

Firewall Testing

Cameron Kerr
Telecommunications Programme
University of Otago

May 16, 2005

Abstract

Writing a custom firewall is a complex task, and is something that requires a significant amount of testing to ensure it's doing what you think it's doing. Failure to accurately test can result in gaping holes in your system, and/or denying access wrongfully. This document aims to show you how you can use various command-line tools under Linux to test the correct operation of a stateful firewall.

1 Testing Methodology

For the purpose of this article, I will assume you have broken up the traffic going through, to, and from your router into various *flows*, so that traffic going from outside to inside is in a different chain from the inside to outside flow.

Breaking down the traffic in this manner will help to manage and audit the firewall against your policy. When testing your firewall, test against each of the policies that are in place, or against any pseudocode created from your policies.

1.1 Inspect Counters

Each rule in the firewall has two counters associated with it, a packet and byte counter. The byte counter can be used for bandwidth accounting. We shall concern ourselves with the packet counter, as every time a packet matches that rule, the counter is incremented.

The basic method to diagnose how packets are being filtered, is to first zero the counters, run a single test, then inspect the counters to see which rules did (or did not as is more often the case) match.

```
root@ralf:~# iptables -Z
Now test a single rule
root@ralf:~# iptables -L -vZ | less
_____ Omitted _____
```

1.2 How to test each rule

When I tell people how to do the above test (inspecting counters), they often ask me how to run each test. I always answer “by using it”. So, if your policy for that flow is to “Accept DNS requests, DHCP requests, and SSH only (dropping by default)”, then your tests would be as follows.

1. Try using `dig` to test that you can get to the DNS server. This will also test whether replies can make it back, so beware of that if you find it times out. You can use the `@` notation to specify a particular server to query.

```
root@client2:~# dig @172.16.0.2 ralf.internet
_____ Omitted _____
```

2. Try getting an IP address via DHCP, by starting a DHCP client, using one of `dhcpcd`, `dhclient`, or `pump`. This will test two rules, the request and reply.
3. Try connecting via SSH. If this is a port-forwarded connection, then SSH to the firewall. When you log in, you should find that the hostname will be the machine to where the connection was forwarded to.

2 Port Scanner – nmap

To see what can be seen from a particular network, we use a port scanner. This can both tell us what machines are seeable (a so-called “ping-sweep”, though it may not always use ICMP echo-requests, but UDP as well), and it will also tell us what ports are open on a machine.

2.1 Ping sweep

In our trivial network we only have one machine on each attached network, so doing this is of limited benefit, although it can be useful to see what can

be seen from the internet. To do a ping sweep, we can use the following command. Here is a real-life example.

```
cameron@mithrak:~$ nmap -sP 10.18.0.0/24
Starting nmap 3.50 ( http://www.insecure.org/nmap/ )
Host belgarath.localdomain (10.18.0.1) appears to be up.
Host ps01.localdomain (10.18.0.2) appears to be up.
Nmap run completed -- 256 IP addresses (2 hosts up)
```

2.2 TCP Port scan

To scan TCP is fairly easy. Note that some fairly secure systems will take steps to avoid being port-scanned, so you may want to play with the timing values to appear more stealthy. The simplest way to see what ports you can connect to is to simply connect to every port. As this takes a long time, often a reduced set of ports is scanned.

```
cameron@mithrak:~$ nmap 10.18.0.1
Starting nmap 3.50 ( http://www.insecure.org/nmap/ )
Interesting ports on belgarath.localdomain (10.18.0.1):
(The 1654 ports scanned but not shown below are filtered)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
3128/tcp  open  squid-http

Nmap run completed -- 1 IP address (1 host up)
```

The other way is to do a syn-scan (also known as a half-open scan), where the SYN packet is sent, but the resultant SYN-ACK packet is not ACK'd. This is often called a stealth scan, but you shouldn't use that term, as there are other ways of appearing stealthy (see the nmap manual page).

```
root@mithrak:~# nmap -sS 10.18.0.1
_____ Omitted _____
```

2.3 UDP Port Scan

A UDP scan must be done very slowly, so we generally just select a smallish range of ports to scan. See the manual page for the reason it must be done slowly.

```
root@mithrak:~# nmap -sU -F -r -v -T Polite 10.18.0.1
_____ Omitted _____
```

Because of the mechanics of this, you may well find you get a *lot* of false positives if your firewall would drop such packets. It also takes a lot of time – 800 seconds.

3 Packet Sniffer – tcpdump

When testing, it is often useful to know what is arriving at your interface, and what is going out of it. You should be aware that when we use this tool, we receive it as it going through the datalink layer. This means that for outgoing packets that have been through a NAT, the destination address will be the public address, not the internal address. Similar comments apply for incoming packets.

The simplest way to use `tcpdump` is to run it and see everything. On a firewall, you will want to specify which interface to use, by default it uses “eth0” on Linux.

```
root@mithrak:~# tcpdump -i eth0
tcpdump: listening on eth0
22:48:04.325314 belgarath.localdomain.domain > ←
    mithrak.localdomain.60276: 12093 NXDomain 0/1/0 (DF)
22:48:04.325468 mithrak.localdomain.60276 > ←
    belgarath.localdomain.domain: 12094+ ←
    PTR? 2.2.168.192.in-addr.arpa. (42) (DF)
^C
16 packets received by filter
0 packets dropped by kernel
```

The output can be quite hard to follow, but once you get used to it, you can use it to quickly see what’s going on. If you want to take a closer look later on, you can write the packets to a file, copy the file to a workstation with `ethereal` installed, and open it on that. One thing you may want to change is the snap length, which controls how much of the packet you read

in. Changing it to 1500 will get the whole packet on Ethernet networks. The default value may be too small if you want Ethereal to show application level protocol headers.

```
— Make sure you SSH in with X11 Forwarding (-X) —  
root@ralf:~# tcpdump -i eth0 -s 1500\  
> -w /mnt/home/cameron/packets.pcap  
^C  
And back on medussa, run ethereal  
cameron@medussa:~# ethereal packets.pcap &
```

Packet sniffers allow us to specify a pattern so it will only capture or display those packets we have an interest in. We can specify some fairly complex expressions, but we'll keep this discussion very simple. For more examples, see the tcpdump manual page.

```
root@client1:# tcpdump -i eth0 tcp and port ssh
```

You can run `tcpdump` on the target system, to see if the request gets through, and whether or not a packet reaches the host. It will not tell you if a packet makes it through the firewall.