MASSIMILIANO ADAMO AND
MAURO TABLÒ

Linux vs. OpenBSD

A FIREWALL PERFORMANCE TEST

Massimiliano Adamo graduated in mathematics and has been involved in network security for 10 years. He is currently technology officer of the Institute for Computing Applications "Mauro Picone" of the Italian National Research Council (IAC-CNR).

adamo@iac.rm.cnr.it

Mauro Tablò graduated in computer science and is a senior detective with the Italian Police Forces, where he currently manages the ICT security. His interests include Internet security and cybercrime.

tablo@iac.rm.cnr.it

SECURE, EFFICIENT, AND INEXPENSIVE firewalls can be implemented by means of common PCs running an open source operating system and a packet filter tool, which restricts the type of packets that pass through network interfaces according to a set of rules.

A packet filter confronts a transit packet with a set of rules: when a matching rule is found, the associated decision for the packet is taken (generally, PASS or NO PASS) [1, 3, 4, 5]. The processing time required by the filter grows with the number of rules.

In this article we report the results of a firewall performance test in which we compare the packet processing time of Linux and OpenBSD equipped with their packet filter tools: iptables and PF (Packet Filter), respectively.

Our main goal was to evaluate the packet forwarding speed in both cases and to determine how different conditions affect performance. Therefore tests were made under a variety of conditions and configurations.

Note that a network firewall can pass packets like an L3 device (which we call "routing-firewall") or like an L2 device (which we call "bridging-firewall") [2]. Linux or OpenBSD–based firewalls are often used as routing-firewalls, but they both also have the ability to act as bridging-firewalls, so we tested and compared them in that configuration too.

Testbed

The testbed is composed of three hosts, equipped with Fast-Ethernet cards and connected, according to RFC 2544 [6], as shown in Fig. 1, with two CAT5 UTP crossover cables. There were no other hosts or devices connected to the testbed hosts, so nothing else could influence their behavior. The only packets traversing the wire were those generated by our test hosts.

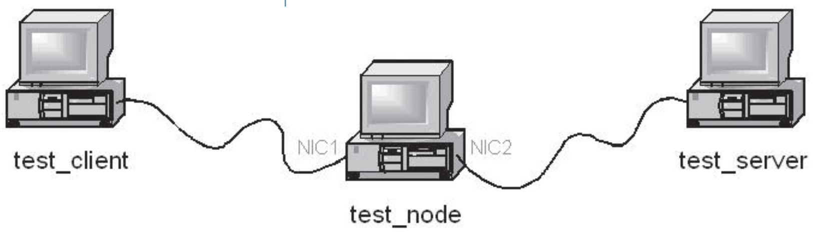

FIGURE 1: BASIC TEST CONFIGURATION

test_client and *test_server* are two Intel Pentium 4 PC (clock speed = 1.5GHz, RAM = 256MB, Network Interface Card = Realtek mod. RTL8139).

The *test_node* configuration is :

- CPU: AMD K6-2
- clock speed: 333MHz
- RAM: 64MB
- Network Interface Card 1: 3com mod. 905c
- Network Interface Card 2: Digital Fast Etherworks PCI mod. DE-500-BA
- OS: either Linux (RedHat 7.3, kernel 2.4.18-3) or OpenBSD (v. 3.3), depending on test session

We used low-performing hardware for host *test_node* (with a slower clock and less RAM than *test_client* and *test_server*) to make sure it represented a bottleneck for the connection. This way, we increased communication delays between client and server, with the purpose of obtaining more apparent differences in measurements.

Efficiency Evaluation

In order to evaluate the firewall efficiency, we measured the delay that host *test_node* caused in packet flow between *test_server* and *test_client*.

Such delay depends on the OS running on host *test_node* (Linux/OpenBSD), on forwarding level (L3 for routing/L2 for bridging), and on "filter/no filter" activity that can be activated on the node.

Tests were performed to measure TCP and UDP throughput performance for different frame sizes and number of rules loaded. We chose four of the frame sizes that are recommended for Ethernet in RFC 2544 [6]: 64, 256, 512, and 1024 bytes. For each of the frame sizes we repeated the test with different rule-set sizes (20, 100, and 500 rules).

Goals

Below is a description of our main goals:

1. Our first goal was to compare performance, in term of throughput, of iptables, a common firewall subsystem for Linux, and PF (Packet Filter), which is the firewall subsystem in the OpenBSD OS. We tested these firewalls in different configurations, with a variable number of filtering rules.

2. The OSes we chose for tests, Linux and OpenBSD, can act as routers or as transparent bridges. For both OSes, we wanted to test whether the bridging is more efficient than the routing feature.

3. Our third goal was to compare the delays that affect TCP packets and UDP datagrams when they traverse a firewall which has rules destined to filter only a single kind of (transport layer) packet (TCP or UDP). For this scope, we measured throughput on the node with the firewall configured with a number of UDP filtering rules but traversed by a flow of TCP packets, and vice versa (UDP traffic with TCP rules).

The Benchmark

To generate the traffic and to measure the throughput, we used Netperf, a network performance benchmark by Hewlett-Packard, available at http://www.netperf.org and designed with the basic client-server model in mind.

By executing the client Netperf on host *test_client,* a control connection was established to the remote system *test_server* (running the server, Netserver) [9] to be used to pass test configuration information and results to and from the remote system.

Once the control connection was up and the configuration information had been passed, a separate connection was established for the actual measurement. Netperf places no traffic on the control connection while a test is in progress [9].

We used Netperf to measure request/response performance.

By executing Netperf from the command line, we could specify some options to define the protocol (TCP or UDP), packet size for requests and responses, and test length (in seconds). A transaction is defined as the exchange of a single request and a single response.

We carried out tests in four different configurations:

1. request and response size = 1024 bytes; protocol = TCP; test time = 120 s.
2. request and response size = 512 bytes; protocol = TCP; test time = 120 s.
3. request and response size = 256 bytes; protocol = TCP; test time = 120 s.
4. request and response size = 64 bytes; protocol = UDP; test time = 120 s.

Test Sessions

We ran nine test sessions. A session is defined as the set of tests made in a given network configuration and systems setup.

In session 1 we connected hosts *test_server* and *test_client* by means of a crossover UTP cable. In this session, *test_client* had IP address 10.0.0.3/24 and *test_server* had 10.0.0.2/24.

Below, we refer to this configuration as the "direct configuration" (or "direct," for short), because the connection between hosts *test_client* and *test_server* is obtained without intermediate devices (router, hub, bridge, switch, etc.).

All further tests (eight more sessions) were done by disposing hosts as in Fig. 1. In this configuration, transactions generated (and measured) by Netperf between *test_client* and *test_server* flowed through *test_node,* which acted as a bottleneck for the connection and introduced a delay: by making a throughput comparison between this case and the "direct" case, we evaluated the delay introduced by host *test_node.*

We repeated every test session three times, obtaining very similar results. For each session, we report only the worst measurement.

SESSION	O.S. FOR *test_node*	CONFIGURATION
1		Direct
2	OpenBSD	Router
3	OpenBSD	Router + Firewall
4	OpenBSD	Bridge
5	OpenBSD	Bridge + Firewall
6	Linux	Router
7	Linux	Router + Firewall
8	Linux	Bridge
9	Linux	Bridge + Firewall

FIGURE 2: TABLE OF TEST SESSIONS AND CONFIGURATIONS

THE "ROUTER" CONFIGURATION

IP address for *test_client* = 10.0.1.2/24.

IP address for *test_server* = 10.0.0.2/24.

IP address for *test_node:nic1* = 10.0.1.1/24.

IP address for *test_node:nic2* = 10.0.0.1/24.

Host *test_client* needs an explicit rule for sending to *test_node* all packets destined to host *test_server.* This is obtained by running the following:

```
test_client# route add 10.0.0.2 gw 10.0.1.1
```

Similarly, host *test_server* needs a routing rule for reaching host *test_client*:

```
test_server# route add 10.0.1.2 gw 10.0.0.1
```

To let host *test_node* act as a router, we have to activate IP forwarding on it.

On OpenBSD this can be done as follows:

```
test_node# sysctl -w net.inet.ip.forwarding=1
```

Whereas on Linux:

```
test_node# sysctl -w net.ipv4.ip_forward=1
```

In the "Router" configuration, the node does not act as a firewall, so no packet-filtering rule is set.

THE "ROUTER + FIREWALL" CONFIGURATION

Without changing the network setup of the "router" configuration, we activated the firewall functionality on the node using a packet filter (iptables on Linux and PF on OpenBSD). Every packet that passed through *test_node* was examined by the packet filter, which decided the action to perform (to drop or to pass it).

To enable packet filtering on OpenBSD [12,16], variable pf in /etc/rc.conf must be set equal to YES:

```
pf=YES
```

Rules contained in /etc/pf.rules are loaded by running:

```
test_node# pfctl -ef /etc/pf.rules
```

and unloaded by running:

```
test_node# pfctl -d
```

Iptables [18] doesn't need a configuration file for loading rules. Although filtering rules can be typed manually one by one from a command prompt, it is better to collect them in a script file.

The iptables filter table uses a linear search algorithm: the data structure is a list of rules, and a packet is compared with each rule sequentially until a rule is found that matches all relevant fields. PF uses a similar search algorithm but, by default, the last matching rule (not the first) decides which action is taken. However, if a rule in PF has the "quick" option set, this rule is considered the last matching rule, and there is no evaluation of subsequent rules.

We started with a set of 20 filtering rules, each one blocking TCP packets destined to a specific port on the server. None of the packets generated by Netperf and exchanged between client and server in our test matched any such rules (a complete description of the rules can be found at http://www.iac.rm.cnr.it/sec/rules.htm). We forced the packet filter to confront every packet in transit with the set of rules and eventually to let it pass. This way, we could measure the delay introduced by the packet filter, which must process the entire list of rules.

We then repeated our tests using lists of 100 and 500 filtering rules for TCP packets and, finally, a list of 500 rules for UDP datagrams.

THE “BRIDGE” CONFIGURATION

After we tested the ‘router’ and “router + firewall” configurations, we set up host *test_node* to act as a transparent bridge.

On OpenBSD [11, 17] this is obtained by running:

```
sysctl -w net.inet.ip.forwarding=0    (deactivates ip-forwarding)
ifconfig xl0 down
ifconfig xl1 down
brconfig bridge0 add xl0 add xl1 up
ifcongig xl0 up
ifconfig xl1 up
```

Linux requires more work[14, 15]. To check that you have bridging and bridge-firewall support compiled into your kernel, go to the directory where your kernel source is installed. For us (with the kernel 2.4.18-3), it is the following path:

```
test_node#  cd /usr/src/linux-2.4
```

In this directory, run:

```
test_node# make menuconfig
```

By navigating through menu items, bring up the “Networking Options” screen and scroll until you see the following:

```
<*> 802.1d Ethernet Bridging
[*]    netfilter (firewalling) support
```

The asterisk to the left in brackets indicates that both options are built in. In other words, our kernel 2.4.18-3 ships with built-in support for bridging and bridge firewalling

If not already available, bridge-firewall support patches for Linux kernels can be obtained from http://bridge.sourceforge.net/download.html. Once downloaded, the patch must be applied and the kernel recompiled.

The next step is to install bridging tools bridge-utils-0.9.3.rpm, downloaded from http://bridge.sourceforge.net/.

Now, we can transform our Linux box in a transparent bridge simply by running:

```
sysctl -w net.ipv4.ip_forward=0   (deactivates ip-forwarding)
ifconfig eth0 down
ifconfig eth1 down
brctl addbr br0
brctl addif br0 eth0
brctl addif br0 eth1
ifconfig br0 0.0.0.0 up
ifconfig eth0 0.0.0.0 up
ifconfig eth1 0.0.0.0 up
```

To deactivate bridging, simply run:

```
ifconfig eth0 down
ifconfig eth1 down
ifconfig br0 down
brctl delif br0 eth1
brctl delif br0 eth0
brctl delbr br0
```

THE "BRIDGE + FIREWALL" CONFIGURATION

By activating a packet filter and loading filtering rules (the same way we did in the "router + firewall" session), the bridge becomes a bridging-firewall.

Test Results

Results are reported as the number of transactions per second.

direct	TCP 1024	4030,80			
	" 512	6465,25			
	" 256	9286,02			
	UDP 64	16020,07			
configuration		BRIDGE		ROUTER	
		Linux	OpenBSD	Linux	OpenBSD
no filter	TCP 1024	2112.32	2026.62	2120.23	2019.17
	" 512	3395.63	3039.21	3430.71	3039.69
	" 256	4909.81	4059.98	4999.73	4057.27
	UDP 64	8216.58	6089.54	8477.04	6089.47
20	TCP 1024	2087.24	1997.71	2111.53	1740.22
	" 512	3340.86	3040.66	3394.70	2514.19
	" 256	4784.87	4047.71	4941.71	3900.61
	UDP 64	8050.35	6081.69	8347.05	6075.76
100	TCP 1024	2093.16	1739.71	2062.29	1739.69
	" 512	3346.76	2541.19	3278.08	2433.69
	" 256	4800.57	4023.15	4685.83	3116.01
	UDP 64	7981.62	6082.07	7705.79	6078.91
500	TCP 1024	1765.26	1353.53	1744.48	1351.02
	" 512	2586.16	1766.63	2534.05	1739.72
	" 256	3383.04	2424.07	3300.93	2050.06
	UDP 64	4809.77	6076.46	4590.34	6079.87
500 UDP	TCP 1024	1876.84	2021.07	1834.17	1744.79
	" 512	2827.54	3036.42	2733.80	3031.48
	" 256	3803.13	4058.83	3642.47	4038.15
	UDP 64	5532.02	3039.65	4966.07	3030.77

As expected, the presence of the node between client and server causes a significant loss of throughput: the number of transactions when *test_client* and *test_server* communicate by means of the intermediate host reduced by half the instances of direct communication in both configurations ("router" and "bridge").

Looking at the experimental results for Linux, we see clearly that the time to classify a packet grows with the number of rules, regardless of the transport protocol (TCP or UDP) and the type of rules (TCP-specific or UDP-specific).

As a matter of fact, iptables compares a packet to the rules, sequentially, starting with the first rule, until a match is found. When a packet matches a rule, then the traversal of rules is stopped and the verdict corresponding to that rule is returned. Since none of the rules in our sets matches the packets that traverse the firewall, in every session iptables compares all packets with *N* rules (where *N* is the number of rules in the list) [4, 5, 10].

PF works in a different and more efficient way. When a rule-set is loaded, the kernel traverses the set to calculate the so-called skip-steps. In each rule, for each parameter, there is a pointer to the next rule that specifies a different value for the parameter. During rule-set evaluation, if a packet does not match a rule parameter, the pointer is used to skip to the next rule that could match, instead of trying the next rule in the set [7]. Analyzing our results, we can see that when the traffic is constituted by UDP datagrams only and all the rules are specific to TCP packets (i.e., the *proto* option is set to *tcp*), we measured a constant throughput for all rule sets (0, 20, 100, and 500 rules): the number of TCP rules in the packet filter doesn't affect the number of UDP transactions between client and server. Similarly, the number of UDP rules in the packet filter doesn't affect the number of TCP transactions.

In general, Linux outperforms OpenBSD for all four configurations. Note that while in OpenBSD the bridging-firewall mode is more efficient than the routing-firewall mode, for Linux there are no significant differences in throughput between bridge-firewalling and router-firewalling.

Conclusion

Linux is, in general, more efficient than OpenBSD. In both router and bridge configurations, it spends less time forwarding packets. Furthermore, iptables filters packets more quickly than PF, with only one exception (in our testing): if the transport-layer protocol of the transit packet, say, UDP, differs from the specified transport-protocol type of a sequence of rules—"protocol type" set to "TCP"in this example—PF ignores those rules and confronts the packet only with the rest of the set, acting more efficiently than Linux, which confronts the packet with all the rules in the set.

This feature of PF is very interesting. UDP-based attacks are very insidious, and most firewalls have rules to prevent many types of UDP datagram from accessing the network. Nevertheless, most traffic from and to a protected network is made up of TCP streams (protocols such as HTTP, SMTP, and FTP all use TCP). In such a case, PF may be more effective: it does not spend processing time comparing TCP packets with the set of rules destined to block UDP datagrams, avoiding delay in processing legitimate packets.

Finally, unlike iptables, PF performs automatic optimization of the rule set, processing it in multiple linked lists [7, 8]. A way to optimize the search on the rule set for iptables is to resort to the "jump" parameter [18] for jumping to a subset of rules (i.e., a chain) reserved for TCP or UDP packets, depending on protocol type.

REFERENCES

[1] Thomas A. Limoncelli, *Tricks You Can Do If Your Firewall Is a Bridge*, *Proceedings of the 1st Conference on Network Administration,* USENIX, April 1999, pp. 47–58.

[2] Angelos D. Keromytis and Jason L. Wright, *Transparent Network Security Policy Enforcement*, *Proceedings of the USENIX Annual Technical Conference, June 2000,* pp. 215–226.

[3] Errin W. Fulp and Stephen J. Tarsa, "Network Firewall Policy Tries," Technical Report, Computer Science Department, Wake Forest University, 2004: http://www.cs.wfu.edu/ ~fulp/Papers/ewftrie.pdf.

[4] Ranganath Venkatesh Prasad and Daniel Andresen, "A Set-Based Approach to Packet Classification," Parallel and Distributed Computing and Systems (PDCS) 2003: http://www.cis.ksu.edu/~rvprasad/publications/pdcs03.ps.

[5] Errin W. Fulp, “Optimization of Network Firewalls Policies Using Directed Acyclical Graphs,” *Proceedings of the IEEE Internet Management Conference, 2005:* http://www.cs.wfu.edu/~fulp/Papers/ewflist.pdf.

[6] S. Bradner and J. McQuaid, “Benchmarking Methodology for Network Interconnect Devices, RFC 2544,” 1999.

[7] Daniel Hartmeier, “Design and Performance of the OpenBSD Stateful Packet Filter (pf),” *Proceedings of 2002 USENIX Annual Technical Conference,* June 2002, pp. 171–180.

[8] Jeremy Andrews, Interview: Daniel Hartmeier, 2002: http://kerneltrap.org/node/477.

[9] Information Network Division of the Hewlett-Packard Company, “Netperf: A Network Performance Benchmark,” Revision 2.1, February 1996: http://www.netperf.org/netperf/training/Netperf.html.

[10] Performance Test Overview for nf-HiPAC, September 2002: http://www.hipac.org.

[11] Brendan Conoboy, Erik Fichtner, IP Filter-Based Firewalls Howto, 2001: http://www.obfuscation.org/ipf/ipf-howto.txt.

[12] Wouter Coene, The OpenBSD Packet Filter Howto, April 2002: http://www.inebriated.demon.nl/pf-howto/pf-howto.txt.

[13] Rusty Russell, Linux netfilter Hacking Howto, Revision v. 1.14, July 2002: http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html.

[14] Nils Radtke, Ethernet Bridge + netfilter Howto, Revision v. 0.2, October 2002: http://www.linux.org/docs/ldp/howto/Ethernet-Bridge-netfilter-HOWTO.html.

[15] Uwe Böhme and Lennert Buytenhenk, Linux Bridge-STP Howto, Revision v. 0.04, January 2001: http://www.linux.org/docs/ldp/howto/BRIDGE-STP-HOWTO/.

[16] The OpenBSD Documentation, PF: The OpenBSD Packet Filter, Revision v. 1.23, February 2005: http://www.openbsd.org/faq/pf/.

[17] The OpenBSD Documentation, Manual Page for brconfig(8).

[18] The Linux Documentation, Manual page for iptables(8).