Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 62 (2015) 159 – 166

Procedia Computer Science

The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

# Modeling Firewalls for Behavior Analysis

Patrick G. Clark[a]*, Arvin Agah[a]

[a]*Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence, KS 6045 USA*

**Abstract**

This work presents a software behavioral model of the capabilities found in firewall type devices and a process for taking vendor specific nuances to a common implementation. This includes understanding interfaces, routes, rules, translation, and policies; modeling them in a common manner such that different models may be compared to each other for functional similarity. This work makes use of recent efforts to model firewall policies in a concise efficient data structure referred to as a Firewall Policy Diagram (FPD). The structure facilitates the canonical representation of a policy as well as human comprehension of the policy. Its use with behavior modeling is to capture and compare the results of a potentially large solution space.



Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

*Keywords:* Firewall Behavior Analysis; Firewall Modeling; Human Comprehension; Firewall Policy Diagram (FPD); Network Software Architecture; Software-Defined Networking (SDN); Software Security Analysis

## 1. Introduction

This work presents a framework in which to model individual firewalls such that software based analysis and other formal analysis methods may be used with the model. Some examples of industry uses for modeling a network and firewalls in software:

- Network trace analysis for understanding how a packet will traverse the device without physically sending the packet. In addition to an individual packet, a packet *space* in the form of an FPD may traverse the network to understand what will make it from point *A* to point *B*.

* Corresponding author. Patrick G. Clark
*E-mail address:* patrick.g.clark@gmail.com



Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)
doi:10.1016/j.procs.2015.08.429

- Logical comparisons of firewall vendor implementations. If two firewalls are configured to behave the exact same way but are from two different implementations, modeling the behavior of each vendor in software allows for the formal verification that each ingress and egress FPD are identical. Subsequently, if they are not identical, the FPD used to traverse the spanning graph can show what is different from what is potentially a large solution space.
- Behavior Abstraction Modeling of specific firewall vendors serves as the basis for automated translation from one vendors' configuration to another with certainty of how the device will behave.
- Participate in a larger software modeled network for more comprehensive simulations.

The remaining sections include a brief description of the common behavioral components found in modern firewalls, how those common elements may be abstracted into a spanning graph, and then how that spanning graph may be used to experiment with device behavior.

## 2. Modeling Firewall Behavior

In a broad description, a firewall is a network device that is used to isolate an organization's internal network from outside networks, allowing some packets to pass and blocking others. Firewalls allow two entities to connect their networks together through existing infrastructure and protocols, while securing the private networks behind them[1].

### *2.1. Behavioral Components of Modern Firewalls*

Traditional firewall capabilities were strictly focused on filtering traffic and applying the local security policy of an organization. Some firewalls were more sophisticated than others, but in general the filtering focused on an individual packet (packet filters), stateful packet flow (stateful filters), or application related traffic (application gateways)[2]. All of these capabilities are built on top of the known TCP/IP stack and the only goal is to accurately allow only the communication channels defined in the local policy.

However, firewall vendors have continued to increase the scope of what defines a firewall. Looking at any modern firewall product today, one will typically find a combination of router, network address translation, and filtering capabilities. In addition, each of these sub-components may be broken down further into other elements such as virtual routers, embedded NAT inside of rules, and multiple filtering policies applied at different places. Therefore, for any abstraction of a firewall there must exist a mechanism to represent these capabilities so that accurate results may be computed.

#### *2.1.1. Interfaces*

An interface on a firewall is the physical network connection to the wire (or radio) that will transmit the traffic to and from the parties communicating[2]. In the strictest definition of the firewall and packet-filtering device, two network interfaces exist. This represents a physical separation between one network and another, requiring the traffic to flow through the firewall from one interface to the other and vice versa. However, as discussed in[3], the firewall will typically have many more than two interfaces, which means that a firewall is no longer just a filtering device, but must also decide the appropriate egress interface for traffic.

#### *2.1.2. Security Policies*

A firewall will have single or multiple security policies to be applied to incoming or outgoing traffic. These policies are typically an ordered sequence of rules that follow the general form:

$$\langle predicate \rangle \rightarrow \langle decision \rangle$$

The *predicate* defines a boolean expression over the fields in a packet tuple that are evaluated and the physical network interface from which the packet arrives. A decision is typically accept or deny with the possibility of additional actions, such as an instruction to log the action[1].

*2.1.3. Routes*

For this work, a route may be defined as a simple one tuple rule with a decision being the egress interface. The one tuple of the traffic being processed is the destination. For a particular routing rule, the destination can be identified as an IP Address, address range, or Classless Inter-Domain Routing (CIDR) format. Therefore, in a similar manner as security rules, the solution space can be split as the traffic is processed.

*2.1.4. Network Address Translation*

Network address translation (NAT) is a feature often offered by firewall and router vendors to solve a number of routing and addressing challenges in packet switched networks. NAT implementation will typically function as a translation table such that the inbound traffic will match an entry (either source or destination address) that is translated to another address on the egress side. The firewall device is then required to keep track of that conversation in order for response packets to have the reverse translation applied and arrive at the appropriate destination. There are typically three types of NAT: source address translation (SNAT), destination address translation (DNAT), and port translation (PAT). Each of these are roughly the same idea, and each will require a translation table, yet they will operate on different fields of the packet[2]. For example, a DNAT table will typically look like the following:

$$\langle \textit{Destination Address} \rangle \rightarrow \langle \textit{Translated Address} \rangle$$
$$192.168.2.1 \rightarrow 74.125.228.39$$
$$10.1.1.10 \rightarrow 157.56.237.251$$
$$\ldots \rightarrow \ldots$$

Then, when a packet arrives on an interface, it will look at where the packet is going and replace the destination of the packet with the appropriate translated address.

## 3. Firewall Abstractions and Spanning Graphs

When dealing with a firewall or network device, the entire goal of the broader network is to provide a path from point A to point B. Because the firewall is responsible for taking traffic in from one interface and sending it out another, there are internal paths that take the data through the various control and routing structures, as defined in the previous section. These paths and structures can be abstracted into *behavior rules*, *behavior groups*, *interface switches* and a *spanning graph*, which provides the links between the behavior groups and interface switches.

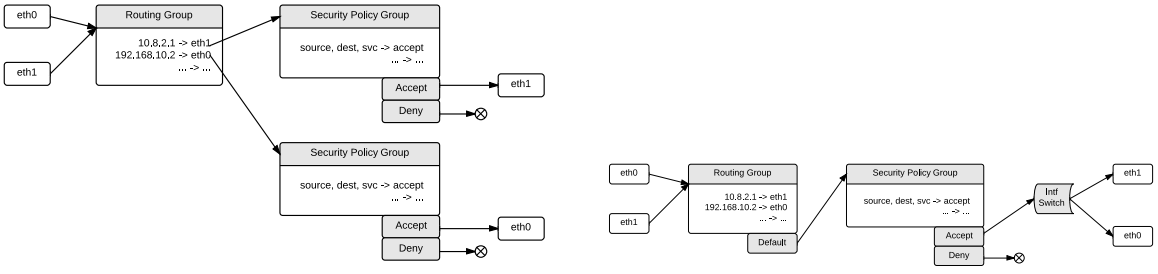


Fig. 1: Behavior spanning graph (a) without and (b) with a switch.

*3.1. Behavior Rules*

As a first step when considering how a particular firewall might process data, one typically thinks about the physical interfaces. Interfaces can be thought of as spokes off of a firewall allowing it to participate in a larger network. To model the internals, one must consider the elements we have defined: routes, security rules, and NAT. Essentially all three items may be thought of as *predicates* and *actions*. The predicate of all three types are over the known tuples or fields of an Internet Protocol and potentially TCP and UDP port, i.e., the source address, destination

address, source port and destination port. The actions that may be defined when the predicate matches are *accept*, *deny*, or *next action*; with two additional state transition operators: *translate* and *egress interface*. When a predicate matches, only one action may be applied (accept, deny, or next), however, any (or all) of the state transition operators may be applied. Using these elements and the initial link from the ingress interface, an internal spanning graph may be constructed generating a data path through the device and out the egress interface that may then be linked to the overall network spanning graph. These ⟨predicate⟩ → ⟨action⟩ constructs are identified as *Behavior Rules*.

### *3.2. Behavior Group*

A behavior group is defined as a set of behavior rules such that they are processed top-down and the first matching predicate for a particular individual packet performs the associated action. The common use for a behavior group is the encapsulation of the larger ideas of a firewall. For example, it is common when modeling a particular vendor firewall to identify a *routing* behavior group or a *security policy* behavior group such that all corresponding routes or security rules are in that group, and may potentially be processed as one entity.

In addition to providing a grouping mechanism for behavior rules, each behavior group possesses three overall actions: *accept, deny,* and *default*. These actions may be linked to the next group in the spanning graph, or potentially to the egress interface. The behavior group accept action will be applied to a traversing packet when the packet has matched a behavior rule predicate and the action is accept. In a similar manner, the behavior group deny action will follow the same logic but takes the deny path. Finally, the behavior group default action will be applied if no behavior rule predicate matches the packet.

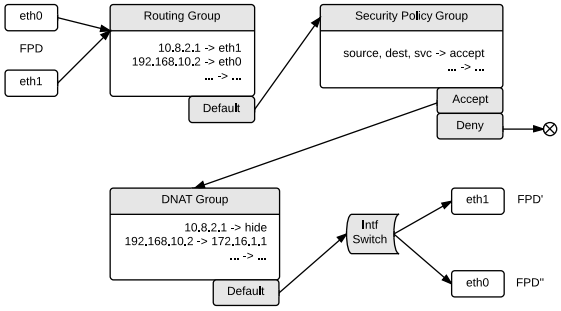


Fig. 2: Example behavior spanning graph with FPD.

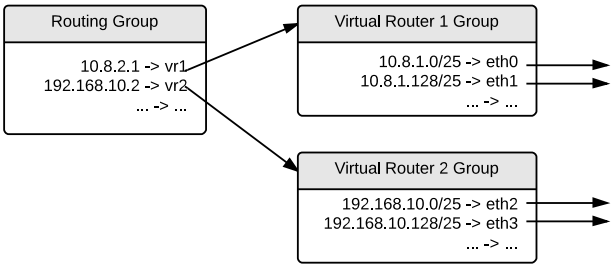


Fig.3: Example virtual router configuration

### *3.3. Interface Switch*

An important concept we present to reduce the number of paths that must be represented in a spanning graph is the use of an *ingress interface*, an *egress interface action*, and an *interface switch*. Switches may be placed in the behavior group model to act upon two elements. The first is the inbound interface the traffic passes. The second is a matching state transition behavior rule that identifies the egress interface at some point in the spanning graph. The reason for providing this capability is that different firewall vendors make egress interface decisions at different times throughout the control flow. The egress interface sometimes will not impact the internal behavior groups traversed until the final step. Without having the means to act later on a state that was set early in the path, many duplicate paths and groups would be needed. Figure 1 demonstrates a spanning graph without and with an interface switch such that without an interface switch, the security policy group is duplicated. By using the switch in Figure 1(b), the potentially large security policy group will only need to be modeled once. Figure 1(b) also demonstrates how the egress interface is selected in the routing group and then acted upon at the interface switch.

Another reason that interface switches are useful is for firewalls that employ *zone* definitions. A zone in firewall language is typically the grouping of a number of interfaces into a logical area of the network. For example, eth0 and eth1 are considered an *internal zone* with eth2 and eth3 in the *unsafe zone*. A vendor may then identify a

security policy when the traffic is passing from zone-to-zone and is specific to that zone-to-zone transition. In this example, there may exist a security policy that will need to be applied if the traffic arrives in the internal zone and is destined for the unsafe zone. Without an interface switch there would need to be a path for every interface-to-interface combination. This is regardless if those interfaces shared the same zones, with the result being duplicated behavior groups and paths.

Address and port translation provide another reason the interface switch provides a useful abstraction for when the egress interface is selected. Some firewall vendors will apply routes and security rules to the *pre*-translation packet, and some will apply them to the *post*-translation packet. This is an important distinction because in the case where the application of the routes is pre-translation, the subsequent security rules will need to be duplicated for every egress interface when an interface switch is not used. This is in a similar manner to what is seen in Figure 1, and an example of how an interface switch simplifies the processing and model.

### *3.4. Spanning Graph*

All of the elements described form a directed acyclic graph rooted at the ingress interfaces with the leaves identified as the egress interfaces. This allows simulation of the traffic to be based on interface origination. This is also a mechanism to compare two firewall types that process traffic differently, but expect the same external results. Figure 2 is an example of the behavior elements coming together to form a behavior spanning graph. In subsequent sections we describe how traversing this spanning graph with the correct data structure will allow for a better understanding of the traffic that may pass the firewall.

### *3.5. Modeling Sub Elements*

Until this point we have covered the general elements of the behavior abstraction model such that there was a single routing group, security policy group, or destination NAT group. However, the reality of modern firewalls is that they are often constructed of smaller elements that may be linked and reused. Constructs such as virtual routers and zone policies may easily be represented as their own behavior groups that are linked. For example, a virtual router is typically a routing table with an action of “next”, taking the processing to another group until finally an *egress interface* decision is made. Figure 3 diagrams this behavior group configuration such that the first routing group uses *next* action that matches the packet predicate in order to direct the processing path to the virtual router group. Once in the virtual router group, the *egress interface* is selected and processing continues to the next group. Furthermore, to illustrate the expressiveness of the behavior group model, zone-to-zone policies may be represented by using an interface switch before selecting the security policy to be processed, allowing the egress interface to be selected once during the model from the routing table and then used multiple times for selecting the zone policy, and then again to select the egress interface. This is in contrast to a less expressive abstraction that requires constructing the same security rules that will be applied for all possible interface-to-interface combinations.

## 4. Modeling Traffic Solution Space

In the previous sections we have gone through a detailed explanation of how individual elements of a firewall may be abstracted into a directed acyclic graph that may be traversed from inbound interface to outbound interface. This is useful for individual packet tracing; however, the more important capabilities come when used with a data structure capable of representing the entire solution space that a packet may represent. In this work we use the Firewall Policy Diagram (FPD).

### *4.1. Firewall Policy Diagram*

A Firewall Policy Diagram (FPD) is a set of data structures and algorithms used to model a firewall policy into an entity allowing efficient mathematical set operations. The entity also has the ability to reconstitute the policy into a set of human comprehensible rules[4,5]. The FPD forms the base of the behavior-modeling engine and allows the fast and efficient manipulation of the packet space. This achieves a complete and thorough understanding of the space as

it comes in an ingress interface and exits another egress interface, yielding an accurate understanding of what traffic would have passed.

The internal storage mechanism of a FPD uses Reduced Ordered Binary Decision Diagrams (ROBDD or BDD)[6,7]. These data structures were introduced as an efficient way to capture hierarchical binary data and related works have described their use in firewall policy validation[8,9,10]. In a similar manner to the FIREMAN system[9], policies and rules are modeled as variable sets represented as BDDs. Using a BDD is an efficient way to represent a Boolean expression, like $(a \vee b) \wedge c$. Extending this concept to firewall policies, the variables in the expression become the bits of the associated IPv4 field. For our research, 32 bits representing the source address, 32 bits representing the destination address, 8 bits representing the protocol, and 16 bits representing the destination port. This means that for a particular accept space; there are 88 variables and $2^{88}$ potential combinations of variable values.

A full description of the algorithms involved with manipulation and extraction of human comprehensible rules can be found in related work[4].

### *4.2. Walking the Spanning Graph*

When considering how data will pass through a firewall device, they must first be accurately modeled using behavior groups. The next step to fully understand what will pass through a firewall configuration is to construct an FPD representing the entire solutions space, *U*, at each interface root. Then the spanning graph is traversed from root to depth with the FPD splitting at each path and changing as each behavior group is processed until it reaches the egress interface leaf. Each leaf FPD result may then be combined to produce the final FPD, representing an accurate space of what traffic can pass through the firewall and out of that interface. The process tests all possible scenarios through all inbound interfaces.

It is this sort of formal verification that allows a firewall configuration team to be confident that they are not allowing any more access than the business requires and managing risk appropriately. Figure 2 demonstrates this operation visually by starting with an FPD at the root of the spanning graph, eth0 and eth1. As the FPD traverses the behavior groups a portion of the space ends up at the leaf interfaces, eth1 and eth0. The resulting FPDs at each interface represent the packet space that has passed through the device and out those individual interfaces.

### *4.3. Performance: Avoiding the Linear Case*

When taking the behavior groups and behavior rules that make up each of the individual firewall entities, the initial idea of how traffic may be processed is linear. Meaning, one will process traffic as a true firewall does, attempting a match in a linear method, one behavior rule at a time to one packet at a time. While this is straightforward, it is also performance prohibitive as the full testing of a firewall configuration requires $2^{88} \times br$. Where *br* is the number of behavior rules that exist in the device and the $2^{88}$ is the solution space represented by an IPv4 packet. It becomes exponentially worse when dealing with IPv6. With a reformulation of each behavior group, the processing time may be bound to the number of decisions that must be made, as opposed to the number of behavior rules and the size of the address space. This same model may then be replayed for behavior groups that represent routing tables and NAT tables. The formulation of the problem is different, but is still a factor of the number of decisions that must be made. In a review of the routing behavior group, the decision to be made is what egress interface matches the traffic and with NAT, the addresses to be translated to.

## 5. Experiments

An experiment to test the performance of a common filtering and routing firewall was designed containing 10,000 routes and 10,000 security rules with two interfaces, eth0 and eth1. The results show the time taken to test all possible packets through the simulated device was 210 milliseconds. A comparison with other model based testing tools show slower processing time with substantially smaller rule base sizes. Model-based conformance testing by[11] involves experiments with the experiment size of 7 rules generating over 700 test cases taking about 3000 seconds, and 4 networks adding an additional 20,000 seconds. A test case generation tool presented in[12] indicates that the

number of tests generated is a cross product of the number of individual sources, destinations, and services found in the rule set, suggesting a much higher complexity when using this brute force method. Furthermore, experiments in[13] show similar results when using a BDD as an internal representation of the policy, although the work was focused on model checking and not simulation. The results support the use of this behavior abstraction model and simulation in industry firewall configuration simulations. A sub-second simulation number is important because it will support further expansion of the behavior abstraction model such that it may participate in the larger network topology, both helping to verify and secure the organization.

## 6. Related Work

Brucker et al. (2008) presents a simple network modeling formalization with the common packet tuples found in many other works[11,14,15,16,17,18,9,1,19,20]. Brucker et al. (2008) uses a model based testing tool in conjunction with a testing framework that analyzes a single policy rule set and generates a list of test cases to be run against the actual firewall with an expected result[11]. The work focused on modeling a single policy with the experiment size of 7 rules.

In a similar manner to our work,[21] presents a firewall modeling framework based on Netfilter[22] and the grouping mechanism. This work considers routing topologies and presents a method to condense a multi-policy and multi-device topology into a single policy with the representation of interfaces in the tuples of the model. They do not discuss packet translation or include it in their models.

El-Atawy et al. presents a framework to generate test data for testing an actual firewall device. The work focuses on the ACL and does not cover the many other capabilities of modern firewalls (routes, NAT, etc.). While they focus on generating the smallest number of test cases based on network segmentation, there is also a behavioral modeling component to the effort. They use BDDs to check the rules for the appropriate network segments to include in the test, without a discussion of how data is extracted from the underlying model[13].

Frameworks capable of generating validation traffic for a particular firewall presented in several works[23,12]. In a similar manner to other topology mapping programs[24,25], the work generates and sends actual traffic through the system, exercising a firewall's internal filtering to ensure consistency.

A survey of firewall modeling is presented by Zalvia[26]. This work demonstrates that the focus is typically on the individual firewall policy and some efforts on modeling the security posture as a broader inter-connected network. Further works describe the analysis engines that model security policies[14,27,28].

FIREMAN focuses on finding configuration anomalies in firewall policies. The primary similarity with our work is the extension of a group model that is also extended from Linux Netfilter[22]. The work suggests converting individual and networked policies into a linear group of processing that includes rules and links between firewalls[9]. Our work extends some of the concepts presented to cover NAT, interfaces, and routing tables. In addition, instead of enumerating all possible linear activities which might occur in a complex network, our work focuses on sharing paths through a spanning directed acyclic graph structure.

The work most closely related to ours involves modeling attack graphs for exploitation of individual hosts in a network[10]. The effort uses a chains concept[22] in portions of the work to reflect how a virtualized space might flow through an attack graph. In addition to ACLs, the work is able to model NAT and routes. They focus on a modeling attack vectors through a reduced attack graph.

## 7. Conclusions

In this effort we have presented a framework for modeling the known capabilities of modern firewalls such that the devices may be abstracted from the manufacturer specific implementation details. This abstraction allows a firewall to be modeled in a common manner in software, such that functional and configuration oriented testing may be achieved with more sophisticated plans than using a brute force, test all possibilities, approach that is seen today. In addition, this framework may be extended to assist in translation and migration from one vendor specific implementation to another, thus allowing the verification of the translation of complicated firewalls with certainty.

## References

1. Liu, A.X., Gouda, M.G.. Complete redundancy detection in firewalls. In: *Proceedings of the 19th Annual IFIP Conference on Data and Applications Security*. 2005, p. 196–209.
2. Kurose, J.F., Ross, K.W.. *Computer Networking: A Top-Down Approach*. Addison Wesley; 4th ed.; 2007.
3. Chapple, M.J., D'Arcy, J., Striegel, A.. An analysis of firewall rulebase (mis)management practices. *ISSA Journal* 2009;:12–18.
4. Clark, P.G., Agah, A.. Firewall policy diagram (fpd): Structures for firewall behavior comprehension. *International Journal of Network Security* 2015;3(1).
5. Clark, P.G., Agah, A.. Firewall policy query language for behavior analysis. In: *Proceedings of the International Conferences on Security and Management (SAM2014)*. 2014, p. 185–191.
6. Shannon, C.E.. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers* 1938; 57(12):713–723. doi:10.1109/T-AIEE.1938.5057767.
7. Bryant, R.E.. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 1992;24:293–318.
8. Hazelhurst, S., Attar, A., Sinnappan, R.. Algorithms for improving the dependability of firewall and filter rule lists. In: *Proceedings of the 2000 International Conference on Dependable Systems and Networks*. 2000, p. 576–585.
9. Yuan, L., Mai, J., Su, Z., Chen, H., Chuah, C., Mohapatra, P.. Fireman: A toolkit for firewall modeling and analysis. *IEEE Symposium on Security and Privacy* 2006;:199–213.
10. Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.. Modeling modern network attacks and countermeasures using attack graphs. In: *Proceedings of the 2009 Computer Security Applications Conference*. 2009, p. 117–126. doi:10.1109/ACSAC.2009.21.
11. Brucker, A., Bru¨gger, L., Wolff, B.. Model-based firewall conformance testing. In: *Testing of Software and Communicating Systems*; vol. 5047 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg; 2008, p. 103–118.
12. Tuglular, T., Kaya, O., Muftuoglu, C., Belli, F.. Directed acyclic graph modeling of security policies for firewall testing. In: *Proceedings of the 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement*. 2009, p. 393–398.
13. El-Atawy, A., Ibrahim, K., Hamed, H., Al-Shaer, E.. Policy segmentation for intelligent firewall testing. In: *Proceedings of the 1st IEEE ICNP Workshop on Secure Network Protocols*. 2005, p. 67–72. doi:10.1109/NPSEC.2005.1532056.
14. Bartal, Y., Mayer, A., Nissim, K., Wool, A.. Firmato: a novel firewall management toolkit. In: *Proceedings of the IEEE Symposium on Security and Privacy*. 1999, p. 17–31. doi:10.1109/SECPRI.1999.766714.
15. Al-Shaer, E.S., Hamed, H.H.. Design and implementation of firewall policy advisor tools. Technical Report; School of Computer Science, Telecommunications and Information Systems, DePaul University, Chicago, USA; 2002.
16. Al-Shaer, E.S., Hamed, H.H.. Modeling and management of firewall policies. *IEEE Transactions on Network and Service Management* 2004;1(1):2–10.
17. Al-Shaer, E.S., Hamed, H.H.. Discovery of policy anomalies in distributed firewalls. In: *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*; vol. 4. 2004, p. 2605–2616.
18. Gouda, M.G., Liu, A.X.. Firewall design: Consistency, completeness, and compactness. In: *Proceedings of the 24th International Conference on Distributed Computing Systems*. 2004, p. 320–327.
19. Chao, C.. A flexible and feasible anomaly diagnosis system for internet firewall rules. In: *Proceedings of the 13th Asia-Pacific Network Operations and Management Symposium*. 2011, p. 1–8.
20. Ju¨rjens, J., Wimmel, G.. Specification-based testing of firewalls. In: Bjørner, D., Broy, M., Zamulin, A., editors. *Perspectives of System Informatics*; vol. 2244 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. ISBN 978-3-540-43075-9; 2001, p. 308–316.
21. Jeffrey, A., Samak, T.. Model checking firewall policy configurations. In: *Proceedings of the 2009 IEEE International Conference on Policies for Distributed Systems and Networks*. 2009, p. 60–67. doi:10.1109/POLICY.2009.32.
22. Russell, R.. The netfilter.org project. http://www.netfilter.org. 2011. URL: http://www.netfilter.org/.
23. El-Atawy, A., Samak, T., Wali, A., Al-Shaer, E.S., Lin, F., Pham, C., et al. An automated framework for validating firewall policy enforcement. *IEEE International Workshop on Policies for Distributed Systems and Networks* 2007;:151–160.
24. Govindan, R., Tangmunarunkit, H.. Heuristics for internet map discovery. In: *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*; vol. 3. 2000, p. 1371–1380.
25. Xie, G.G., Zhan, J., Maltz, D.A., Zhang, H., Greenberg, A., Hjalmtysson, G., et al. On static reachability analysis of ip networks. In: *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*; vol. 3. 2005, p. 2170–2183.
26. Zaliva, V.. Firewall policy modeling, analysis and simulation: a survey. *Source-Forge, Tech Rep* 2008;:1–11.
27. Mayer, A., Wool, A., Ziskind, E.. Fang: A firewall analysis engine. In: *Proceedings of the 2000 IEEE Symposium on Security and Privacy*. 2000, p. 177–187.
28. Wool, A.. Architecting the lumeta firewall analyzer. In: *Proceedings of the 10th conference on USENIX Security Symposium*; vol. 10. 2001, p. 85–97.