A Probing Technique for Discovering Last-Matching Rules of a Network Firewall

[1]K. Salah [2]K. Sattar [3]M. Sqalli [4]Ehab Al-Shaer
[1,2,3]*Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia,* Email: {salah,karimas,sqalli}@upm.edu.sa
[4]*School of Computer Science, Telecommunication, and Information Systems, DePaul University, Chicago, Illinois 60604, USA,* Email: ehab@cs.depaul.edu

Abstract

In this paper we identify a potential probing technique for remotely discovering the last-matching rules of the security policy of a firewall. The last-matching rules are those rules that are located at the bottom of the ruleset of a firewall's security policy, and would require the most processing time by the firewall. If these rules are discovered, an attacker can potentially launch an effective low-rate DoS attack to trigger worst-case or near worst-case processing, and thereby overwhelming the firewall and bringing it to its knees. As a proof of concept, we developed a prototype program that implements the detection algorithm and validated its effectiveness experimentally.

1. Introduction

A network firewall is typically the first line of defense for a private network against attacks originating from the Internet. A firewall itself can become a target of Internet DoS (Denial of Service) and DDoS (Distributed DoS) attacks, and thereby jeopardizing its effectiveness to filter in and out traffic. If the firewall is under attack, then the whole network becomes open for further attacks and it will not provide services to the legitimate users for which this network has been built. In this paper we identify a potential probing technique for discovering the last-matching rules of the security policy of a firewall. The last-matching rules are those rules that are located at the bottom of the ruleset of the firewall's security policy.

In general, the firewall policy consists of a list of rules, with each rule representing a set of conditions. If an incoming packet matches all conditions of a particular rule, then a certain action is taken. Examples of an action can be to allow the packet to pass or drop it immediately. If a rule is matched, the firewall matching engine skips the remaining rules. A packet can match the conditions of more than one rule. In such a case, the first rule will have priority and its action (i.e., allow/reject) will be applied to the packet. Thus, *logically*, the firewall checks these rules sequentially, one by one, till a rule is matched. There are a number of techniques to optimize and speed up this sequential matching operation of a firewall. These often involve building a tree-like ruleset with chains, and also positioning the most frequently accessed rules at the beginning of the ruleset [1-3].

Regardless of whether optimization is used or not for packet filtering, there will always be default rules and last-matching rules which consume considerable processing times compared to others. A firewall of a large-size enterprise network can typically have a ruleset or ACL (Access Control List) comprised of 20,000 rules or more. Typically, default rules and last-matching rules are positioned at the bottom of the entire ruleset or at the bottom of a rule-tree branch or chain. Discovering how to trigger these rules by an outside attacker can be disastrous. The attacker can then launch a DoS attack consisting of a single flow of packets to trigger these last-matching rules which require the most CPU cycles of the firewall. And therefore with a low-rate flow of these packets, the cost of processing and filtering at the firewall becomes so high that could potentially result in overwhelming the firewall and bringing it to its knees.

Such an attack against firewalls can be classified as a complexity-algorithm attack [4] whereby the attacker is able with a low-bandwidth DoS flow to trigger worst-case or near worst-case processing. Unlike traditional DoS attacks which are commonly based on sending packets with the highest rate possible, this DoS attack (crafted particularly against firewalls) is designed to be sent with *low* rate, but would be so effective in causing the attacked firewall to perform very poorly. What is more crucial is that such a low-bandwidth DoS attack can subvert existing mitigation techniques used by intrusion detection and prevention systems (IDS and IPS). Existing IDS and IPS appliances can detect a suspicious high-rate flow and then subject the flow to traffic shaping and throttling. However, a flow with a low rate can potentially bypass such appliances, and consequently jeopardizing the network security. Detection can even be more subverted when multiple flows are launched with much lower rates from an army of zombie machines as the case of today's botnet DDoS attacks.

The rest of the paper is organized as follows. Section 2 details the probing technique to discover the last-matching rules of a network firewall security policy. The algorithm, key issues, and formalization of the probing technique are presented. Section 3 presents a proof-of-concept experiment to validate the probing and detection technique, where different probing parameters and environment are considered. Section 4 presents related work found in the literature with regards to remote discovery of last-matching rules of a firewall. Finally, Section 5 concludes the study and identifies future work.

2. The Probing Technique

In a typical enterprise network, as illustrated in Figure 1, a firewall is deployed with a tri-homed configuration to pass or deny incoming and outgoing traffic between three network segments (Viz., Internet, Demilitarized Zone (DMZ), and

978-1-4244-3397-1/08/$25.00 ©2008 IEEE

private or enterprise LAN) [5]. The DMZ contains publicly accessible servers such as DNS, FTP, Web, and E-Mail. The key idea behind our probing technique of remotely discovering the last-matching rules of a network firewall is to send a number of back-to-back probing packets or a probing packet train (PPT) and then measure its length changes or train stretch. The train stretch would increase depending on the amount processing encountered at intermediate nodes, specifically at the firewall. A PPT length would stretch out if the probing packets are faced with significant processing by the firewall engine, and therefore reflecting the discovery of the default or one of the last matching rule.

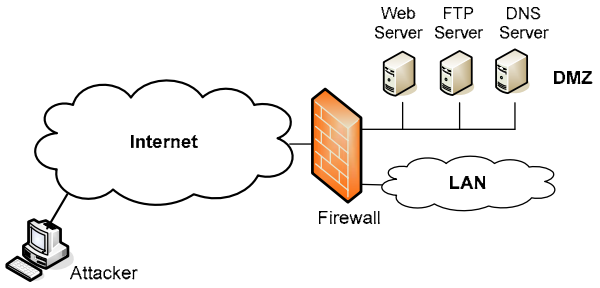

Figure 1. Tri-homed deployment of an enterprise firewall

2.1 The Algorithm

In order to measure the stretch of the train, we will send a non-cacheable HTTP request just right after the PPT, as illustrated in Figure 2. The HTTP request will be destined to the web server in the DMZ. Both HTTP request and reply are *always* configured to pass the firewall. The time between sending the HTTP request and receiving the reply will reflect the PPT length. Throughout this paper, we use a web server to measure the PPT length; however other servers in the DMZ such as FTP or E-mail can be equally used. In order to make sure that our HTTP request will not be cached locally or at some intermediate node, the technique requires a non-cacheable HTTP request to be sent. Therefore we ascertain that the reply comes from the original web server behind the targeted firewall. Non-cacheable HTTP requests typically include those requests with headers that contain authentication or cache-control directives, or having methods such as OPTIONS [6,7]. In our experimental work, we used an HTTP request with methods of OPTIONS which queries all possible methods supported by the web server.

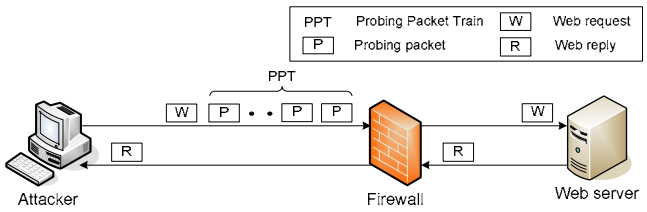

Figure 2. Probing packet technique using a web server

The details of one round of our PPT probing technique are depicted in Algorithm 1. To determine the position of a particular rule in the ruleset, multiple probing packets of the same type are crafted (as shown in Line 1) for that particular rule. The number of the probing packets in the train is determined by "PPT size". The probing packet is crafted by setting the proper fields for protocol type, source port and address, and destination port and address. If the targeted rule is using TCP, then a TCP connection has also to be first opened. In order to send HTTP requests, a TCP connection is opened (as shown in Line 2). Then the PPT is constructed by sending back-to-back probing packets (as shown in Line 4-7). The count of packets in the PPT is configurable per probing round. The PPT delay stretch is determined by the difference in time between sending the HTTP request and receiving the reply back from the web server (as shown in Line 8-12).

Algorithm 1 ProbingRound (*PPT* size)

```
1  P ← ProbingPacketType(protocol, srcIP, srcPort, destIP, destPort)
2  Open TCP connection with the web server
3  count ← PPT size
4  Repeat
5      Send P;
6      Decrement count;
7  until (count =0)
8  t1 ← GetTimeStamp();
9  Send W; // send HTTP request
10 Wait(); // wait only for HTTP reply and ignore other possible replies
11 t2 ← GetTimeStamp();
12 delay ← t2 – t1;
13 return delay
```

2.2 Properties and Requirements

For the probing technique to be effective, a smart attacker may consider a number of key and practical issues in order to remotely discover the default and last matching rules of a network firewall. These key issues and requirements of the probing technique to be considered are as follows:

1) For each targeted rule, the firewall may react to probing packets differently based on its rule policy. For example, if the rule action is to accept the packet and pass it through, it is possible that replies could come back from network nodes to the probing machine of the attacker in response to those probing packets. In such a case the attacker may ignore all of these replies, and only the HTTP reply is counted (as shown in Line 10 of Algorithm 1).

2) Prior to sending probing rounds, the attacker has to determine the baseline of the time that it takes to send an HTTP request and receive its reply, i.e., a PPT with zero probing packets (PPT size =0). The baseline of the time should be the minimum time of a set of rounds. After this the attacker can start propping rounds (with PPT size >0) targeting different rules and measure the PPT stretch. Only a noticeable PPT stretch that is larger than the minimum baseline would be of an interest to the attacker.

3) The PPT stretch can be affected by many other factors than processing of rules by the firewall. These factors may include the proximity of the attacker machine to the firewall, the load conditions due to network traffic at the links on the path, and the load conditions due to background processing at intermediate nodes, attacker machine, firewall, and web server. To compensate for such fluctuation, the attacker could use more than one probing round (if a noticeable stretch in the PPT is attained) until a consistent minimum PPT stretch delay is observed.

4) The enumeration process of targeting different rules to determine their position in the ruleset can be done exhaustively, randomly, or selectively. Rather than using exhaustive or random enumeration, the attacker could enumerate rules more intelligently by making use of common guidelines, patterns, and best practices for writing firewall security policies [8,9]. And therefore enumeration is narrowed down only to a set of rules of interest which are most likely to be the default and last-matching rules. Also, the authors of [3] list few sampling guidelines that can help the attacker to select the probing packets.

5) Too many probing rounds with a large number of probing packets in the PPT can be a sign of an anomaly which can be detected by an IPS. Still, the attacker may probe in a stealth fashion whereby he can disperse the probing rounds over a long period of time with a small-size PPT. The impact of the size of PPT on the probing technique performance (in terms of observed PPT stretch) will be studied experimentally in Section 3.

2.3 Formalization of PPT Delay

We further characterize the probing technique to understand better the impact of the firewall filtering engine on stretching the train length. In particular we formalize the probing technique and express the total delay seen by the attacker due to processing of rules at the firewall.

Let

P denote the total number of probing packets included in the PPT (i.e., the PPT size),

$1/\mu_i$ denote the average processing or filtering time per rule i serviced for a probing packet p,

n denote the position of the rule that triggers an action by the firewall for packet p, and

$1/\lambda_p$ denote the average interarrival time of probing packets seen at the firewall.

As the packet arrives at the firewall, it gets processed sequentially until it triggers a certain rule at position n in the ruleset. Hence, the average accumulative processing time of a probing packet by the firewall engine, denoted by T, can be expressed as

$$T = \sum_{i=1}^{n} 1/\mu_i$$

This delay T will impact or stretch the length of PPT *if and only if* it causes the next probing packet to be buffered at the firewall. A buffering delay of the next probing packet occurs when the firewall is still busy processing the first probing packet, i.e., when T is larger than the average interarrival time $1/\lambda_p$. Otherwise, the PPT length will not be stretched. Therefore, the average stretch delay D_p introduced by one probing packet can be expressed as

$$D_p = \begin{cases} T - 1/\lambda_p & \text{if } 1/\lambda_p < T \\ 0 & \text{otherwise} \end{cases}$$

Therefore, the average total delay seen by the attacker depends on the number of probing packets in the train (i.e., PPT size) and can be expressed as

$$D_{PPT} = \sum_{p=1}^{P} D_p$$

3. Proof-of-Concept Experiment

As a proof of concept and to validate and study further our probing technique, we set up an experiment comprised of three Linux machines: attacker, firewall, and web server (as shown in Figure 2). The firewall has two Ethernet network interface cards with one card connected to the attacker machine and the second card connected to the web server. The connection is made over 1 Gbps crossover Ethernet crossover cables. The three machines are all Intel Pentium 4 processor running at 3.2 GHz with 512 MB of RAM. The network cards are 3COM Broadcom NetXtreme Gigabit Ethernet with BCM5752 controller. All machines are running with Fedora Core 5 Linux 2.6.15 and with the default tg3 NIC device driver.

We setup the firewall with Linux *Netfilter* and we created a ruleset using *iptables* commands that includes rules for web traffic as well as 10,000 other dummy rules. We configured the Linux *Netfilter* to accept and pass web traffic, but to drop probing packets. Rules related to web traffic were positioned at the beginning of the firewall's ruleset. The other 10,000 dummy rules were created without chains using a shell script of *iptables* command with each rule having its own same conditions of "*any*" except for the source MAC address. We used a condition for source MAC addresses since they were experimentally found to be computationally more expensive. This is so because the MAC address is one of the last conditions to be checked by *Netfilter* within a rule [10]. We obtained an average processing time per rule of 0.1 μs. We also determined that the average time to process the whole ruleset of 10,000 rules was approximately 1 ms. We measured this by instrumenting Linux code with timestamps. For timestamps, we used rdtscl macro which basically implements the assembly instruction of rdtsc (read time stamp counter) which returns the number of CPU cycles since system bootup.

We developed a prototype program in C that implements the probing algorithm shown in Algorithm 1. From the attacker Linux machine, we ran the probing program with PPT size of 1, 4, and 10 such that PPT probing packets target different rules placed at different positions in the ruleset. In particular, we targeted rules at positions 1, 500, and 10,000. Fig. 3 illustrates the results obtained for the average PPT stretch observed by the probing program when running five probing rounds. There are several important observations that can be made from the figure. First, the last-matching rules indeed incur more delays regardless of the PPT size. Second, the PPT size plays an important role in detecting (or amplifying) the processing delay incurred at the firewall, as shown in Fig. 3(a). Probing delays with PPT size of 10 and 4 are more noticeable than those with PTT size of 1. Third, the amplification of the processing delay has almost linear relationship with the PPT size. Fig. 3(a) shows that the measured stretch with PPT size of 4 is four times more than the stretch with PPT size of 1. Similarly, the measured stretch with PPT size of 10 is ten times more than the stretch with

PPT size of 1. Fourth, the fluctuation or range from the average point of the measured stretch or delay of probing rounds (shown with the red vertical bar, and clearer in Fig. 3(b)) depends primarily on PPT size. For example, the fluctuation is less with PPT size of 10 than 4. In summary, we can conclude that a relatively large PPT size would minimize fluctuation in PPT stretch and also amplify the processing delays encountered at the firewall, and thus simplifying the task of the attacker in detecting the position of these rules.

We next consider the impact on PPT stretch when considering a number of changes to our experimental setup which include: (1) running the probing program from a Windows machine, (2) probing from distant location with an attacker machine of a limited-bandwidth uplink, and (3) having a multiprocessor firewall. Figure 4(a) shows that probing from a Windows machine gives similar results as those with Linux, but with more and inconsistent fluctuations. Figure 4(b) shows that probing (regardless of the PPT size) from a machine with a limited bandwidth such as an ADSL uplink of a bandwidth of 56Kbps will not will be successful. This is so because the interarrival time of the probing packets will be larger than the processing time (of a maximum of 1ms) of the firewall. This is in line with our formalization in Section 2.3. However, probing from a machine with a fast 100 Mbps Ethernet uplink will show similar impact as that of Figure 3(b). The results of Figure 4(b) for 100 Mbps uplink are shown when probing with a PPT size of 4. It is to be noted that unlike other cases, the probing with the 100 Mbps uplink Linux machine was preformed from the University student dormitory located at the edge of our University campus network which has a diameter of 6 kilometers. Lastly, we found out experimentally that using a firewall with multiprocessor or single processor makes negligible difference in PPT stretch. The reason is that Linux *Netfilter* processing is single threaded and does not distribute processing evenly between the available processors.

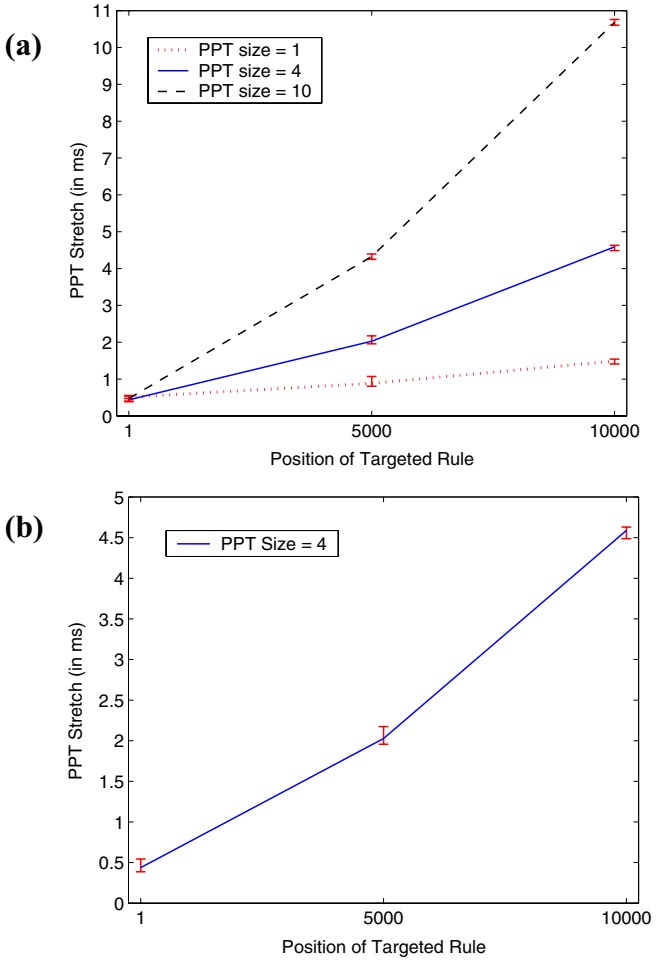

Figure 3. Probing from Linux machine

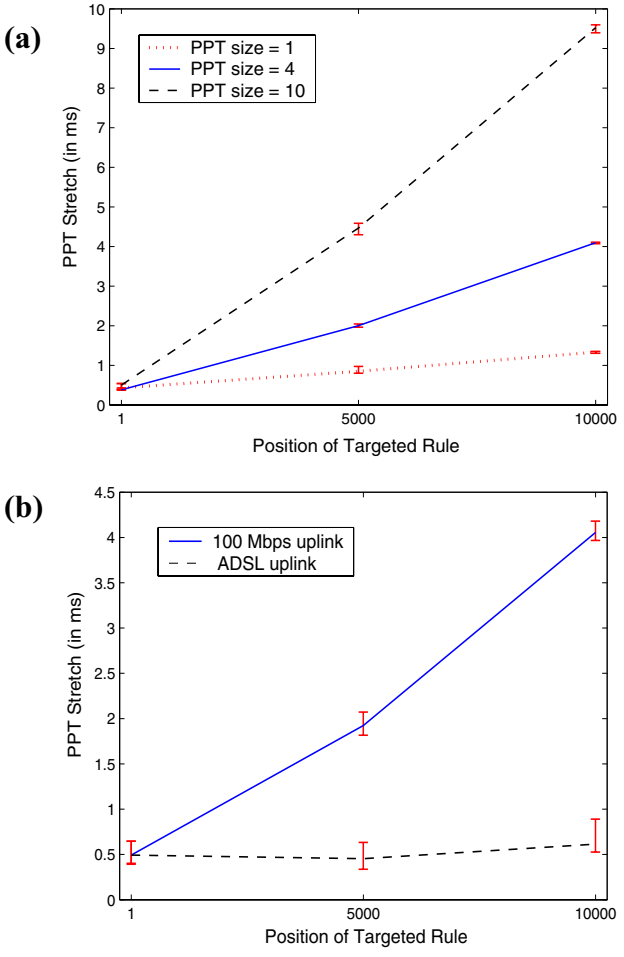

Figure 4. Impact of (a) probing from Windows, (b) location and uplink rate

4. Related Work

There exists little work in the literature with regards to remotely discovering security policy of a firewall or at attacking network firewalls using low-bandwidth DoS attacks. The closest related work with regards to *remotely* discovering security policy of firewall was reported by Samak et al. [3]. In [3], the attacker uses two simple agents: a "Pinger" and "Ponger". The Pinger is located outside the private network and the Ponger is located behind the firewall inside the private network. It is required that the Ponger be pre-compromised, and that both Pinger and Ponger be of a close proximity to the firewall in order to minimize the delay jitter. The primary task of the Pinger is to send crafted probing packets to the Ponger. The Ponger will receive these packets and send them back to the Pinger. According to [3], the security policy and default rules can then be deduced based on the packets received and not received, and also by examining the delays of received packets.

There are a number of shortcomings of the probing technique reported in [3] when compared to ours, and thus making our technique superior and more realistic. First, unlike Samak et al.'s technique, our PPT probing technique does not require pre-compromising a machine inside or outside the network. We use two machines; one for the attacker outside the private network, and one (e.g., web or FTP server) already located inside the network and with very close proximity to the firewall, i.e., located right behind the firewall in the DMZ. Second, delay measurements of probing packets received by the Ponger are impractical as the clock of the Pinger and Ponger are not synchronized. To compensate for this, both Pinger and Ponger need to run NTP clock synchronization protocol, with regular updates. Third, Samak et. al.'s technique requires that the Pinger send all crafted probing packets to the Ponger. This way, proper conclusions can be made about filtering rules and positions of the rules. This is a major flaw and limitation in their technique, as it only depends on the "accept" rule space with packets destined only to the Ponger. So their technique can not draw proper conclusions about the "deny" rule space or "accept" space destined to other hosts within the protected network. Fourth, Samak et al.'s work does not address the issue of delay fluctuation in the network. In our PPT technique a *train* of packets made it possible to compensate for small variations in one single probe trial and also to magnify small processing delays at the firewall.

5. Conclusion and Future Work

We presented and discussed a potential probing technique that can be used by an attacker to remotely discover the last-matching rules of a network firewall. An attacker can then target these rules to launch an effective and low-rate DoS attack to trigger worst-case or near worst-case processing, and thereby overwhelming the firewall and bringing it to its knees. We studied and validated the probing and detection technique experimentally. In our experiment, we studied important probing parameters and considered different factors which included the probing platform, location, and uplink bandwidth. Future research work may include the following:

- Targeting a different type of firewalls other than Linux *Netfilter* such as CISCO PIX or FreeBSD Packet Filter
- Experimental performance evaluation and comparison of traditional DoS attacks with this new type of intelligent DoS attacks that only target last-matching rules. An important issue to investigate is the required rate to degrade significantly the firewall performance.
- Impact on probing measurements when varying load conditions of network links and network hosts including the firewall, server, and attacker machine.
- Investigation of possible solutions and countermeasures for subverting such a probing technique, and also detecting and recovering from these low-rate types of DoS attacks.

Acknowledgments

We acknowledge the support of King Fahd University of Petroleum and Minerals in completion of this work. This work has been funded under Project #INT-361. We also thank Taghrid Samak and Adel El-Atawy for their valuable replies and comments to some of the issues faced throughout this work.

References

[1] A. El-Atawy, T. Samak, E. Al-Shaer, and H. Li, "Using Online Traffic Statistical Matching for Optimizing Packet Filtering Performance," Proceedings of the 26th IEEE INFOCOM'07, Anchorage, Alaska, May 6-12, 2007, pp. 866-874

[2] H. Hamed, A. El-Atawy, and E. Al-Shaer, "Adaptive Statistical Optimization Techniques for Firewall Packet Filtering," Proceedings of the 25th IEEE INFOCOM'06, Barceloan, Spain, April 23-29, 2006.

[3] T. Samak, A. El-Atawy, and E. Al-Shaer, "FireCracker: A Framework for Inferring Firewall Policy using Smart Probing," Proceedings of the 15th IEEE International Conference on Network Protocols (ICNP'07), Beijing, China, October 2007.

[4] S. Cosby and D. Wallach, "Denial of Service via Algorithm Complexity Attacks," Proceedings of the 12thUsenix Security Symposium, August 4-8, 2003, Washington, DC.

[5] B. Hickman, D. Newman, S. Tadjudin, and T. Martin, "Benchmarking Methodology for Firewall Performance," RFC3511, April 2003.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC2616, June 1999.

[7] C. Chi, L. Liu, L., and L. Zhang, "Quantitative Analysis on the Cacheability Factors of Web Objects," Proceedings of the 30th International Computer Software and Applications Conference (COMPSAC), September 2006, Chicago IL, September 2006, pp. 532-538.

[8] CERT/CC, "Overview Incident and Vulnerability Trends", May 15, 2003.

[9] J. Wack, K. Cutler, and J. Pole, "Guidelines on Firewalls and Firewall Policy", NIST Special Publication 800-41, January 2002.

[10] A. J. Melara, "Performance Analysis of the Linux Firewall in a Host," Master Thesis, California Polytechnic State University, June 2002.