In *Proceedings of the 24th International Conference on Computer Systems and Applications (COMPSAC'2000)*, pages 116–121, October 2000.

[27] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 177–187, 2000.

[28] Nessus. http://www.nessus.org/. March 2004.

[29] D. Senn, D. Basin, and G. Caronni. Firewall conformance testing. In *Proceedings of the Testcom (Testing of Communicating Systems)*, May 2005.

[30] A. Wool. Architecting the lumeta firewall analyzer. In *Proceedings of the 10th USENIX Security Symposium*, pages 85–97, August 2001.

[31] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004.

[32] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra. Fireman: a toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy*, May 2006.