RESEARCH ARTICLE

A potential low-rate DoS attack against network firewalls

K. Salah1*, K. Sattar1, M. Sqalli1 and Ehab Al-Shaer2

1 Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
2 School of Computer Science, Telecommunication, and Information Systems, DePaul University, Chicago, Illinois 60604, USA

ABSTRACT

In this paper we identify a potential Denial of Service (DoS) attack that targets 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, thereby overwhelming the firewall and bringing it to its knees. In this paper, we present a probing technique to remotely discover the last-matching rules of a firewall. We study experimentally the effectiveness of this probing technique taking into account important factors such as the firewall’s motherboard architecture and load conditions at network links and hosts. In addition we examine the impact of launching a low-rate DoS attack on a firewall’s performance. The performance is studied in terms of the firewall’s CPU utilization and throughput, packet loss, and latency. Copyright © 2009 John Wiley & Sons, Ltd.

KEYWORDS

network security; DoS attacks; firewalls; complexity-algorithm attacks

*Correspondence

Prof. K. Salah, ICS Department, KFUPM, PO Box 5066, Dhahran 31261, Saudi Arabia.
E-mail: salah@kfupm.edu.sa

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, thereby jeopardizing its effectiveness to filter traffic in and out. 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.

A firewall policy consists in general 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 to drop the packet 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. Some techniques are hardware driven and others use sophisticated data structures of a tree-like ruleset with chains and positioning the most frequently accessed rules at the beginning of the ruleset [1-6].

Regardless of whether or not optimization is used for packet filtering, there will always be default rules and last-matching rules which consume considerable processing times compared to others. The firewall of a large-sized enterprise network can typically have a ruleset or ACL (access control list) comprising 10s of 1000s of rules or more. In commercially deployed firewalls, a firewall ruleset containing 10 K rules is not uncommon, and there exist firewalls with rulesets containing around 50 K rules [4,7-9]. 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. An outside attacker’s discovery of how to trigger these rules 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. Therefore, with a low-rate flow of these packets, the cost of processing and filtering becomes so high
that could potentially result in overwhelming the firewall and bringing it to its knees.

These DoS attacks targeting last-matching rules of firewalls can be classified as a complexity-algorithm attack [10] 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 evade 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 evaded when multiple flows are launched with much lower rates from an army of zombie machines as in today’s botnet DDoS attacks.

In previous work [11], we presented a preliminary and brief description and limited experimental evaluation results of our proposed probing technique to remotely discover last-matching rules of a network firewall. This paper is a major extension and differs from [11] in significant ways. First, this paper discusses the probing technique in much more-depth and details. Second, this paper addresses and studies the effectiveness of the probing technique when having a major change in the probing and target environment. Specifically, the paper examines the impact on PPT stretch when considering a number of key changes which include: (1) running the probing program from both Linux and Windows operating systems, (2) probing from a distant location with an attacker machine of a limited-bandwidth uplink, and (3) having a multiprocessor firewall.

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, requirements, 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 conditions are considered. Section 4 presents related work found in the literature with regard to remote discovery of last-matching rules of a firewall. Section 5 concludes the study and briefly identifies possible countermeasures.

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, namely Internet, demilitarized zone (DMZ), and private or enterprise LAN [12]. 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 of 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, thus reflecting the discovery of the default or one of the last matching rules.

2.1. The algorithm

In order to measure the stretch of the train, we will send a non-cacheable HTTP request immediately after the PPT, as illustrated in Figure 2. The HTTP request will be destined to the web server in the DMZ. Both the HTTP request and the 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. 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 [13,14]. In our experimental work, we used an HTTP request with methods of OPTIONS which queries all possible methods supported by the 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).

For example, probing selectively can employ a number of techniques in order to narrow down the search and sampling space. First, practical source and destination IP addresses can be used as those of interest and related to the targeted enterprise network. With publically available tools like whois, traceroute, lookup, etc, the IP addresses of the enterprise routers, gateways and DMZ servers can be known. Second, it is a common practice that rules handling traffic for the DMZ servers appear earlier in the ruleset to quickly filter such traffic in and out. Third, it is a common practice that certain types of traffic are always blocked [17]. This includes inbound and outbound traffic with private IP addresses starting with 10, 172, and 192. Also Traffic containing source addresses 127.0.0.1 and 0.0.0.0 is denied. Addresses in the IP multicast address range 224.0.0.0/4 are also denied [18]. Fourth, common protocols and source and destination ports can be utilized to narrow down the search. Well-known ports of 80 and 25 are very common, and rules related to these ports are likely placed earlier in the ruleset. Also in real networks, the protocol and port fields are typically paired. If a web server is present in the DMZ, then its protocol and port will have the values ‘TCP’ and ‘80’ respectively. Therefore, rules for respective counter pairs, such as ‘UDP’ and ‘80’, are highly likely to be present as possible default rules. Fifth, it is common that certain traffic (such as NetBIOS and IPX/SPX services) are inherently vulnerable to attacks and are commonly dropped and denied [19]. Such services

### Algorithm 1 ProbingRound (PPT size)

1. $P \leftarrow $ ProbingPacketType(protocol, srcIP, srcPort, destIP, destPort)
2. $\texttt{Open TCP connection with the web server}$
3. $\texttt{count} \leftarrow $ PPT size
4. $\texttt{Repeat}$
5. $\texttt{Send P;}$
6. $\texttt{Decrement count;}$
7. $\texttt{until} (\texttt{count} = 0)$
8. $t_1 \leftarrow $ GetTimeStamp();
9. $\texttt{Send W;}$  // send HTTP request
10. $\texttt{Wait();}$  // wait only for HTTP reply and ignore other possible replies
11. $t_2 \leftarrow $ GetTimeStamp();
12. $\texttt{delay} \leftarrow t_2 - t_1$
13. $\texttt{return delay}$
are infrequent and therefore are positioned typically at the bottom of the ruleset.

2.2. Properties and requirements

Several important properties and requirements have to be demystified of a potential automated and comprehensive probing process that an attacker can undertake in order to effectively discover the default and last matching rules of a network firewall. These properties and requirements may include:

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 the 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 ($PPT = 0$). The baseline of the time should be the minimum time of a set of rounds. After this the attacker can start probing rounds (with $PPT > 0$) 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’s 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’s 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. 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 evade an IDS or IPS system by probing in a stealthy fashion whereby he can disperse the probing rounds over a long period of time with a small-size PPT consisting of only one or two probing packets. 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.

5. The processing power of the attacker’s machine and available uplink bandwidth (from the attacker’s machine to the firewall) play a key role in the success of the probing technique. If the probing packets are not sent at a fast rate, the interarrival times will be large. For the success of the probing technique, the interarrival time, seen at the firewall between two probing packets in the PPT, should not be larger than the firewall processing time of the probing packet. If it is larger, no buffering will occur at the firewall, thus resulting in no noticeable PPT stretch. We formalize this further in the next section, and we also prove this requirement 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

$$D_{\text{PPT}} = \sum_{p=1}^{P} D_p.$$
A potential DoS attack against firewalls  

K. Salah et al.

3. A PROOF-OF-CONCEPT EXPERIMENT

As a proof of concept, and to validate and study further our probing technique, we set up an experiment comprising three Linux machines: attacker, firewall, and web server (as shown in Figure 2). The firewall had two Ethernet network interface cards with one card connected to the attacker’s machine and the second card connected to the web server. The connection was made over 1 Gbps crossover Ethernet cables. The three machines were all Intel Pentium 4 processors running at 3.2 GHz with 512 MB of RAM. The network cards were 3COM Broadcom NetXtreme Gigabit Ethernet with BCM5701 controller. All machines were 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. All of these dummy rules were with UDP protocol type. We used a condition for source MAC addresses since they were experimentally found to be computationally more expensive. This is because the MAC address is one of the last conditions to be checked by Netfilter within a rule [20]. 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 the Linux code with timestamps. For timestamps, we used the 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, 5000, and 10 000. Figure 3 illustrates the results obtained for the average PPT stretch observed by the probing program when running five probing rounds. Several important observations 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 Figure 3(a). Probing delays with PPT size of 10 and 4 are more noticeable than those with PPT size of 1. Third, the amplification of the processing delay has an almost linear relationship with the PPT size. Figure 3(a) shows that the measured stretch with PPT size of 4 is four times greater than the stretch with PPT size of 1. Similarly, the measured stretch with PPT size of 10 is 10 times greater 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 Figure 3(b)) depends primarily on PPT size. For example, the fluctuation is less with PPT size of 10 than with 4. In summary, we can conclude that a relatively large PPT size would minimize fluctuation in PPT stretch and would also amplify the processing delays encountered at the firewall, thus simplifying the task of the attacker in detecting the position of these rules.
3.1. Impact of changing the probing environment and having a multiprocessor firewall

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 a 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 results similar to those with Linux, but with more numerous 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 56 Kbps, will not be successful. This is because the interarrival time of the probing packets will be larger than the processing time (of a maximum of 1 ms) 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 an impact similar to 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. Unlike other cases, the probing with the 100 Mbps uplink Linux machine was performed from the university student dormitory located at the edge of our university campus network which has a diameter of 6 km. Lastly, Figure 4(c) shows the impact of probing a firewall with a dual-processor architecture. We used a PPT size of 4. Figure 4(c) shows that using dual-processor architecture resulted in a minor processing speed-up by the firewall, and relatively small reduction in the PPT stretch. The reason is that Linux Netfilter processing is single threaded and does not distribute processing evenly between the two available processors.

3.2. Impact of load variability of links and hosts

We next consider the impact on PPT stretch under fluctuation and variability of background processing load of network hosts, as well as load conditions of network traffic at links. To experimentally study this impact, we changed the testbed to include an additional Linux machine (with the same spec as the Attacker machine) to generate relatively highly fluctuating traffic. This traffic gets aggregated with the attackers’ traffic via a 3COM 2816 Layer 2 switch. To generate the highly fluctuating traffic, we used the open-source D-ITG 2.4.4 generator [21]. D-ITG is a user-level multithreaded generator and is composed of two agents: ITGSend and ITGRecv. ITGRecv can receive flows from multiple ITGSend agents running on different machines. An ITGSend agent has the ability to generate different types of network traffic with concurrent single or multiple flows. For each flow, ITGSend allows variation of the flow’s interarrival times of generated packets as well as variation of the flow’s packet sizes. The variation can be specified according to a random distribution. Distributions include constant, uniform, Poisson, and Pareto.

In order to introduce load variability at links and hosts of the testbed shown in Figure 5, we ran a single ITGRecv agent on the Web server and two ITGSend agents. One ITGSend ran at the traffic generator machine, and the second ran at the attacker machine. The ITGSend of the traffic generator machine and attacker machine generated multiple flows with different variations in packet sizes and interarrival times. Figure 6(a) shows the variability of the accumulated traffic observed at the firewall ingress link. The figure was obtained by using `ifconfig` and `watch` Linux utilities which recorded the average rate per second for a

![Figure 4](image1.png)

**Figure 4.** Impact of (a) probing from Windows machine, (b) probing from distance location with a limited uplink bandwidth, and (c) having a multiprocessor firewall.

![Figure 5](image2.png)

**Figure 5.** Addition of a traffic load generator to testbed.
A potential DoS attack against firewalls

K. Salah et al.

Figure 6. Fluctuation of (a) Link Utilization and (b) CPU utilizations.

duration of 30 s. The corresponding CPU utilization of the attacker machine, firewall, and Web server are shown in Figure 6(b). We used `sar` and `watch` Linux utilities to obtain the CPU utilizations.

The corresponding impact on PPT Stretch latency when running the probing algorithm (with PPT size of 4) under load and traffic variability exhibited in Figure 6 is shown in Figure 7. Figure 7(a) shows that the measured PPT stretch latency at the targeted rule has a large degree of fluctuation or deviation from the average point. The range is exhibited by the red vertical bars. This is expected due to the fluctuation of the load at links and hosts. Figure 7(b) plots the minimum values of the PPT stretch latency. These values are consistent and similar to the measurements shown in Figure 3(b) which were taken under no variability of network hosts and links. Therefore, to compensate for load variability at network hosts and links, it can be concluded that the probing technique is definitely more effective in discovering last-matching rules when the minimum value of the PPT stretch latency is considered for multiple probing rounds. It is worth mentioning that we found that 5–6 probing rounds with PPT size of 4 would give a value close to the minimum value, under such high fluctuation and variability conditions.

3.3. Impact of DoS attack rate on performance degradation

We next examine the impact and severity of traditional and smart DoS attack rates on the performance degradation of the firewall. The performance is studied in terms of the firewall’s CPU utilization and throughput, packet loss, and latency. We define a smart DoS attack as an attack that targets the last-matching rules. On the other hand, a traditional DoS attack has no intelligence about last-matching rules, and targets rules typically positioned at the top of the ruleset. To get an insight into the impact of the attack rate on the performance, we use the same testbed as shown in

Figure 7. (a) Impact of load variability on PPT Stretch and (b) corresponding minimum PPT Stretch.
Figure 5. We use D-ITG to generate a normal constant 10 Kpps flow from the traffic generator to the Web server, and we launch a DoS attack from the attacker machine. For the DOS attack, we used different kernel and user-level applications to generate crafted packets to target firewall rules at position 1000, 5000, and 10000. The results shown in Figure 8 were obtained using KUTE [22] as the DoS attack program. KUTE is an open-source kernel-level UDP traffic generator. Similar results were obtained when using other user-level traffic generators such as TrafficGen [23] and mgen [24]. At the attacker machine, running these generators from Windows or Linux platforms made little difference to results. As opposed to D-ITG, these programs generate traffic without requiring a reception agent as that of ITGRecv.

At the firewall we configure the ruleset to have rules at the top positions to pass a normal constant 10 Kpps flow. We examine the performance degradation and severity shown by this flow in terms of packet loss, throughput, and roundtrip delay on this normal traffic flow, by varying the rate of DoS attacks that target firewall rules at position 1000, 5000, and 10000. DoS attack flow that targets rules positioned at 5000 and 10000 can represent smart DoS attacks that target last-matching rules, whereas DoS attack flow that targets rules positioned at 1000 can represent traditional DoS attacks. The CPU utilization at the firewall is shown in Figure 8(a). The CPU utilization shows how rapid the firewall gets saturated, i.e., by reaching its maximum capacity, and therefore resulting in dropping any additional traffic load exceeding such capacity. From the figure it is clear that the CPU utilization rapidly shoots to 100% when subjecting the firewall to a very low rate of smart DoS attacks which target rules positioned at 5000 and 10000. However for traditional attacks targeting rules positioned at 1000, the CPU utilization gradually increases as the DoS attack rate increases. Figure 8(b) shows the impact of the different types of DoS attack rates on throughput of the 10 Kpps D-ITG normal flow that passes the firewall to Web Server. Also the corresponding packet loss and round trip delay of this flow are shown in Figure 8(c) and 8(d) respectively. It is clear from these figures that smart attacks with low rate have severe and significant performance degradation,
specifically when the rate is greater than 5 Kpps. On the other hand, traditional DoS attacks show small performance degradation only at a high rate, specifically when the rate is greater than 15 Kpps. For our measurements, the CPU utilization was measured by the sar Linux utility. However, the throughput, packet loss, and round trip time were measured by D-ITG.

4. RELATED WORK

The literature comprises little work on remotely discovering the security policy of a firewall or on attacking network firewalls using low-bandwidth DoS attacks. However, there exist firewall analysis tools to optimize and detect misconfiguration in the firewall security policy as reported in Reference [1,2]. Other firewall analysis tools focus on firewall’s performance in terms of implementation and filtering delays [11,25]. Some work has been done where the analysis tested firewalls for the vulnerability to traffic-specific attacks, such as IP spoofing attacks [26]. In Reference [27] performance metrics for vulnerabilities resulting from firewall operations were presented and analyzed. In Reference [28] a traceroute technique was used to determine whether or not a particular packet can pass from an outside remote host to a destination host behind a firewall.

The closest related work with regard to remotely discovering the security policy of a firewall was reported by Samak et al. [3]. In Reference [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 pinger be pre-compromised, and that both pinger and ponger be in 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 Reference [3] the security policy and default rules can then be deduced from the packets received and not sent, and also by examining the delays of received packets.

There is a number of shortcomings of the probing technique reported in Reference [3] when compared to ours, 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 directly 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 the NTP clock synchronization protocol, with regular updates. Our past experimental work with NTP synchronization shows that such an implementation is impractical for fine microsecond granularity of delays. Third, Samak et al.’s technique requires that the pinger send all crafted probing packets to the ponger, so that 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 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 the ‘accept’ space destined to other hosts within the protected network. Fourth, Samak et al.’s work did not address the issue of delay fluctuation in the network. In our PPT technique a train of packets makes it possible to compensate for small variations in one single probe trial and also to magnify small processing delays at the firewall.

5. CONCLUDING REMARKS

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 subsequently target these rules to launch an effective and low-rate DoS attack to trigger worst-case or near worst-case processing, thereby overwhelming the firewall and bringing it to its knees. We validated the probing and detection technique experimentally. In our experiment, we studied important probing parameters and considered different environment factors which included the probing platform, location, and uplink bandwidth. We demonstrated that there was little effect on the probing technique when considering firewalls with uniprocessor and multiprocessor architecture. In addition, we examined experimentally the impact of load variability of links and hosts on probing technique. We have shown that the probing technique is more effective in discovering last-matching rules when the minimum value of the PPT stretch latency is considered for multiple probing rounds. Lastly, we examined the impact and severity of traditional and smart DoS attack rates on the performance degradation of the firewall. The performance is studied in terms of firewall’s CPU utilization and throughput, packet loss, and latency. It was clear that smart DoS attacks with low rate may cause severe and significant performance degradation.

The probing technique presented in this paper can potentially be applied to other network security appliances. These appliances may include intrusion detection or prevention appliances such as Snort, whereby an attacker can probe for rules that trigger worst-case processing before possibly launching an effective low-rate DoS attack. Therefore, strong countermeasures to evade and subvert such an attack or probing mechanism should be devised. Possible countermeasures against the probing technique may include inline anomaly detection using statistical or AI approaches of accessing last-matching rules. A profile of abnormal and sudden access to last-matching rules (such as that of back-to-back probing packets when using PPT of large size) may indicate an anomaly. Once an anomaly is detected, a new rule would be defined and placed on the top of the
ruleset (of a firewall or IPS appliance) to deny all traffic from the probing source. Another countermeasure is to re-structure and re-design the software of firewalls and IPS appliances for caching the most frequently accessed rules. In this way, under DoS attack, the last-matching rules will be logically on the top of the ruleset and be matched first. A third countermeasure is to design firewalls and IPS appliances to be more resilient and fault-tolerant against DoS attacks by employing a multi-threaded filtering engine that takes advantage of today’s multi-core architecture. To date, both Linux Netfilter engine and Snort are single-threaded. Also, as was demonstrated in this paper, Netfilter does not take advantage of multi-processor architecture to distribute the processing load evenly between available processors. With uniprocessor machines, Linux Netfilter can be designed with multiple threads such that each thread is responsible for processing only a subset of the ruleset. High priority threads can be designed to process top rules, and low-priority threads can process last-matching rules. With multiprocessor architecture utilizing hard processor affinity, performance can be further improved by binding each thread execution permanently to a separate processor. In this way, and under DoS attacks targeting last-matching rules, normal traffic will be given a higher chance to be processed.

ACKNOWLEDGEMENTS

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 thank Taghrid Samak and Adel El-Atawy for their valuable replies and comments on some of the issues faced throughout this work. We also thank the anonymous reviewers for their valuable comments in improving considerably the quality of this paper.

REFERENCES


