Firewall Fingerprinting

Amir R. Khakpour* Joshua W. Hulst* Zihui Ge† Alex X. Liu* Dan Pei† Jia Wang†
*Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA
†AT&T Labs – Research, Florham Park, NJ, USA
{khakpour, hulstjos}@cse.msu.edu, gezihui@research.att.com,
alexliu@cse.msu.edu, {peidan, jiawang}@research.att.com

***Abstract*—Firewalls are critical security devices handling all traffic in and out of a network. Firewalls, like other software and hardware network devices, have vulnerabilities, which can be exploited by motivated attackers. However, because firewalls are usually placed in the network such that they are transparent to the end users, it is very hard to identify them and use their corresponding vulnerabilities to attack them. In this paper, we study firewall fingerprinting, in which one can use firewall decisions on TCP packets with unusual flags and machine learning techniques for inferring firewall implementation.**

I. Introduction

A. Motivation

The security and reliability of firewalls are critical because they serve as the first line of defense in examining all traffic in and out of a network and they have been widely deployed for protecting both enterprize and backbone networks. However, just like any other networking and computing devices, firewalls often have vulnerabilities that can be exploited by attackers [1], [2]. To exploit firewall vulnerabilities, the first step that attackers need to do is firewall fingerprinting, *i.e.,* identifying the particular implementation of a firewall including brand name, software/firmware version number, etc. On the defense side, we need to know how attackers possibly can fingerprint a firewall so that we can design countermeasures accordingly. In this paper, for the first time, we investigate firewall fingerprinting methods with quite high accuracy. Designing countermeasures for firewall fingerprinting is out of the scope of this paper, but is the next step of this line of research.

B. Limitation of Prior Art

Prior art mostly focused on operating system fingerprinting [3]–[8]. Many tools such as NMAP [3] has been developed to identify a target host's operating system using TCP and UDP response characteristics. There are several approaches to finding out the operating system ranging from simple banner observation to highly complicated TCP, UDP and ICMP-header analysis. However, none of these methods can be used for firewall fingerprinting because firewalls, like other network middleboxes, forwards the traffic and cannot be targeted directly. For security purposes, some firewalls are configured in bridge mode with no IP address to be remotely accessible by the administrator. Hence, such approaches cannot be effective for firewall fingerprinting.

C. Technical Challenges

Firewall fingerprinting has two major technical challenges. First, finding the firewall implementation characteristics that we can use for fingerprinting is difficult because firewalls are mostly closed source and it is difficult to infer any implementation details from them. Moreover, there are many parameters and configuration details that can affect the performance of a firewall. Second, inferring the type of a target firewall is hard for attackers as they have no remote access to the firewall.

D. Our Approach

In this paper, for the first time, we propose a set techniques that can collect some information about each firewall using packet processing time of probe packets and be used to identify firewall implementation. For our study, we build a testbed consisting of three popular firewalls (including an open-source firewall and two enterprize firewalls [9]), two computers hosting eight virtual machines for generating traffic to the firewalls, and one computer for sending probe packets and measuring the firewall processing time of the probe packets. Because firewalls are very expensive and our budget was limited, we could only support three firewalls in our testbed. Moreover, due to privacy and legal reasons, we are obligated to keep the brand and model of firewalls confidential.

We address the two technical challenges as follows. To address the first challenge, we measure packet processing time to identify the type of packet classification algorithms in use by the three firewalls, sensitivity of firewall performance to traffic load, and other characteristics. We use four different techniques to send probe packets to identify the type of caching mechanism that a firewall uses in both stateful and stateless modes. We also examine the sensitivity of firewall performance to transport protocol types (*i.e.,* TCP or UDP) and packet payload size. To address the second challenge, we first study firewall decisions for TCP packets with unusual flags to find fingerprints for different firewalls. We also use machine learning techniques to identify firewalls using features extracted from probe packet processing time.

E. Key Contributions

We made two key contributions in this paper. First, we identified firewall implementation characteristics that one can evaluate for black box firewalls. Second, we proposed methods for inferring the implementation of a target remote firewall.

II. Related Work

To best of our knowledge, there has been no work on firewall fingerprinting. Yet, there have been many tools developed for operating system fingerprinting tools including NMAP [3], xprobe2++ [4], and p0f [5]. NMAP and other similar tools have a database of heuristics for identifying different operating systems based on the operating system response to TCP and UDP probe packets. Medeiros *et al.* [6] uses TCP SYN packets to collect TCP Initial Sequence Number (ISN) samples and classifies operating systems accordingly. Snacktime [7] exploits TCP window, IP time-to-live, and the length and number of retransmissions of the SYN-ACK during TCP handshakes to identify the operating systems.

978-1-4673-0775-8/12/$31.00 ©2012 IEEE

Interestingly, Smart *et al.* [8] proposed fingerprint scrubber as a module of a firewall to defeat the remote OS fingerprinting.

Work has also been done on firewall performance evaluation [10], [11]. Lyu and Lau measured the performance of a firewall under seven different policies, where each policy is for one security level [10]. In a similar vein, Funke *et al.* evaluated the firewall performance (mostly firewall throughput) under policies with differing number of rules [11]. They also show that more rules do not necessarily imply poorer firewall performance.

III. Background

A. Firewall Policies

For each incoming or outgoing packet, a firewall decides to accept or discard it based on its policy. A firewall policy is composed of a sequence of rules, where each rule specifies a predicate over fivefields: source and destination ports, source and destination IP addresses, and IP protocol. Typically, firewall policies do not check the source port field. The rules in a firewall policy may overlap and even conflict. To resolve conflicts, firewalls follow the first-match semantic, *i.e.,* the decision of the first rule that a packet matches is the decision for the packet. An example firewall policy is in Table I.

Rule	Src IP	Dest IP	Src Port	Dest Port	Protocol	Action
r_1	1.2.3.0/24	*	*	*	TCP	discard
r_2	*	1.2.3.0/28	*	80	*	accept
r_3	*	*	*	*	*	discard

TABLE I
AN EXAMPLE FIREWALL POLICY

B. Caching and Statefulness

One method of increasing firewall performance is to cache rules or flows based on temporal locality. Rule caching stores the four-tuple of source IP, destination IP, destination port, and protocol type for packets that a firewall has performed a full lookup on its policy for. The decision associated with each entry is stored in the cache. When a firewall with rule caching receives a packet, it first checks whether the four-tuple header of the packet is in its cache; if found, the decision for the packet can be made without checking the packet against the main firewall rules; if not found, the firewall checks the packet against its policy and then caches the four-tuple of the packet with the decision. Flow caching stores the five-tuple, which includes the source port field in addition to the four fields used in rule caching. The lookup process for flow caching is similar to that for rule caching. The purpose of flow caching is to have a fine-grained access control beyond firewall policies. For example, to protect against SYN flooding attacks, some firewall products stop accepting new SYN packets with new source ports when they see too many open flows from a specific source address with different source ports.

Commercial firewalls often support both stateful or stateless modes. A stateful firewall tracks TCP sessions in a state table by examining the TCP flags of incoming TCP packets. This ensures that the packet in a TCP session follows the correct order that includes a proper handshake and tear-down. The firewall drops any packet with an illegitimate flag. After a correct handshake, an entry is made in the state table. The packets that match the session entries bypass the firewall. Once a session goes through the correct termination procedure, its table entry is removed.

C. Packet Classification Solutions

The process of checking a packet against a firewall policy is called packet classification. Packet classification solutions fall into two main categories: software based solutions and Ternary Content Addressable Memory (TCAM) based solutions. Software based packet classification solutions include the simple sequential search algorithm and other algorithms based on complex data structures (*e.g.,*, [12]–[16]). The sequential search algorithm compares a packet with each rule in a firewall policy sequentially until a match is found. Complex data-structure-based packet classification algorithms include Recursive Flow Classification (RFC) [14], Aggregated Bit-Vector [15], Tuple space [16], HiCut [12], and HyperCut [13], etc. For TCAM based packet classification, firewall rules are stored in a special memory chip; for any given packet, the hardware circuit of the chip compares the packet with every stored rule in parallel and returns the decision of the first rule that matched the packet. TCAM based packet classification is widely used in high performance routers and firewalls because the lookup is done in constant time.

IV. Overview

A. Roadmap

To study firewall fingerprinting, we design a testbed with three popular firewalls for conducting extensive experiments and performance measurements. Of the three firewalls tested, two are software firewalls while the other is hardware based. A software firewall is implemented fully in software and may reside on a multipurpose machine as one of many services being provided. Typically, software firewalls are highly configurable and offer more customization and services than their hardware counterparts. Hardware firewalls are made specifically tailored for packet classification. Generally, they are more limited in capabilities than software firewalls but are usually very fast in classification as they are purpose built.

Our measurements are mostly based on probe packet processing time taken on remote hosts before and after a firewall. In our initial experiment we study firewall characteristics induced by their implementation. We examine firewall packet classification algorithms to understand whether or not they use sequential search for packet filtering. We then measure the sensitivity of firewall packet classification algorithms to firewall background traffic load. We continue our studies by inspecting the firewall caching techniques and specifying their caching effectiveness. We finalize our study by looking at firewall processing time with respect to probe packet payload size to understand if they have an impact on the firewall packet processing time (PPT).

The second experiment is to determine if an attacker can infer the implementation of a firewall remotely by sending probe packets through the firewall. The firewall implementation inference process is studied from two perspectives. First, we try to find a signature for each firewall based on the decision of a sequence of TCP packets with an unusual set of flags. The results show that the three firewalls, especially if they are in stateful mode, discard TCP packet sequences with unusual TCP flags. As administrators rarely define policies on TCP flags, the obtained signature usually has a close association with the firewall implementation. An attacker can use this signature to infer firewall implementation remotely

with high confidence. Second, as a complementary method, we use PPT of a sequence of probe packets to train a classification model for each firewall and use it accordingly to infer the implementation of a target firewall. Note that in the attacking scenario, the attacker needs to build simple testbeds including all speculated firewall brands to acquire signatures and the classification model. He then needs to (1) install a bot (or a trojan) on a machine or compromise a host inside the network, (2) use security scanner tools such as `nmap` [3] to find the packets that can go through the target firewall and reach the compromised host and (3) generate and send probe packets to measure their PPT. An attacker can also obtain more information using other monitoring tools (*e.g.,* `traceroute`) to understand the number of hops and the extra delay between the probe packet sender and receiver to create more accurate models for firewall implementation inference.

B. Measurement Environment

Figure 1 shows the testbed topology for our testing of three different firewalls. Firewalls FW1 and FW2 are software firewalls running on a Linux machine with SMP kernel 2.6. Each firewall has 2 quad-core Intel Xeon 2.66GHz CPUs and 16GB of RAM. FW3 is a hardware firewall that runs on a routing engine board with a 850MHz processor, 1,536MB DRAM, and 256MB compact flash. Each firewall is configured with the same policy comprised of 375 rules. The first 374 rules are set to accept traffic with the final rule discarding all traffic that is not specified previously. The firewall policy is chosen from real-life firewall policies used in a university campus network. The rules are defined over four packet header fields: source IP, destination IP, destination port number, and protocol. As with most real-life firewall policies, only a few rules overlap. Moreover, there is no rule hidden by another rule (*i.e.,* there is no rule with lower index that completely covers a rule with higher index). Furthermore, the firewalls are only configured for packet filtering; other services such as VPN or NAT are disabled. Note that the rules are known for the first experiment where we study firewall characteristics induced by their implementation. However, for the second experiment, in which we propose algorithms to fingerprint firewalls, an attacker does not know the rules.

In addition to the firewalls, the testbed has two machines, VM1 and VM2, running VMWare ESX 3.5.0 on a similar machine with 2 quad-core Intel Xeon 2.66GHz CPUs and 16GB of RAM Each VMWare instance has four Linux virtual machines connected to each other by virtual switches. These virtual switches are connected directly (without an intermediary switch) to each firewall (FW1, FW2, and FW3). The virtual machines on VM1 and VM2 are used to place background traffic load on the firewalls by sending a substantial amount of packets to different interfaces of the firewall. The traffic is generated by Mausezahn network traffic generators (aka `mz`) [17], an open-source traffic generator. Using both VM1 and VM2, we are able to sustain a traffic rate of up to 300Mbps. Based on the design of experiments and attacks, the generated traffic can be accepted or discarded by the firewall to which it is sent. To put maximum load on the firewalls, the generated traffic has no packet payload. This maximizes the number of packets that a firewall needs to process. If packets have payloads, firewall throughput will increase, but traffic packet rate (*i.e.,* packets per second) will decrease. As mentioned, the virtual switches are directly connected to the firewalls. This is to separate the generated traffic for each firewall and make firewall experiments independent from each other.

The last portion of the testbed is the Probe Machine & Traffic Analyzer (PMTA): a Linux machine with Dual Quad-core Intel Xeon 2.66GHz CPUs and 16GB of RAM. We send probe packets by PMTA directly (*i.e.,* no switch in between) to each firewall using an open-source packet generator `hping2` [18]. If the probe packets are accepted by the target firewall they are routed back to PMTA through another interface (as it is shown in Figure 1). In order to measure firewall packet processing time, we use packet trace time-stamps. We use tcpdump [19] to dump packets with time-stamps with microsecond resolution. For the software firewalls (FW1 and FW2), we can analyze the packet traces and calculate the PPT based on the difference of packet trace time-stamps of outgoing and incoming interfaces. However, the hardware firewall (FW3) does not support tcpdump or any traffic monitoring (*i.e.,* packet dumping) feature. Therefore, since we cannot measure the packet processing locally on the firewalls, the probe packets are forwarded to PMTA and we calculate the time-stamp difference of the packet traces on PMTA. The time-stamp differences calculated on PMTA comprise the firewall PPT plus probe packet round trip time (RTT) which in turn reduces the accuracy of firewall PPT.

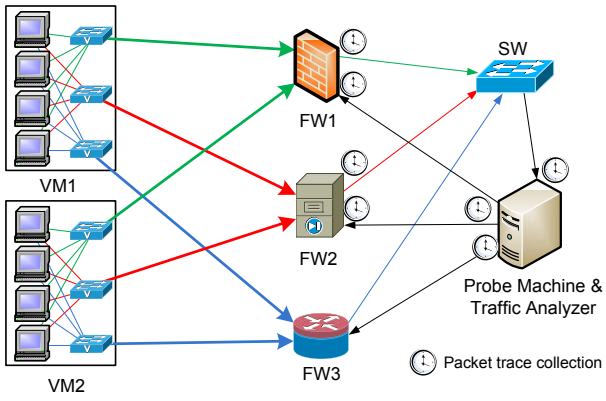

Fig. 1. The testbed

V. Firewall Characteristics

To study firewall characteristics, we first give an overview on the methodology basics such as how the probe packets are sent and how the PPT is measured by PMTA. We then show the results for different firewall features containing firewall packet classification algorithm, firewall statefulness and caching, and packets protocol and payload size impact.

A. Methodology Basics

The probe packets are sent by the PMTA in four modes as follows:

- **TCP Fix:** A sequence of TCP packets with the same packet header.
- **TCP Vary:** A sequence of TCP packets with the same packet header except the source port which is chosen randomly for each probe packet.

- **UDP Fix:** A sequence of UDP packets with the same packet header.
- **UDP Vary:** A sequence of UDP packets with the same packet header except the source port which is chosen randomly for each probe packet.

We conduct two sets of experiments with and without background traffic load in the testbed. The first set of experiments are performed under no background traffic load, *i.e.,* the probe packets are the only packets that are transmitted in the testbed during the experiments. In contrast, the second set of experiments are performed under background traffic load. In this case, all virtual machines send dummy packets with no payload to the target firewall as the background traffic. The header of dummy packets are chosen such that they are discarded by the rule configured in the firewall, *i.e.,* these dummy packets never pass through the firewall. The dummy packet rate varies from 870,000 packets to 1,875,000 packets per second. Since packets have no payload, the dummy traffic varies from 250Mbps to 300Mbps. Because the firewalls are installed on powerful machines, they are not under any type of resource constraints in terms of CPU and memory when the firewalls are under the background traffic load. This indicates that the experimental results for the firewalls under the background traffic load may not be affected by hardware resource constraints.

We use two methods for measuring PPT: (1) *Local* measurements are based on packet traces collected from the incoming and outgoing interfaces of the firewall. (2) *Remote* measurements are based on the packet traces collected from the PMTA's incoming and outgoing interfaces. The local measurements of PPT are more accurate than the remote measurements of PPT, but they require (1) local access to the firewalls and (2) the firewall interface must support packet analyzers which dump packets passing through the firewall's interfaces. In contrast to local, the remote measurement of PPT includes the packet transmission time, which reduces the accuracy. Because FW3 does not support any packet analyzers, we use local measurement for FW1 and FW2 as well as remote measurement for all three firewalls to compare between them.

B. Packet Classification Algorithm

Identifying the exact packet classification algorithm that the firewall uses is very difficult if we treat the firewall as a black box. However, we can design experiments to test (1) whether a firewall adopts a sequential search based algorithm for packet classification, (2) whether the performance of a firewall is sensitive to its traffic load; and (3) how a firewall performs in terms of the PPT.

1) Using Sequential Search: To test if a firewall uses sequential search for packet classification, we generate a sequence of probe packets where each packet matches exactly one of the rules in the firewall policy. We then measure the PPT for the probe packets. If the PPT increases linearly as we progress further down the rule list, it is likely that the firewall uses a sequential-search-based approach for packet classification. If the PPT exhibits a different change pattern or lack of change (*i.e.,* remains flat), the packet classification algorithm used by the firewall is not sequential-search-based and could be any of other algorithms described in Section III-C. We repeat this test 10 times and compute the median value of the PPT. The median value is preferred over mean value because it is less sensitive to outliers, which can be caused by the variability of network congestion and interface packet buffering, especially when the firewalls are under load. Figure 2 shows the median value of the PPT.

Figures 2 (a) and (b) show the median value of PPT measurements with and without background traffic load for FW1. Using the remote measurement method, we observe that the median PPT increases as the rule index when there is no background traffic load. A similar increasing trend is also observed on median PPT under background traffic load when the local measurement method is used. The slopes for the regression lines for PPT of FW1 using remote measurement in Figure 2(a) for with-load and no-load curves are 0.1176 and 0.1645, respectively. Similarly, the corresponding slopes for curves in Figure 2(b) are 0.1411 and -0.0317, respectively. This observation implies that FW1 is likely to use a sequential-search-based packet classification algorithm. The very small negative slope of the median PPT using local measurement under no background traffic load may indicate that FW1 uses some type of rule pre-fetching or caching, yet as the slope is very small the effect is not significant.

The results for FW2 (shown in Figures 2 (c) and (d)) suggest similar sequential-search-based classification algorithms, especially when the firewall is under load. The slopes for regression lines for PPT curves for with and without background traffic load in Figure 2(c) are 0.1339 and 0.0208, respectively. Similarly, the corresponding slopes for PPT curves in Figure 2(d) are 0.3809 and -0.0073, respectively. We can also observe that FW1 and FW2 has considerably different transmission delay especially when FW1 is under load by comparing the differences between graphs in remote measurement (*i.e.,* Figures 2(a) and (c)) with their corresponding ones in local measurement (*i.e.,* Figures 2(b) and (d)). Since the experiment environment is the same for FW1 and FW2, it seems that such difference is due to the different queuing implementation in FW1 and FW2, yet because we do not have access to both firewalls source codes, it is difficult to ensure.

We have different observations on median PPT for FW3. The slopes for regression lines for PPT curves for with and

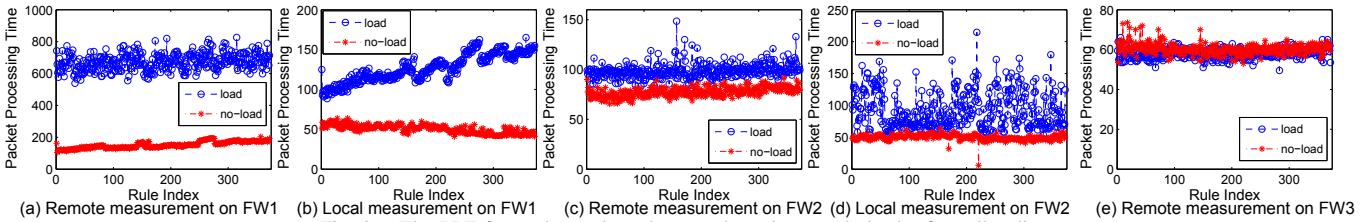

Fig. 2. The PPT for probe packets that match against a rule in the firewall policy

without background traffic load in Figure 2(e) are 0.0033 and 0.0082, respectively. The fairly flat regression lines for FW3 implies that FW3 likely uses some other techniques rather than sequential search for packet classification. As FW3 is a hardware firewall using TCAM-based packet classification methods, we believe it uses parallel exhaustive search.

2) Sensitivity to Traffic Load: Using the same experimental settings, we also evaluated the sensitivity of firewall performance to traffic load. We observe that, among all firewalls, FW1 is the most sensitive and FW3 is the least sensitive to the traffic load. Considering remote measurements shown in Figures 2 (a), (c), and (e), the median PPT of firewalls with background traffic load is 4.6034, 2.7385, and 0.9874 times larger than the median PPT of firewalls with no background traffic load for FW1, FW2, and FW3, respectively.

We observe that the PPT curves for FW1 and FW2 have sharper slopes when the firewalls are under the load. This implies that the packet classification mechanism, including packet classification algorithm and possible caching scheme, depends on the current traffic and load on the firewall.

We also find that the traffic load on the firewall has an impact on the variance and dispersion of the PPT of probe packets, which directly relate to the stability of the firewall and firewall packet reordering. Figure 3 shows the standard deviation (STD) of the PPT for probe packets. We observe in Figure 3(a) that on average the STD of the PPT for FW2 has 52.4749 times larger than that for FW1 in local measurements.

To show the relation between the STD of the PPT of probe packets in local and remote measurement, let S_i^L and S_i^R denote the vector of PPTs obtained in local and remote measurements for the i-th firewall, respectively. Let T denote the transmission delay from PMTA to the firewall and from the firewall to PMTA. Therefore, $S_i^R = S_i^L + T$. The STD of the PPT for local measurement can be calculated from the STD of the PPT for remote measurement as follows:

$$STD(S_i^R) = STD(S_i^L + T) = \sqrt{STD(S_i^L)^2 + STD(T)^2 + COV(S_i^L, T)}$$

As PPT and transmission time are independent, $COV(S_i^L, T) = 0$. Also, STD of transmission time can be represented by a constant vector c . Hence, $STD(S_i^L) = \sqrt{STD(S_i^R)^2 - c^2}$. Note that the transmission time and its standard deviation can be different based on the load on the firewalls and the way the firewalls handle queuing and packet forwarding. With that in mind, Figure 3(b) shows the STD of the firewall PPT calculated from the remote measurements. We observe that, on average the STD of PPT on FW2 is 1.7910 and 33.3 times larger than those on FW1 and FW3, respectively. In conclusion, the hardware firewall (FW3) shows less sensitivity to the traffic load and seems to be more stable in terms of performance under different network traffic loads.

3) Average PPT: In general, Figure 2 shows that FW3 yields the lowest PPT regardless of the background traffic load on the firewall. The average PPTs without background traffic load on FW1, FW2, and FW3 are 151.7891, 77.5470, and 60.3360 microseconds in the remote measurements (shown in Figures 2 (a), (c), and (e)), respectively. Similarly, the corresponding figures with traffic load on FW1, FW2, and FW3 are

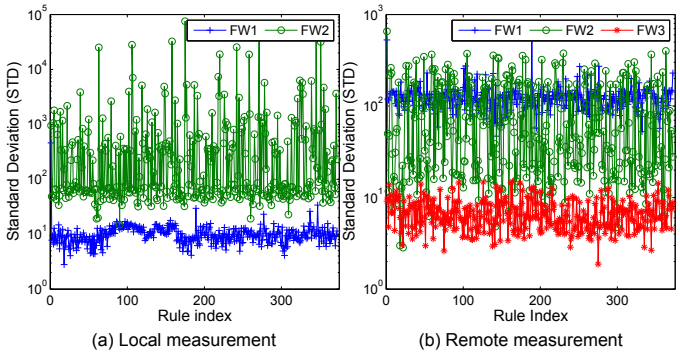

Fig. 3. The STD of the PPT for probe packets that match against a rule in the firewall policy

672.8522, 98.7970, and 57.8777 microseconds, respectively. However, in the local measurements (shown in Figures 2 (b) and (d)), FW1 and FW2 have similar average PPTs (50.3710 and 49.7796 microseconds, respectively) when there is no background traffic load on the firewalls. On the other hand, when there is background traffic load on the firewall, the average PPTs for FW1 and FW2 are in turn 126.7352 and 92.8078 microseconds. This result again indicates the high sensitivity of FW1 to the traffic load. Overall, FW3 outperforms FW1 and FW2 with the lowest average and minimum STD of PPTs, and the least sensitivity to the traffic load.

C. Caching and Statefulness

As explained, modern firewalls often use different caching mechanisms for rule and flow caching to reduce the performance overhead of packet classification. To identify if a firewall uses caching and how effective the caching is, we define *cache effectiveness* (C) as the ratio of the PPT for the first probe packet in a sequence of probe packets, whose header is not in the cache table, to the median PPT of the rest of the probe packets in the same sequence, whose headers are supposedly in the cache. If $C > 1$, the firewall uses caching effectively. If $C \simeq 1$, the firewall either does not use caching, or the caching that the firewall uses is not effective.

In our experiments, we measure the firewall's C value as follows. For each experiment we first send 10 probe packets and measure the PPT for each probe packet and calculate C value accordingly. The C value reported in this paper is the median C values for 10 repeated experiments. To determine the effectiveness of a firewall caching in stateless and stateful modes, we conduct experiments using four types of probe packet modes: TCP Fix, TCP Vary, UDP Fix, and UDP Vary. If the firewall has effective caching in TCP Fix and UDP Fix probe packets, it means that the firewall caches all 5 packet header fields in its cache table, *i.e.,* it performs flow caching. However, if a firewall has effective caching in TCP Vary and UDP vary probe packets, it means the firewall caches only 4 packet header fields (excluding source port) in its cache table. *i.e.,* it performs rule caching.

Table II presents the C values calculated based on local and remote measurements for experiments on three firewalls FW1, FW2 and FW3. In the local measurements, the results show that FW1 performs very effective flow caching on UDP packets as the cache effectivenesses for UDP Fix are significantly more than 1 ($7.4931 < C < 16.75$). However,

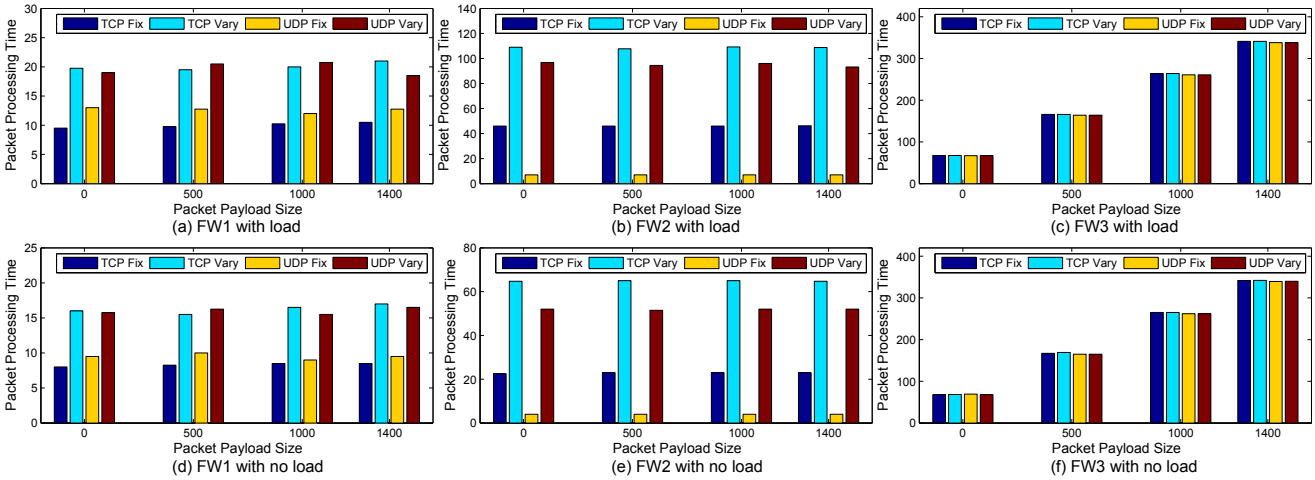

Fig. 4. The remote PPT for probe packets with different packet payload sizes

FW1 flow caching on TCP packets is not as effective since the cache effectivenesses for TCP Fix are variable around 2 ($1.9038 < C < 2.3411$). In addition, the results imply that FW1 does not support rule caching because the cache effectivenesses for UDP Vary and TCP Vary are around 1. However, there is one exception case where the cache effectiveness for TCP Vary probe packets when FW1 is under the background traffic load and configured in stateless mode is 3.2020. This could be an indication of some caching mechanisms that are enabled when FW1 is under load.

For FW2, the caching effectivenesses for UDP Fix and TCP Fix experiments range from 5.4214 to 9.8167, while the corresponding figures for UDP Vary and TCP Vary experiments range from 2.8588 to 4.6833. Because the cache effectiveness values for FW2 are much larger than 1 for all experiments, it seems that FW2 uses rule caching. In addition, because the cache effectivenesses for UDP Fix and TCP Fix experiments are larger than those for UDP Vary and TCP Vary experiments, seemingly the FW2 uses separate flow caching and rule caching mechanisms. Comparing the cache effectiveness results for FW1 and FW2, the flow caching mechanism on FW1 is more effective on UDP packets, whereas the flow caching mechanism on FW2 is more effective on TCP flows.

In the remote measurements, the transmission delay is quite larger than the actual PPT. Therefore, the cache effectiveness calculated based on remote measurement are not as expressive. However, there is one exception where FW3 has a cache effectiveness of larger than 2 in UDP Fix probe packets when FW3 is in the stateful mode. The result indicates that FW3 performs flow caching in this typical case. In addition, we find a unique feature on FW3. When FW3 is in stateful mode, once PMTA sends the first TCP SYN packet, FW3 does not accept any other TCP packet with the same packet header for a while until it receives a TCP ACK packet from the packet destination. Thus, calculating C for TCP Fix for FW3 in stateful mode is not applicable.

Another observation that we can make from the results is that in most of the cases, the cache effectivenesses when a firewall does not have background traffic load are slightly higher than those when a firewall has background traffic load. One possible explanation is that when a firewall is under load, the cache table has a large number of entries. This results in longer search time (and PPT) for the rest of the probe packets. This makes the cache less effective compared to no-load experiments, where the firewall's cache table has a small number of entries.

D. Impact of Packet Protocol and Payload Size

Firewalls usually perform queuing management techniques to improve their PPT. Such techniques can be made to be aware of the protocol and payload size of packets. In order to evaluate the impact of packet protocol and payload size, we

		Local Measurement							
		Stateful		Stateless		Stateful		Stateless	
		UDP Fix	UDP Vary	UDP Fix	UDP Vary	TCP Fix	TCP Vary	TCP Fix	TCP Vary
FW1	no load	10.4000	1.0315	16.7500	1.1943	1.9038	0.8723	2.3043	0.8000
	with load	7.4931	0.9690	10.7955	0.9725	2.3411	0.9050	2.2195	3.2020
FW2	no load	8.2333	4.9444	8.6000	4.9500	9.8167	4.6833	8.2727	4.5424
	with load	5.4214	4.4446	5.9857	4.0451	8.0074	2.8588	9.0727	3.7576
		Remote Measurement							
		Stateful		Stateless		Stateful		Stateless	
		UDP Fix	UDP Vary	UDP Fix	UDP Vary	TCP Fix	TCP Vary	TCP Fix	TCP Vary
FW1	no load	1.7246	0.9455	1.6800	0.9334	1.3957	0.9692	1.4286	0.9870
	with load	0.9999	1.1088	1.2151	1.0010	1.2243	0.9883	1.1185	1.1378
FW2	no load	1.0103	0.9393	0.9825	0.9576	1.1181	0.9240	0.9560	0.8733
	with load	0.9938	0.9141	1.3036	0.8373	1.1909	0.5466	1.0971	0.9068
FW3	no load	2.3172	0.9442	0.8090	0.8148	–	0.9756	0.7975	0.8062
	with load	2.1525	0.9318	0.8692	0.8354	–	0.9841	0.7725	0.8540

TABLE II
CACHE EFFECTIVENESS BASED ON LOCAL AND REMOTE MEASUREMENTS

configure all three firewalls in the stateless mode and repeat the same set of experiments while varying the packet payload size. Figure 4 shows the median PPT results for packet payload size of 0, 500, 1000, 1400 bytes.

We have three main observations from the results. First, Figures 4(a), (b), (d) and (e) show that software firewalls FW1 and FW2 have different PPT in TCP Fix, TCP Vary, UDP Fix, and UDP Vary experiments. We observe that the PPTs are smaller in TCP Fix and UDP Fix probe packets than those in TCP Vary and UDP Vary probe packets. This can be a result of effective flow caching on FW1 and FW2. More specifically, this observation on FW2 seems to indicate that flow caching is more effective than rule caching on FW2. Note that the above observations are made regardless of the packet payload size. However, Figures 4(c) and (f) shows that FW3 has the same PPT for all of the TCP Fix, TCP Vary, UDP Fix, and UDP Vary packets. This indicates that if FW3 has the same rule caching mechanism for all TCP and UDP packets (when it is in stateless mode). Second, the results in Figures 4(a), (b), (d) and (e) indicate that the packet payload size does not impact the PPT on FW1 and FW2. However, Figures 4 (c) and (f) show that the PPT increases linearly with regression slope of 0.1945 ± 0.0014 as the packet payload size increases. Finally, we observe from Figure 4 that the impact of packet protocol and payload size on PPT is independent from whether the firewall has background traffic load.

VI. Firewall Inference

The first step toward defeating an opponent is to know them – the same principle applies when it comes to attacking a firewall. If attackers can successfully infer the type (*e.g.,*, vendor/version) and the characteristics (*e.g.,*, statefulness) of the target firewalls, they can potentially render a much more effective attack. In this section, we examine the feasibility of firewall implementation inference using probe packets.

Our approach is motivated by the wide range of so-called operating system (OS) fingerprinting [20]–[22] techniques. The idea is that different OSes respond to non-conforming protocol (such as TCP, HTTP) interactions differently. By tracking the error-handling responses, one may uniquely identify the OS of the target host. In our case, we study the decision of firewalls for sequences of TCP probe packets with varying TCP header flags – the decision of the firewall is limited to a binary sequence of whether the corresponding packet is accepted or discarded by the firewall. To distinguish the firewall's accept/discard decision due to its configured policy, we force all probe packets in the same sequence to share the same source IP, destination IP and destination port. This ensures that these probe packets hit the same firewall rule in the typical firewall settings. However, in some uncommon cases, certain types of firewalls support policies that are based on TCP flags in addition to the other common TCP header information. This makes our binary sequence decision unreliable. Hence, we further supplement the decision binary sequence with the PPT of the probe packets and use them to infer the target firewall implementation. Note that in the remainder of the paper, all PPTs are measured remotely.

To extract the firewall behavior fingerprints and construct classifiers, we first establish a controlled environment, which includes various candidate firewalls of interest, devices outside the firewall that can be used to launch probe packets and devices behind the firewall that can be used to receive the probe packets and measure the delays. The testbed network shown in Figure 1 is an example of such a set up. The signatures and classifiers identified herein can then be applied in the "battlefield". We next describe in detail the methodology we applied for firewall implementation inference and present the result for the three firewalls in our testbed. While our testbed is limited to the three different firewalls available to us, we believe that our methodology is more generally applicable for fingerprinting other firewalls in the market.

A. Firewall inference using TCP Probe Packets

As there are eight different TCP flags defined in a TCP header, one can construct 2^8 different combinations in each probe packet. This can be further compounded by the permutations of different probe packet sequences. In our limited testbed case, we find that it is sufficient to use *two* consecutive probe packets to distinguish the behavior of different firewalls (and different modes – stateful and stateless). We show the results when we enable the TCP SYN flag along with one other TCP flag in each of the two packets. Table III and Table IV present the results for the stateful and the stateless firewalls, respectively. For the ease of presentation, we condense the information in the table such that the columns represent the different TCP flags enabled in the first probe packet (besides the TCP SYN flag) and the rows represent the different TCP flags enabled in the second probe packet (besides SYN flag). In each cell, there are three indicators representing the firewall decision from the firewall FW1, FW2 and FW3 respectively. Indicator "*" means that both probe packets are accepted by the firewall and successfully received at the device behind the firewall; indicator "-" means that one or both probe packets are discarded, or more strictly speaking, missed by the receiver.

	URG	FIN	RST	PSH	ACK	ECE	CWR
URG	**-	**-	-*-	**-	-*-	**-	**-
FIN	**-	-*-	**-	**-	**-	**-	**-
RST	-*-	**-	**-	**-	-*-	-*-	-*-
PSH	**-	**-	**-	**-	-*-	-*-	***
ACK	-*-	**-	-*-	-*-	-*-	-*-	-*-
ECE	**-	**-	-*-	-*-	-*-	**-	***
CWR	**-	**-	-*-	***	-*-	***	**-

TABLE III
STATEFUL FIREWALL

	URG	FIN	RST	PSH	ACK	ECE	CWR
URG	***	***	-**	***	***	***	***
FIN	***	-**	***	***	-**	***	***
RST	-**	***	***	-**	***	***	***
PSH	***	***	-**	***	***	***	***
ACK	***	-**	***	***	-**	***	***
ECE	***	***	***	***	***	***	***
CWR	***	***	***	***	***	***	***

TABLE IV
STATELESS FIREWALLS

The result in Table III and IV demonstrates that tracking the firewalls' feedback to well-crafted TCP probe packets can be effective in obtaining valuable information to distinguish different firewalls. Unlike FW1 and FW3 that filter out some probe packets, for the stateful and stateless setting of FW2, both probe packets have passed through the firewall in all 98 cases. While our example uses certain combinations of TCP flags, other combinations can prove useful for other

firewall types and settings. We next demonstrate that one can use supplement information from the PPTs for better firewall implementation inference.

B. Firewall inference using Packet Processing Time

Although firewall fingerprinting using a sequence of TCP packets is a deterministic method used to infer firewall implementation, firewall rules on TCP flags can change the default decision of the firewall and cause misidentification of the firewall implementation. Thus, we propose an alternate method to use implementation characteristics including PPT and cache effectiveness to infer firewall implementation and its statefulness. In this method, we build a classification model for each firewall and for their statefulness modes based on their median PPT, STD of the PPTs, and cache effectiveness. We then use this classification model to infer firewall implementation.

To build a classification model and analyze its accuracy, we first create a dataset containing 3,600 data points. For each data point, we send 11 consecutive probe packets in four different modes (TCP Fix, TCP Vary, UDP fix and UDP vary) with and without payload (total of 8 times). Each data point is represented by $\mathbf{x} = \langle x_1, \cdots, x_{24} \rangle$ that has 24 features where x_{3i-2} is the median of the PPTs, x_{3i-1} is the STD of the PPTs, and x_{3i} is the cache effectiveness for the 11 probe packets. The data points are collected when the firewalls are under three different load levels: no load, medium load, and full load. We also use three set of labels: the labels for the firewall type ($Y_1 = \{$'FW1','FW2','FW3'$\}$), the labels for the firewall statefulness ($Y_2 = \{$'stateful', 'stateless'$\}$), and the labels for firewall type and statefulness ($Y_3 = \{$'FW1-SF','FW2-SF','FW3-SF','FW1-SL','FW2-SL','FW3-SL'$\}$).

For the classification, we use multi-class Support Vector Machines (SVMs) with Radial Basis Function (RBF) kernel with parameters, (γ=0.01, C=500). Note that the value for RBF kernel parameters have been chosen by model selection algorithms that we used to maximize the classification accuracy. We conduct classification separately for each set of labels, under two conditions (1) if the attacker somehow knows the firewall load and (2) if the attacker does not know the firewall load when he tries to infer firewall implementation.

The classification problem under first condition is a classic one, where the data point is $\mathbf{x}$ with an additional feature of firewall load level, denoted by z. However, the classification problem under the second condition is not as straightforward. When we train the classifier we know the firewall load level, but when we use it for testing we do not know the load level. To solve this problem, we first formulize the problem as follows:

$$P(Y|\mathbf{x}) = \sum_{z} P(Y, z|\mathbf{x}) = \sum_{z} P(Y|z, \mathbf{x}) P(z|\mathbf{x}) \quad (1)$$

Formula 1 indicates that we need two probabilistic classifiers: the first one is to speculate the firewall load level (z) given test data point ($\mathbf{x}$), and the second classifier is to predict the firewall label, given the test data point and the speculated load level. Using the probabilistic classifiers, we first calculate the probability of each label for a given data point and then choose the label with the highest probability as the final label of the data point. For the probabilistic classifier, we use multi-class libsvm with probability estimates [23].

Before classification, we use feature selection to maximize classification accuracy rates. By using feature selection, we not only alleviate the curse of dimensionality, but also find the most important and distinctive features in our feature set. Thus, we use Sequential Forward Searching (SFS) algorithm [24] for feature selection for each set of labels. The results indicate that: (1) To infer the firewall implementation, we only need 6 features that contain median of PPTs for probe packets in TCP Fix and TCP Vary modes with payload, and cache effectiveness for all probe modes with payload. (2) To infer the statefulness, we need 16 features that contain all probe packets with payload features as well as cache effectiveness of probe packets with no payload features. (3) To infer the firewall implementation and statefulness, we need 7 features that contain the cache effectiveness for all probe test modes with payload, median of PPTs for probe packets in TCP Fix and UDP Vary modes with payload, and cache effectiveness for probe packets in UDP Vary mode with no payload. Note that when using different methods of classification, the important features may be different. However, our results show that the most distinctive features are the cache effectiveness for the probe packets with payload, which complies with Figure 4.

	Y_1: Firewall Type - Known Load			
	Accuracy	Misclassification		
Total	94.56%	FW1	FW2	FW3
FW1	100%	-	12.13%	0.54%
FW2	84.36%	0%	-	0.35%
FW3	99.11%	0%	3.51%	-
	Y_1: Firewall Type - Unknown Load			
	Accuracy	Misclassification		
Total	94.61%	FW1	FW2	FW3
FW1	100%	-	12.41%	0.63%
FW2	84.58%	0%	-	0.30%
FW3	99%	0%	3.01%	-

TABLE V
ACCURACY AND MISCLASSIFICATION FOR FIREWALL TYPE LABELS

The accuracy results for the classification under two conditions for each set of labels are reported in Tables V, VI, and VII. The results are the mean of 10 cross-validation accuracy and misclassification results for the dataset.

The results in Table V indicate that we can predict the firewall implementation with 94.56% and 94.61% accuracy for known and unknown load, respectively. The results also show that while the accuracy of predicting FW1 is 100%, the accuracy of predicting FW2 is 84.36% (and 84.58% for unknown load) because in 12.13% (and 12.41% for unknown load) of the time it is misclassified by FW1. The closeness of the accuracy rates for known and unknown load assumptions show that the firewall load level plays an insignificant role in classification. Yet for other classification methods or other set of firewalls, the firewall load level may be an important factor in classification. Thus, it should not be overlooked.

The results in Table VI show that we can predict the statefulness of the firewall with 85.79% and 85.72% of accuracy for known load and unknown load, respectively. Surprisingly, the accuracy rate is very good knowing that the statefulness of a firewall has a trivial impact on the PPT.

The results in Table VII show that we can predict the type of a firewall and its statefulness with 74.04% and 74.06% of accuracy for known load and unknown load, respectively,

	Y_2: Statefulness - Known Load		
	Accuracy	Misclassification	
Total	85.79%	Stateful	Stateless
Stateful	89.49%	-	17.76%
Stateless	82.24%	10.51%	-
	Y_2: Statefulness - Unknown Load		
	Accuracy	Misclassification	
Total	85.72%	Stateful	Stateless
Stateful	89.60%	-	18.01%
Stateless	81.99%	10.40%	-

TABLE VI
ACCURACY AND MISCLASSIFICATION FOR STATEFULNESS LABELS

	Y_3: Firewall Type and Statefulness - Known Load						
	Accuracy	Misclassification					
Total	74.04%	FW1-SF	FW2-SF	FW3-SF	FW1-SL	FW2-SL	FW3-SL
FW1-SF	69.39%	-	2.17%	0.92%	20.55%	5.19%	0%
FW2-SF	59.68%	0%	-	1.62%	15.02%	44.54%	1.87%
FW3-SF	97.48%	0%	0%	-	0%	0%	0%
FW1-SL	79.03%	30.61%	6.77%	1.60%	-	7.82%	0%
FW2-SL	41.30%	0%	28.16%	0%	0.42%	-	0%
FW3-SL	98.13%	0%	3.22%	0%	0 %	1.16%	-
	Y_3: Firewall Type and Statefulness - Unknown Load						
	Accuracy	Misclassification					
Total	74.06%	FW1-SF	FW2-SF	FW3-SF	FW1-SL	FW2-SL	FW3-SL
FW1-SF	69.39%	-	2.32%	0.92%	20.62%	5.27%	0%
FW2-SF	59.53%	0%	-	0%	0%	44.31%	1.96%
FW3-SF	97.39%	0%	0%	-	0%	0%	0%
FW1-SL	78.95%	30.61%	6.77%	1.69%	-	7.82%	0%
FW2-SL	41.76%	0%	28.35%	0%	0.42%	-	0%
FW3-SL	98.04%	0%	3.03%	0%	0%	0.84%	-

TABLE VII
ACCURACY AND MISCLASSIFICATION FOR FIREWALL TYPE AND STATEFULNESS LABELS

which is relatively good as we have six classes and random classification accuracy rate is 16.67%. As show in Table V, FW1 is classified with high accuracy, while FW2 is classified with relatively low accuracy. Inferring FW3, on the other hand, can be done with very good accuracy. The misclassification rates also suggest that both stateful FW1 and FW2 are misclassified as stateless FW1 and FW2 with high probability of 30.61%.

If we use a different set of firewalls we may have different results for accuracy and misclassification rates. However, the results for this set of firewalls indicate that an attacker can effectively use these two methods to predict a network firewall and design attacks accordingly to either seriously impact performance or exploit possible vulnerabilities. Nevertheless, in practice, the accuracy results will be lower because of the impact of the transmission delay induced by other middleboxes along the probe path. Yet, the attacker can obtain the number of routers in the probe path (using tools such as `traceroute`) and build a similar testbed for learning to reduce such impact.

VII. CONCLUSION AND FUTURE WORK

In this paper, we present methods for finding the firewall characteristics that are introduced by firewall implementations. Such characteristics can be exploited by attackers to identify black box firewalls with high accuracy and launch effective attacks on firewalls. We show two methods for inferring firewall implementation using these characteristics. The first method is based on the firewall decision on a sequence of TCP packets with unusual flags, which could be used as a firewall fingerprint for identification. The second method is based on machine learning techniques. As future work we are willing to propose defense mechanisms from the firewall administrators' perspective, particularly in preventing attackers from gaining information about the deployed firewall and hence forcing attackers to use less-effective and blind attacks. Such mechanisms are designed to increase the chance of incorrect firewall implementation inference by concealing firewall TCP flag fingerprints and obscuring the pattern in probe PPT. To evaluate the effectiveness of these defense mechanisms and measure their impact on firewall performance, we need to conduct extensive experiments, for which we will need to expand our testbed.

REFERENCES

[1] Dan Goodin, "Hacker pierces hardware firewalls with web page", http://www.theregister.co.uk/2010/01/06/web_based_firewall_attack/, 2010.
[2] "Cisco Firewall Services Module DoS vulnerability", http://www.net-security.org/secworld.php?id=10673, 2011.
[3] Faydor, "Nmap: Free network security scanner", http://nmap.org.
[4] Fedor V. Yarochkin, Ofir Arkin, Meder Kydyraliev Shih-Yao Dai, Yennun Huang, and Sy-Yen Kuo, "Xprobe2++: Low volume remote network information gathering tool", in *Proc. DSN*, 2009.
[5] Michal Zalewski, "p0f", http://lcamtuf.coredump.cx/p0f.shtml.
[6] João Paulo S. Medeiros, Agostinho M. Brito, and Paulo S. Motta Pires, "A new method for recognizing operating systems of automation devices", in *Proc. of IEEE EFTA*, 2009.
[7] "Snacktime: A perl solution for remote OS fingerprinting", http://www.planb-security.net/wp/snacktime.html.
[8] Matthew Smart, G. Robert Malan, and Farnam Jahanian, "Defeating TCP/IP stack fingerprinting", in *Proc. of USENIX Security*, 2000.
[9] Elizabeth Millard, "Best firewalls for big enterprises", http://www.ecommercetimes.com/story/business/21764.html, 2003.
[10] Michael R. Lyu and Lorrien K. Y. Lau, "Firewall security: Policies, testing and performance evaluation", in *Proc. Int.Conf. on Computer Systems and Applications (COMPSAC)*, 2000.
[11] Reiner Funke, Andreas Grote, and Hans-Ulrich Heiss, "Performance evaluation of firewalls in gigabit-networks", in *Proc. Symposium on Performance Evaluation of Computer and Telecommunication Systems*, 1999.
[12] Pankaj Gupta and Nick McKeown, "Packet classification using hierarchical intelligent cuttings", in *Proc. Hot Interconnects VII*, Aug. 1999.
[13] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang, "Packet classification using multidimensional cutting", in *Proc. ACM SIGCOMM*, 2003, pp. 213–224.
[14] Pankaj Gupta and Nick McKeown, "Packet classification on multiple fields", in *Proc. ACM SIGCOMM*, 1999.
[15] Florin Baboescu and George Varghese, "Scalable packet classification", in *Proc. ACM SIGCOMM*, 2001.
[16] V. Srinivasan, Subhash Suri, and George Varghese, "Packet classification using tuple space search", in *Proc. ACM SIGCOMM*, 1999, pp. 135–146.
[17] "Mz-traffic generator", http://www.perihel.at/sec/mz.
[18] "Hping", http://www.hping.org.
[19] "Tcpdump public repository", http://www.tcpdump.org.
[20] David Watson, Matthew Smart, G. Robert Malan, and Farnam Jahanian, "Protocol scrubbing: network security through transparent flow modification", *IEEE/ACM Transactions on Networking*, 2004.
[21] Lloyd G. Greenwald and Tavaris J. Greenwald, "Toward undetected operating system fingerprinting", in *Proc. USENIX workshop on Offensive Technologies*, 2007.
[22] Robert Beverly, "A robust classifier for passive TCP/IP fingerprinting", in *Proc. PAM*, 2004.
[23] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng, "Probability estimates for multi-class classification by pairwise coupling.", *Journal of Machine Learning Research*, vol. 5, pp. 975–1005, 2004.
[24] P. Somol, P. Pudil, J. Novovicova, and P. Paclik, "Adaptive floating search methods in feature selection", *Pattern Recognition Letter*, vol. 20, pp. 11–13, 1999.