On the Usability of Firewall Configuration

Tina Wong
Carnegie Mellon University
tinawong@cmu.edu

ABSTRACT

The firewalls in an enterprise network must be configured correctly or the internal corporate network can be infiltrated, leading to serious security, financial and performance implications. However, firewall configuration is a complex and error-prone task. Configuration languages are like assembly languages: they are low-level and vendor-specific. Moreover, usually multiple firewalls must be configured to protect an enterprise network. This task has been compared to programming a distributed system with an assembly language. While many researchers have tackled the firewall configuration problem from various perspectives, including new models, languages and complete systems, little has been done from the usability standpoint. Recently, studies have demonstrated that administrators strongly prefer textual or command line interfaces (CLIs) over GUIs. Most administrators are reluctant to invest time to learn new models, languages or systems for their everyday tasks. In this paper, we study the firewall configuration problem from the usability perspective. We first propose models to measure the lexical and structural complexity of firewall configuration. Using these models, we examine where complexity lies in the configurations of real networks. With the assumption that CLI will remain as the main user interface for administrators, we suggest visualizations to make firewall configuration more usable.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Management, Security, Human Factors, Languages

Keywords

Firewalls, Routers, Configuration, Usability, Complexity

1. INTRODUCTION

It is well known that networks are difficult to manage and operate. Administrator errors are common and root causes of failures in networks [14]. In particular, configuring a network is complex and error-prone. The firewalls in an enterprise network must be configured correctly or the internal corporate network can be infiltrated, leading to serious security, financial and performance implications. However, the situation is bleak. Wool [18] conducted a quantitative study on 37 firewalls, and found that all of them have some form of misconfigurations. The author's conclusion is "complex rule sets are apparently too difficult for administrators to manage effectively."

There are several reasons behind the firewall configuration problem. It is a low-level, device-specific task. To protect a network, one needs to configure multiple firewalls in the network separately. A change in one firewall can potentially affect other firewalls, or even the whole network. The more complex firewalls can contain hundreds or even thousands or configuration commands. Furthermore, many have described firewall configuration languages as "arcane" and compared them to assembly languages. Configuration languages are designed by router vendors, with router processing in mind, and do not necessarily have the appropriate constructs for network administrators to specify their intent.

Moreover, as a network evolves, its configurations become even more difficult to understand, extend and debug. In Wool's study [18], the author makes the hypothesis that the most expensive and highest performance firewalls in his study would be the least misconfigured, as the organizations which deployed these firewalls should have more human resources to manage the networks. To his surprise, the results are the opposite. It turns out that these more powerful firewalls have longer histories and have had multiple engineers managing them, therefore making them more complex.

Many researchers have tackled the firewall configuration problem using "clean-slate" approaches – new configuration languages, user interfaces and complete systems that are designed to be high-level. However, we believe a gap exists between the research and network administration communities. Haber and Bailey [7] conclude in their ethnographic field study of system administrators that "we witnessed enough problems to believe that administration tools are often created without sufficient understanding of the full context of administration work." Both Haber and Bailey [7] and Botta el al [3] have found that administrators strongly prefer textual or command line interfaces (CLIs) over GUIs, even though many of them lack a formal background in computer science. Administrators perceived CLIs as faster, more flexible, trustworthy, reliable, robust and accurate. GUIs can sometimes hide important details or are buggy, which means administrators face risks in relying solely on them. Botta el al [3] states that "With a plain text editor like vi, the user (administrator) can be confident that what you see is what you get." This is contrary to the common understanding of the WYSIWYG principle, which applies mainly to GUIs for end-users, but not administrators.

In this paper, we make the assumption that CLI is the main user interface for firewall administrators. With this assumption in mind, we set out to find ways to support and simplify the act of configuring firewalls. We first propose complexity models to

Submitted to Workshop on Usable IT Security Management, part of Symposium On Usable Privacy and Security (SOUPS) 2008, July 23-25, 2008, Pittsburgh, PA, USA.

measure the lexical and structural complexity of firewall configuration (Section 3). These models allow us to systematically examine where the complexity lies in firewall configuration; the places in which administrators need help in reducing their mental model burdens. We apply the models on the configuration files from a university campus network and describe our observations (Section 4). Based on these observations, we propose several visualization tools that can be integrated into the configuration environment without replacing the CLI as the main user interface (Section 5). Finally, we compare and contrast our work with related work (Section 6).

2. BACKGROUND

In this section, we give a brief overview of firewall configuration. The main function of a firewall is to examine packets and decide to either allow or disallow packets according to a configured security policy. There are two types of firewalls: (1) personal firewall which protects the machine on which it is installed, and (2) enterprise firewall which sits between the Internet and the corporate network and protects the latter from the former. The security policy and configuration of enterprise firewalls are much more complex and the focus of this paper. Note that routers also implement firewall functions, that is, packet filtering, among other things. Thus, some networks may use routers as their firewalls, and do not have firewall-only devices. We use the words router and firewall interchangeably.

Router vendors have different configuration languages. The most widely used languages are those from Cisco and Juniper. For simplicity, we use a simplified form of the Cisco IOS syntax in the examples throughout this paper. The main configuration command for firewalls is a packet filter, which takes the following syntax:

```
access-list name {permit|deny} protocol source dest
```

A packet filter matches packets based on the protocol, source IP address and destination IP address values, and sometimes other optional values such as port number, and either permits or denies the packets. There can be multiple rules in a single packet filter. For example, the following packet filter "101" has four rules:

```
access-list 101 deny ip 10.0.0.0/8 any
access-list 101 deny ip 127.0.0.0/8 any
access-list 101 deny ip 192.168.0.0/16 any
access-list 101 permit any
```

The IP addresses in the prefixes 10.0.0.0/8, 127.0.0.0/8 and 192.168.0.0/16 are reserved addresses and should be not used in the public Internet. This packet filter blocks packets with these private source addresses (spoofed packets), and allows all other packets. In current firewall configuration languages, "first rule wins": when a packet is matched by a rule, the rest of the rules are not executed. Packet filters are applied to router interfaces. For example, packet filter "101" is applied on interface "Ethernet0", on both inbound and outbound traffic:

```
interface Ethernet0
 ip address 1.2.3.4 255.255.255.0
 ip access-group 101 in
 ip access-group 101 out
```

If "Ethernet0" is the interface on which inbound Internet traffic arrives on, then packets with spoofed IP addresses are not allowed to enter the internal network. Similarly, such packets are not allowed to leave the internal network for the public Internet. Packets with spoofed IP addresses are usually used by attackers to hide their identity.

3. COMPLEXITY MODELS

A network with well-maintained firewall configurations is more dependable and reliable. Maintainability is defined as "the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment" [9]. In large-scale software development, maintainability of source code is measured by quantitative models. Halstead Complexity [8] and Cyclomatic Complexity [11] are two widely used models. No such models exist for firewall configurations. In this section, we propose network-wide models to measure the lexical and structural complexities of a network's firewall configurations. These models can help to identify configuration commands within a network that are complex and thus need to be carefully maintained. Even though we present our complexity models in the context of firewalls, they are applicable to the configuration of other network devices, such as routers.

3.1 Lexical Complexity

The Halstead Complexity [8] measures the lexical complexity of source code. It is based on the number of operators and operands in source code. Five measurements are calculated: Program Length, Program Vocabulary, Program Volume, Difficulty and Effort. In particular, the Program Vocabulary and Program Volume measurements are shown to be correlated with the maintainability of source code. The Program Vocabulary n is the sum of the number of distinct operators and the number of distinct operands. It is related to the mental capacity demands of the source code on the software developer. A large vocabulary size means high demands on the developer. The Program Volume v is calculated as $v = N * (\log n)$, where N is the sum of the total number of operators and the total number of operands.

We apply concepts from Halstead Complexity to firewall configuration. We model configuration commands as operators and the corresponding parameters to the commands as operands. For example, in Cisco IOS, the following commands configure IP addresses for name servers:

```
ip name-server 1.2.3.4
ip name-server 1.2.3.5
ip name-server 1.2.3.6
```

In this case, the operators are the commands "ip" and "name-server", and the operands are the IP addresses "1.2.3.4", "1.2.3.5" and "1.2.3.6". The Program Vocabulary size and the Program Volume measure would increase when additional name servers are configured. Recall the packet filter "101" from the last section:

```
access-list 101 deny ip 10.0.0.0/8 any
access-list 101 deny ip 127.0.0.0/8 any
access-list 101 deny ip 192.168.0.0/16 any
access-list 101 permit any
```

The command "access-list" is a keyword in IOS and are the operators, while the parameters "101", "deny", "permit", "ip", "any" and the IP addresses (prefixes) are the operands. The parameter "101" is the name of this packet filter. Packet filters such as the one above are usually copied across multiple firewalls

in a network. When a packet filter is copied verbatim to another firewall the network-wide Program Volume measure increases but the Program Vocabulary size does not. However, if for whatever reasons, the name of the packet filter changes during copying but the implementation remains the same, the Program Vocabulary size increases because of the extra name, which demands additional mental capacity from the network administrator.

3.2 Structural Complexity

One of the weaknesses of the Halstead Complexity is that it only considers the lexical complexity of source code. Cyclomatic Complexity measures the structural complexity of source code, and is used together with Halstead Complexity to determine the maintainability of software. Cyclomatic Complexity calculates the number of linearly independent paths in source code. It is calculated as $CC = E - V + 2p$, where E is the number of edges, V is the number of nodes, and p is the number of connected components, and the source code is modeled as a connected graph of control flows. *CC* is traditionally used as a static analysis tool to assess the complexity of source code during different phases of software development, and the risks involved in modifications of the source code afterwards.

We borrow concepts from Cyclomatic Complexity to measure the complexity of a network's firewall configurations. We first model a network's firewall configurations as flow graphs. A flow graph is a directed graph $G=<V,E,R>$, in which V is the set of vertices, E is the set of edges, and R is a set of functions that annotate the edges. A vertex v is associated with a rule. There is an edge e from *v1* to *v2* if there is a flow of control along the rules from *v1* to *v2*, and a function r annotates e with the set of packets that can flow on e. Note that no explicit flow of control exists between the configurations of two firewalls. Firewall configuration is a device-specific task. Thus, when applied to a network with multiple firewalls, a flow in our model is an implicit flow of control. In our model, there is a flow of control between two vertices if the following conditions are satisfied: (1) the set of packets filtered by *v1* and the set of packet filtered by *v2* intersect, and (2) the vertices belong to the same packet filter or there exists a path between the two vertices according to the topology of the underlying network. Intuitively, a flow of control exists between two vertices if the administrator needs to form mental pictures of the relationship between the corresponding rules, for example, if the order of execution of the corresponding rules changes the resulting packet flow.

We first illustrate Cyclomatic Complexity with a single firewall example. Recall the "101" packet filter from the previous section:

```
access-list 101 deny ip 10.0.0.0/8 any
access-list 101 deny ip 127.0.0.0/8 any
access-list 101 deny ip 192.168.0.0/16 any
access-list 101 permit any
```

The source IP addresses ("10.0.0.0/8", "127.0.0.0/8", "192.168.0.0/16") are disjoint. The destination IP addresses ("any") are wildcards. Thus, in our model, there is no flow of control for the "101" packet filter as the sets of packets filtered by each rule do not intersect. The following packet filter "201" has a flow of control according to our model because the packets being filtered by the first rule is a subset of the second rule:

```
access-list 201 deny tcp 192.168.1.0/24 any
access-list 201 accept tcp 192.168.0.0/16 any
```

If we swap the order of the two rules, all packets in 192.168.0.0/16 will be allowed.

The following example illustrates Cyclomatic Complexity of an enterprise network with two firewalls. This enterprise network owns the IP address space 1.2.0.0/16, and its internal corporate network utilizes a subset, 1.2.3.0/24. The first firewall *F1* acts a simple screening device. It denies access to the internal network but allows access to public servers (e.g. Web, Mail) in the DMZ:

```
access-list 301 deny tcp 1.2.3.0/24 any
access-list 301 accept tcp any any
```

F2 protects the internal corporate network and denies everything:

```
access-list 401 deny tcp 1.2.0.0/16 any
```

Even though the individual packet filters do not have a flow of control within them, there is a flow of control between the packet filters "301" and "401", since "301" denies a subset of the packets denied by "401". If we swap the first rules of the two packet filters, access to the public servers will be denied, for example.

4. WHERE THE COMPLEXITY LIES

In this section, we apply the lexical and structural complexity models to packet filters defined in router configuration files from production networks to examine where the complexity lies in real life. We present our preliminary results here.

4.1 Data

Our data comes from a university campus network (*NETU*). We also use knowledge about the best common practices employed by security administrators. *NETU* has more than 50 routers. We only focus on two border routers and two core routers which implement most of *NETU*'s firewall functions.

Because of the nature of *NETU*, it performs limited packet filtering. We suspect the degree of complexity is magnified in enterprise networks. Nonetheless, we are examining the complexity of the firewall configuration language in general, but not of particular networks, so our results are still valid.

4.2 Observations

One main source of lexical complexity is the use of IP addresses (and prefixes) in configurations. There can be many IP addresses in one packet filter. Usually, the list of IP addresses in a packet filter is a logical unit, either in the network protected by the firewall(s) or in the Internet. For example:

- *NETU* owns eight public address blocks. *NETU* defines two packet filters, *p1* and *p2*, each matching all of the eight prefixes. *p1* prevents packets with spoofed source IP addresses from leaving *NETU*'s network. *p2* allows only packets destined to IP addresses in one of the eight address blocks into *NETU*'s network.

- *NETU* also defines two packet filters to prevent packets with reserved IP addresses from leaving or entering *NETU*'s network. There are four private address blocks, and these prefixes are listed in both packet filters.

As *NETU* is multi-homed to two service providers, the above packet filters are duplicated on two routers. The Program Volume reflects the number of times IP addresses are duplicated in the configurations. Often, configurations are modified via "cut-n-paste", and an administrator needs to ensure all IP addresses are

duplicated appropriately during modification. The Program Vocabulary reflects the number of IP addresses an administrator needs to remember. In fact, since our model treats each unique IP address as one unit of vocabulary, it yields a conservative measurement of lexical complexity. This is because an IP address, more precisely, a prefix, can embody multiple addresses. For example, 1.2.3.0/24 represents 256 addresses.

Another source of lexical complexity is assigned names. Firewall administrators assign names to packet filters. When used properly, an administrator remembers and uses these names instead of the details of the packet filters. However, there are several ways assigned names can add to the mental model burdens of administrators. Within a single firewall, there is usually more than one packet filter, each assigned different names. But:

- Some of the packet filters may not be used by the firewall. A packet filter is active only when it is applied to one of the interfaces of the firewall. The Program Vocabulary reflects the total number of names, but not all the names are necessary.
- Differently named packet filters can be equivalent. This happens when two packet filters are exactly the same except for their names, or that they use different rules to achieve the same function. The Program Vocabulary again reflects the total number of names, which can be reduced if there is only a single packet filter.

Across multiple firewalls, packet filters can be assigned the same names. There are several possible scenarios. Two packet filters with the same name, can implement the same function on multiple firewalls, can implement different functions on multiple firewalls, or can have functions that intersect one another. For each packet filter named *p*, an administrator either needs to remember the meaning of *p* on each firewall that defines *p*, or reconstruct its meaning by reading *p*'s rules in low-level configuration commands. Sometimes, packet filters that are equivalent, either syntactically or semantically are given different names on different firewalls. Again, an administrator needs to either remember that these packet filters are indeed equivalent, or reconstruct this information on-the-fly.

We also observe structural complexity in *NETU*'s configurations. Recall that *NETU* owns eight public address blocks. We have described two packet filters that match on these blocks. Additional packet filters that match on subsets of these blocks are defined on multiple routers in *NETU*. For example:

- *NETU* has HTTP, SMTP, DNS and NTP servers, each of which is assigned a public address from one of the eight public address blocks. Packet filters are defined on the border routers and applied to the outgoing traffic from these servers so that the traffic is sampled for accounting purposes. Other packet filters are defined on the core routers and applied to the incoming traffic to these servers to allow for port exceptions for Microsoft services.
- *NETU* restricts Virtual Private Network (VPN) access to only certain address pairs. A packet filter is defined and applied to both inbound and outbound VPN traffic.
- Residence hall users are rate-limited to use a certain amount of bandwidth. Packet filters are defined that match the address blocks belonging to the residence halls.

Recall that structural complexity measures the number of paths in configurations. In our model, there is an edge between two vertices if the packets matched by the rules represented by the vertices intersect. We have described several cases above in which this intersection happens in *NETU*. There can be many chains of packet filters in a network, and an administrator needs to ensure each one of these chains yields the intended set of packets during the configuration and modification each of the packet filters along the chain. Furthermore, packet filters can be applied on the inbound or outbound direction, and an administrator also needs to keep a mental model of the network topology and its interaction with all the packet filters and the direction of their applications.

Note that not all uses of packet filters are directly security related. Some packet filters are used to identify packets for accounting or traffic engineering. In general, an administrator needs to juggle several additional mental models besides security when working with packet filters.

5. USABILITY SUGGESTIONS

In this section, we suggest ways to improve the usability of firewall configuration. Our suggestions follow three simple design principles:

- Treat CLI as the main user interface. Administrators prefer to use CLIs and distrust GUIs. We make the assumption that administrators do most of their configuration writing and modification with CLI, and use other tools as support.
- Keep the existing configuration languages. Although these languages are full of problems, in the short term, administrators are reluctant to learn new languages and device vendors do not see the incentive to deploy them.
- Provide visualizations for the most complex parts of a network's firewall configuration. These parts are either in design documents or mental images in an administrator's head. Design documents are often not updated promptly and following them can be misleading and lead to more errors.

Our complexity models show that four aspects of firewall configuration are especially complex. The four aspects are (1) IP addresses, (2) packet filter names, (3) interfaces, and (4) packet filter interactions. Below, we discuss ways to visualize each of the four aspects and how to tie the information together in a unified way. The goal is to help administrators build mental images of the most complex parts of his network's firewall configurations, while allowing him to continue using CLI as the main user interface. The visualizations gather information that are scattered across the network and present them to the administrator in a context appropriate way.

5.1 IP Addresses

When configuring the source and destination IP addresses in a packet filter, an administrator is clear on his intent for the addresses. For example, the addresses can represent internal subnets, private address space, known malicious networks, and so on. Visualizations can fill in the details that the administrator

may not always remember. The same addresses can be in packet filters within the current firewall and firewalls across the networks. Moreover, prefixes that cover the addresses can also exist in other packet filters. Showing these related addresses gives an administrator a global picture of how his network treats the addresses in question. The following information can be visualized:

- Packet filters matching the exact IP address, whether it is on the same firewall or on another firewall. This information helps the administrator to see if he can "cut-and-paste" configuration commands from elsewhere, or if the function he is trying to implement already exists.
- Packet filters matching a superset of the IP address. This information if exists show that the IP address in question is being treated as an exception.
- Packet filters matching a subset of the IP address (prefix). This information reminds the administrator if there are exceptions in network regarding the IP address (prefix) in question.

There are potentially a large number of related IP addresses. IP addresses are not discrete information. They are logically organized in a tree. Existing techniques for visualization information in a tree can be used to effectively and efficiently display the IP address hierarchy.

5.2 Names

In the ideal case, a packet filter is defined only once, maintained by a central repository and applied to a firewall or multiple firewalls. In this way, a packet filter that is meant to be the same on multiple firewalls will remain the same even after modifications. The more sophisticated enterprise networks use a database to maintain their configurations. However, researchers have observed that such a database is usually out-of-date, so in reality "the network is the database" [4]. Visualizations can help administrator handle this complexity by displaying:

- Packet filters with the same name and highlight the differences.
- Packet filters with similar names and highlight the differences. For example, "bogon" and "Bogon" are similar names, and "internal1" and "internal" are also similar names. Configurations can be manipulated by several administrators, each with different coding styles.

Displaying these names in alphabetical order might not be the most usable to administrators. Some packet filters are more relevant than others, depending on the context. Instead, names can be listed in the order of topological closeness to the firewall being configured. In other words, names that are within the same firewall are listed first, then the names that are on firewalls one hop away next, and so on. Also, some packet filters are not actively in use. It is possible to have many definitions of packet filters on a firewall, but only a handful of them are active. The names of these packet filters can be displayed last because they are not as important to the administrators.

5.3 Interfaces and Interactions

A packet filter is not active unless it is applied on an interface of the firewall being configured. However, interface definitions are separate from packet filter definitions in the configurations. Visualizations can show the details of the interface in question when an administrator is examining a packet filter so he does not need to manually search for the details.

Interfaces are connected together which forms the physical topology of a network. Packet filters can potentially interfere with one another if the packets they match intersect and if the underlying topology allows these filters to be applied in sequence. Visualizations can show these interactions explicitly.

5.4 Information Linking

A network can contain many IP addresses, names, interfaces, and interactions in its firewall configurations. Visualizations should only display information that is relevant to the task at hand. In their study on visualizing large-volume time-series data for network management, McLachlan et al [12] found that explicit linking is effective in coordinating multiple views of related information. The information in Section 5.1-5.3 are should be tied to the main CLI. When an administrator is working on a specific packet filter of a particular firewall, the relevant information about IP addresses, names, interfaces and interactions are explicitly linked in side windows. For example, as an administrator steps through each rule in a packet filter using a text editor, the IP addresses window is updated for each rule.

6. RELATED WORK

Many researchers have proposed solutions to the firewall configuration problems. This paper differs in that we tackle the problems from a usability perspective. We suggest ways to improve the usability of writing and modifying configurations based on models that measure lexical and structural complexity of firewall configuration. Our goal is to support and augment the CLI with appropriate visualizations.

As far as we know, Geng, Flinn and DeDourek [5] is the only paper on usable firewall configuration. They elegantly state the problems with current firewall configuration, and propose a simulation tool to allow administrators to visualize network traffic flow.

PolicyVis [17] is a visualization tool for firewall packet filters. It has a query-based GUI which allows administrators to enter parameters of interest (i.e. source and destination IP addresses and port numbers) to generate a visualization of the packets being permitted or denied by the configured filters.

The authors of Fang [13] propose a query-and-answer system that allows administrators to understand deployed firewall packet filters in a network. Fang is easy to use as it works at a high level of abstraction, allowing administrators to ask questions such as "what machines can reach which services on my network?" on deployed configurations.

Firmato [1] is a complete firewall management solution. It defines a model to specify security policy and network topology, a high-level language to specify an instance of the model, and a translator from such an instance to low-level firewall configuration commands.

The authors of EDGE [4] argue that router configuration is hard and manual configuration of routers should be replaced by automated provisioning. EDGE is a system that analyzes existing

network state and uses the results to fill a database with information for future configuration changes.

Many have proposed methods to detect misconfigurations in firewalls. Most are rule-based checkers based on best common practices or well-known security vulnerabilities [15, 16, 20]. FIREMAN [19] models packet filters in firewalls using binary decision diagrams, and detects redundant and conflicting rules that can be misconfigurations.

There are many tools for visualizing real time network traffic going through a firewall [2, 6, 10]. One goal is to detect network attacks (DDoS, worms, viruses) visually. Another goal is to see if the firewall is properly configured (e.g. by examining visually if certain traffic is being allowed by the firewall unintentionally). The problem with the latter goal is one cannot visualize the complete firewall configuration by examining only real time network traffic. Thus, not all misconfigurations can be detected, only the ones that are triggered and present in the traffic.

7. CONCLUSIONS

In this paper, we have proposed complexity models to measure the configuration complexity of firewalls. We have observed that the main source of complexity in firewall configuration is in the configurations of IP addresses, names, interfaces and interactions between firewalls. Making the assumption that administrators prefer to use CLI as their main user interface, we have described several visualizations to support with the firewall configuration task. For future work we plan to prototype the visualizations and evaluate them with user studies.

ACKNOWLEDGMENTS

We would like to thank the networks who gave us access to their router configuration files. This work would not have been possible without their help.

REFERENCES

[1] Y. Bartal, A. Mayer, K. Nissim and A. Wool. Firmato: A Novel Firewall Management Toolkit. ACM Transactions on Computer Systems, 2004.

[2] E. Bethel, S. Campbell, E. Dar, K. Stockinger, and K. Wu. Accelerating Network Traffic Analysis Using Query-Driven Visualization. In Proceedings of IEEE Symposium on Visual Analytics Science and Technology. 2006.

[3] D. Botta, R. Werlinger, A. Gagne, K. Beznosov, L. Iverson, S. Fels, B. Fisher. Towards Understanding IT Security Professionals and Their Tools. In Proceedings of Symposium On Usable Privacy and Security (SOUPS), July 2007.

[4] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford. The Cutting EDGE of IP Router Configuration. In Proceedings of HotNets-II, 2003.

[5] W. Geng, S. Flinn, and J. DeDourek. Usable Firewall Configuration. In Proceedings of the Third Annual Conference on Privacy, Security and Trust. Oct 2005.

[6] J. Goodall, W. Lutters, P. Pheingans and A. Komldi. Preserving the Big Picture: Visual Network Traffic Analysis with TNV. In Proceedings of Workshop on Visualization for Computer Security. Oct. 2005.

[7] E. Haber and J. Bailey. Design Guidelines for System Administration Tools Developed through Ethnographic Field Studies. In Proceedings of Computer Human Interaction on Management of Information Technology (CHIMIT), Mar. 2007.

[8] M. H Halstead. Elements of Software Science, Operating, and Programming Systems Series Volume 7. New York, NY: Elsevier, 1977.

[9] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[10] C. Lee, J. Trost, N. Gibbs, R. Beyah, and J. Copeland. Visual Firewall: Real-time Network Security Monitor. In Proceedings of Workshop on Visualization for Computer Security. Oct 2005.

[11] T. McCabe and C. W. Butler. Design Complexity Measurement and Testing. Communications of the ACM 32, December 1989.

[12] P. McLachlan, T. Munzner, E. Koutsofios, and S. North. LiveRAC: Interactive Visual Exploration of System Management Time-Series Data. In Proceedings of CHI, Apr. 2008.

[13] A. Mayer, A. Wool and E. Ziskind. Fang: A Firewall Analysis Engine. In Proceedings of IEEE Security and Privacy, May 2000.

[14] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? 4th USENIX Symposium on Internet Technologies and Systems (USITS '03), March 2003.

[15] Router Security Configuration Guide. System and Network Attack Center, National Security Agency, 2003. Available at http://www.nsa.gov/snac/routers/cisco scg-1.1b.pdf

[16] The Router Audit Tool (RAT). http://www.cisecurity.org/bench cisco.html.

[17] T. Tran, E. Al-Shaer, and R. Boutaba. PolicyVis: Firewall Security Policy Visualization and Inspection. In Proceedings of 21st Large Installation System Administration Conference, Nov. 2007.

[18] A. Wool. A Quantitative Study of Firewall Configuration Errors. IEEE Computer, June 2004.

[19] L. Yuan, J. Mai, Z. Su, H. Chen, C.N. Chuah and P. Mohapatra. FIREMAN: A Toolkit for FIREwall Modeling and Analysis. In Proceedings of IEEE Security and Privacy 2005.

[20] D. Zerkle and K. Levitt. NetKuang – A Multi-Host Configuration Vulnerability Checker. In Proceedings of USENIX Security Symposium, Jul 1996.