

# Packet Crafting for Firewall and IDS Audits

Don Parker

With the current threat environment that home and corporate users face today, having a firewall and IDS is no longer a luxury, but rather a necessity. Yet many people do not really take the time to make sure though that these lines of defense are indeed working properly. After all, it is very easy to invalidate your router's entire ACL list by making a single misconfigured entry. The same can be said for your firewall, whereby one poor entry into your iptables script, for example, could leave you vulnerable. Have you properly configured certain options which may be available with your firewall? All of these questions can be answered, and more importantly verified through the use of packet crafting. What this will allow you to do is manually verify that all is working well with your firewall and IDS, and that each is properly configured.

It is best to not blindly rely on the output of certain automated tools when auditing devices that safeguard your valuable computing assets. I would compare this to an analogy where one manually checks his door locks and the burners on the stove before going to bed, instead of waiting for a burglar or a fire alarm to be woken up. You know you've done everything required to safeguard your environment, but at the end of the day you still want to make sure. Packet crafting, when used to audit your network, cannot verify all conditions affecting your firewall and IDS, but the process can do so a fair number of them.

This article is the first of a two-part series that will discuss various methods to test the integrity of your firewall and IDS using low-level TCP/IP packet crafting tools and techniques. The focus is on a Linux environment but the process will work similarly well with other Unix-like environment too.

## Benefits of Packet Crafting

There are some side benefits to learning how to audit your firewall and IDS though the use of packet crafting. To learn how to effectively use tools such as *hping*, and reliably interpret their data, you force yourself to learn more about TCP/IP. Learning more about the base unit for computer communications, the packet, is a laudable goal for anyone wishing to further his knowledge of computers. That being said, I will not assume that all of the readers of this magazine have a good grasp of TCP/IP. Over the course of this article there will be detailed explanations of the output which will be shown. All of the fields shown in the Snort and *tcpdump* output will be explained clearly and concisely. You might say this will be as much about understanding TCP/IP as it is about learning how to test your firewall and IDS ruleset.

Several examples will be shown for both a firewall and IDS. Some very generic ones are used, as this document is meant to show you how to use *hping*, Snort, and *tcpdump*. Therefore, the examples listed below are a good starting point. After having played with the below noted examples you should feel comfortable enough to actively test your own firewall and IDS. Note that there are also automated tools out there which can do this for you. It is important, though, that you know how to do it yourself and are able to monitor and analyze the output. This will give you a feeling of confidence knowing that your perimeter defenses are indeed working as advertised.

## Testing Your Firewall - First Example

We will now start off by showing several examples of how to test your firewall for various conditions. The first one will be to see if you have port 80 open through your firewall. The second example will check to see if your firewall has port 53 open as well, and then we'll conclude part one of this article.

## Assumptions

Please note that these tests were performed on Linux using SuSE Professional 9.0 with the standard iptables firewall ruleset in place. I am not including a sample IPTables syntax here as you may be using IPChains on Linux instead, or perhaps a firewall appliance, a commercial firewall or some other type of solution. It would also be difficult to put down a rule that takes into account all of the other rules as well, which is the primary reason for not having provided a sample rule syntax here. To that end, each

# Packet Crafting for Firewall and IDS Audits

Don Parker

condition being tested will be explained so as to give you the context in which it was tested. In part two we'll look at Snort; the Snort build used was Snort 2.1.0 with a standard default ruleset in place.

The below noted example shows a packet sent to port 80. The packet is silently dropped as it should be, due to the configuration of our SuSE firewall. Normal behavior for a TCP packet sent to a port with no listening service would be a reset packet being sent back to the originating machine.

Shown below is a cut and paste of my term window and the syntax used to invoke *hping*. I will explain the *hping* switches used below for this example, however unless there is a major difference in syntax I will not elaborate again on *hping* syntax.

**hping** This is the name of the packet crafting program.  
**-S** This tells *hping* to send a SYN packet.  
**192.168.1.108** This is the destination address which will receive a SYN packet.  
**-p 80** This specifies the port on the destination computer, in this case port 80.  
**-c 1** This switch allows you to control exactly how many packets to send, in this case just one.

Note as well the output of the *hping* below the command line syntax. This just shows the address that the packet is going to, that S is set (SYN packet), that there are 40 bytes in the packet (normal TCP/IP header size) and that there are 0 data bytes in the packet.

The last remaining bit of our *hping* output, as seen below, are the statistics for the packet crafting you just did using *hping*. It tells you 1 packet was transmitted, 0 packets were received back, and that it was %100 packet loss. It also gives you the round trip time if any packets were sent back.

```
monkeylabs:/home/don # hping -S 192.168.1.108 -p 80 -c 1 HPING 192.168.1.108 (eth0
192.168.1.108): S set, 40 headers + 0 data bytes --- 192.168.1.108 hping statistic --
- 1 packets tramitted, 0 packets received, 100% packet loss round-trip min/avg/max =
0.0/0.0/0.0 ms
```

Having used the above noted command syntax, seen below is what the packet looks like as it is sent from the machine that it was crafted on using *hping*. The *tcpdump* syntax noted below to invoke *tcpdump* will now be explained.

**tcpdump** The name of the program  
**-n** This means keep the ip addresses in numerical format vice resolving them.  
**X** This means to dump the information in both HEX and ASCII.  
**v** Be verbose in output ie: show everything in the packet.  
**s** This is for the snaplength ie: how much of the packet do you want to capture.  
**0** By putting in 0 for snaplength *tcpdump* will capture the default length for your computer. In the case of mine that would be 1518.  
**tcp** Specify that you want only TCP packets, not UDP or ICMP, only TCP.  
**and host**  
**192.168.1.100** This specifies that you want packets copied from 192.168.1.100  
**and 192.168.1.108** This specifies that you also want packets from 192.168.1.108.

Using the above two IP addresses with the noted syntax stipulates to *tcpdump* that you want to collect packets that go between 192.168.1.100 and 192.168.1.108 only. This cuts down on chatter that you may pick up on from, say, your ISP's DNS server or your switch's ARP packets.

# Packet Crafting for Firewall and IDS Audits

Don Parker

To help you in reading *tcpdump* data I will fully explain all the fields in the packet below, just as I have explained the *tcpdump* syntax used. Starting with the time field we will now see what every field means.

<b>10:07:30.171332</b>	time that the packet was received, right down to the microsecond
<b>192.168.1.100.1321</b>	source ip address and port of the sender's computer
<b>&gt;</b>	just a director telling you which way the communications are going in
<b>192.168.1.108.80</b>	destination computer and port which we are sending to
<b>S</b>	denotes that we are sending a SYN packet
<b>[tcp sum ok]</b>	means the TCP checksum was good and the packet is not corrupted
<b>1907499058:1907499058</b>	the TCP sequence number
<b>(0)</b>	the number of data bytes sent in this packet
<b>win 512</b>	the amount that the sending computer will send before requiring an ACK packet back from the destination computer
<b>[tos 0x8]</b>	type of service field
<b>ttl 64</b>	number of hops that the packet has to reach its destination
<b>id 45106</b>	IP id number it is used for fragmentation reassembly
<b>len 40</b>	length of the packet

```
/home/don # tcpdump -nXvs 0 tcp and host 192.168.1.100 and 192.168.1.108 tcpdump:
listening on eth0 10:07:30.171332 192.168.1.100.1321 > 192.168.1.108.80: S [tcp sum
ok] 1907499058:1907499058(0) win 512 [tos 0x8] (ttl 64, id 45106, len 40) 0x0000
4508 0028 b032 0000 4006 4675 c0a8 0164 E..(.2..@.Fu...d 0x0010 c0a8 016c
0529 0050 71b2 2032 53e1 85d2 ...l.).Pg..2S... 0x0020 5002 0200 b8b0 0000
P.....
```

The below noted screenshot is from the destination computer that we are testing, showing what happened when this packet was received. It was dropped by default as there is no service presently there, and that is how IPTables is configured to deal with packets sent to ports offering no services by default on SuSE.

```
Mar 2 10:06:40 linux kernel: SuSE-FW-DROP-DEFAULT IN=eth0 OUT=
MAC=00:50:da:c5:9d:8b:00:0c:6e:8c:d4:61:08:00 SRC=192.168.1.100 DST=192.168.1.108
LEN=40 TOS=0x08 PREC=0x00 TTL=64 ID=45106 PROTO=TCP SPT=1321 DPT=80 WINDOW=512
RES=0x00 SYN URGP=0
```

This is the packet as it arrived at the test machine. All is normal, as can be seen:

```
/home/don # tcpdump -nXvs 0 tcp and host 192.168.1.108 and 192.168.1.100 tcpdump:
listening on eth0 10:06:40.474204 192.168.1.100.1321 > 192.168.1.108.80: S [tcp sum
ok] 1907499058:1907499058(0) win 512 [tos 0x8] (ttl 64, id 45106, len 40) 0x0000
4508 0028 b032 0000 4006 4675 c0a8 0164 E..(.2..@.Fu...d 0x0010 c0a8 016c
0529 0050 71b2 2032 53e1 85d2 ...l.).Pg..2S... 0x0020 5002 0200 b8b0 0000
0000 0000 0000 P.....
```

So as we can see when the packet was sent, it was indeed received by the destination machine, however it was silently discarded.

## Testing Your Firewall - Second Example, Probing Port 53 UDP

Now we'll try a second example and probe port 53 UDP. Note below the *hping* syntax that was used, and its ensuing output as well:

# Packet Crafting for Firewall and IDS Audits

Don Parker

```
monkeylabs:/home/don # hping -2 192.168.1.108 -p 53 -c 1 HPING 192.168.1.108 (eth0
192.168.1.108): udp mode set, 28 headers + 0 data bytes --- 192.168.1.108 hping
statistic --- 1 packets tramitted, 0 packets received, 100% packet loss round-trip
min/avg/max = 0.0/0.0/0.0 ms monkeylabs:/home/don #
```

We have now sent a UDP packet to port 53 to see if it will also silently drop the packet. As we can see from the *hping* output and the firewall information below, it was indeed discarded as port 53 UDP is closed off.

```
Mar  2 10:24:30 linux kernel: SuSE-FW-DROP-DEFAULT IN=eth0 OUT=
MAC=00:50:da:c5:9d:8b:00:0c:6e:8c:d4:61:08:00 SRC=192.168.1.100 DST=192.168.1.108
LEN=28 TOS=0x10 PREC=0x00 TTL=64 ID=47873 PROTO=UDP SPT=2180 DPT=53 LEN=8
```

This is the packet as it was received on the machine we are testing:

```
/home/don # tcpdump -nXvs 0 udp and host 192.168.1.108 and 192.168.1.100 tcpdump:
listening on eth0 10:24:30.172588 192.168.1.100.2180 > 192.168.1.108.53: [udp sum ok]
0 [0q] (0) [tos 0x10] (ttl 64, id 47873, len 28) 0x0000 4510 001c bb01 0000 4011
3b9f c0a8 0164 E.....@.;....d 0x0010 c0a8 016c 0884 0035 0008 7304 0000
0000 ....1...5..s..... 0x0020 0000 0000 0000 0000 0000 0000 0000
.....
```

This is the packet as it was sent out from the machine we are using to test the security of the firewall:

```
monkeylabs:/home/don # tcpdump -nXvs 0 udp and host 192.168.1.100 and 192.168.1.108
tcpdump: listening on eth0 10:25:19.887529 192.168.1.100.2180 > 192.168.1.108.53:
[udp sum ok] [|domain] [tos 0x10] (ttl 64, id 47873, len 28) 0x0000 4510 001c bb01
0000 4011 3b9f c0a8 0164 E.....@.;....d 0x0010 c0a8 016c 0884 0035 0008
7304 ....1...5..s.
```

Normal behaviour when a UDP packet is sent to a port that is not listening is that an ICMP port unreachable message is sent. In our case however this does not happen as the test machine has its IPTables script set to silently discard these types of packets.

As you are beginning to see, packet crafting is a rather excellent tool to use in verifying your firewall ruleset. Especially one which can get rather complex and granular as an IPTables script can become.

We have looked at two examples of testing your firewall using *hping* and *tcpdump*. Next time, in part two of this article, we'll look at another firewall example and then move on to test the Snort IDS using similar methods as were outlined here. Stay tuned.

This is the second of a two-part article series that discusses various methods of testing the integrity of your firewall and IDS, using low-level TCP/IP packet crafting tools and techniques. We will now continue the discussion with a third test of the firewall, using the same tools as noted above, and then move on to test your IDS signatures and detection ability. Note that the focus here is on a Linux environment, but the process is similar with other Unix-like firewall/IDS environments as well.

# Packet Crafting for Firewall and IDS Audits

Don Parker

## Testing Your Firewall - Third Example, ICMP Echo Requests

The example shown below is a simple ICMP echo request to see if a machine is alive, in this case our test machine.

As was used extensively in part one of this article, we'll be using *hping* and *tcpdump* to perform the tests. The syntax and output of *hping* for a simple ICMP echo request is shown below:

```
monkeylabs:/home/don # hping -1 192.168.1.108 -c 1 HPING 192.168.1.108 (eth0
192.168.1.108): icmp mode set, 28 headers + 0 data bytes len=46 ip=192.168.1.108
ttl=64 id=122 icmp_seq=0 rtt=0.4 ms --- 192.168.1.108 hping statistic --- 1 packets
tramitted, 1 packets received, 0% packet loss round-trip min/avg/max = 0.4/0.4/0.4 ms
```

This time there is one packet received as was noted above, and the round trip time is shown. By going with this it looks like our ICMP echo request was indeed allowed through the firewall.

Below you will note the packet as it was sent on our *hping* machine. The output shows the packet was received on our *hping* test machine from the machine we just pinged from. The *tcpdump* syntax is shown below once again.

```
monkeylabs:/home/don # tcpdump -nXvs 0 ip and host 192.168.1.100 and host
192.168.1.108 tcpdump: listening on eth0 11:10:37.458058 192.168.1.100 >
192.168.1.108: icmp: echo request (ttl 64, id 51585, len 28) 0x0000 4500 001c c981
0000 4001 2d3f c0a8 0164 E.....@.-?...d 0x0010 c0a8 016c 0800 5dd0 9a2f
0000 ...l..]../. 11:10:37.458260 192.168.1.108 > 192.168.1.100:
icmp: echo reply (ttl 64, id 117, len 28) 0x0000 4500 001c 0075 0000 4001 f64b c0a8
016c E...u..@..K...l 0x0010 c0a8 0164 0000 65d0 9a2f 0000 0000 0000
...d..e../. 0x0020 0000 0000 0000 0000 0000 0000 0000
.....
```

We can deduce from the above two packets that the destination computer's firewall did allow the ICMP packet through, as we received an answer packet back from it. I did not show the destination computer receiving the packet as we have successfully proven that the firewall is accepting our test criteria. Nor is there any firewall output to show as it did not cause any firewall rulesets to trigger.

## IDS Signature Test - Reserved Flag Test

We will now begin to do some testing of our IDS, which in this case is Snort. This test was done with Snort's default ruleset.

The first test that we will do is to see if our IDS triggers as it is supposed to when it receives packets with the ECN and CWR flags active in the 13th byte offset of the TCP header.

*Hping* syntax is as noted below:

```
monkeylabs:/home/don # hping -X -Y 192.168.1.108 -p 80 -c 1 HPING 192.168.1.108 (eth0
192.168.1.108): XY set, 40 headers + 0 data bytes --- 192.168.1.108 hping statistic -
-- 1 packets tramitted, 0 packets received, 100% packet loss round-trip min/avg/max =
0.0/0.0/0.0 ms
```

Noted below is the packet as seen when it was sent by the machine which had crafted it:

```
/home/don # tcpdump -nXvs 0 tcp and host 192.168.1.100 and 192.168.1.108 tcpdump:
listening on eth0 08:42:00.081599 192.168.1.100.1215 > 192.168.1.108.80: WE [tcp sum
```

# Packet Crafting for Firewall and IDS Audits

Don Parker

```
ok] win 512 (ttl 64, id 29279, len 40) 0x0000 4500 0028 725f 0000 4006 8450 c0a8
0164 E.(r_..@..P...d 0x0010 c0a8 016c 04bf 0050 460e ae8b 6c89 fe96
...l...PF...l... 0x0020 50c0 0200 c43a 0000 P.....
```

Note the flags WE in the above packet. These signify that the ECN and CWR flags are set in this packet. Now, note the packet below comes from the IDS test machine.

```
/home/don # tcpdump -nXvs 0 tcp and host 192.168.1.108 and 192.168.1.100 tcpdump:
listening on eth0 08:40:58.589496 192.168.1.100.1215 > 192.168.1.108.80: WE [tcp sum
ok] win 512 (ttl 64, id 29279, len 40) 0x0000 4500 0028 725f 0000 4006 8450 c0a8
0164 E.(r_..@..P...d 0x0010 c0a8 016c 04bf 0050 460e ae8b 6c89 fe96
...l...PF...l... 0x0020 50c0 0200 c43a 0000 0000 0000 0000
P.....
```

The packet is as it should be: a mirror image of the packet sent by the machine on which it was crafted.

Now let's take a look at the Snort output. Below you will note the Snort output shown as it received the packet. You will see that there is a number 1 and 2 that are set in the output. This signifies that the proper bits for this byte are set. In this case it is bit 128 and 64 in the 13th byte offset in the TCP header. This is where all the flags are contained, in other words, flags such as SYN, FYN, PSH and such.

There is a great deal of information logged in the Snort output. Starting on the first line and going from left to right I will explain the fields themselves. We start off with the date and time field, down to the microsecond (just like *tcpdump*). This is followed by the MAC addresses of the sending and receiving computers. Next is the type field which stands for DoD ethernet, and then the length of the packet itself.

Now the source IP address and port is followed by the destination IP address and port. After that, you see TCP which is for the TCP packet and a time to live of 64. The type of service is shown to be zero.

Then the IP id number is shown as 29279, and after it is the IP header length of 20 bytes. DgmLen stands for the datagram length. As mentioned above, the 1 and 2 mean that the bit positions 128 and 64 are set in this packet. Then we have the sequence number, as well as the acknowledgement number. Closing out this info is the window size and TCP length.

Here is the Snort output from our first example:

```
03/03-08:40:58.589496 0:C:6E:8C:D4:61 -> 0:50:DA:C5:9D:8B type:0x800 len:0x3C
192.168.1.100:1215 -> 192.168.1.108:80 TCP TTL:64 TOS:0x0 ID:29279 IpLen:20
DgmLen:40 12***** Seq: 0x460EAE8B Ack: 0x6C89FE96 Win: 0x200 TcpLen: 20
```

In the `/var/log/snort/alert` file, the output noted below was logged as a result of the signature this packet triggered when it was parsed by Snort:

```
linux:/var/log/snort # more alert [**] [111:1:1] (spp_stream4) STEALTH ACTIVITY
(unknown) detection [**] 03/03-08:40:58.589496 0:C:6E:8C:D4:61 -> 0:50:DA:C5:9D:8B
type:0x800 len:0x3C 192.168.1.100:1215 -> 192.168.1.108:80 TCP TTL:64 TOS:0x0
ID:29279 IpLen:20 DgmLen:40 12***** Seq: 0x460EAE8B Ack: 0x6C89FE96 Win: 0x200
TcpLen: 20
```

Our example was used to see if the Snort IDS would indeed trigger and log this as an alert, on receipt of a packet with the ECN and CWR flags set. As noted, it did indeed perform as it should have.

# Packet Crafting for Firewall and IDS Audits

Don Parker

## Second IDS Signature Test - LSRR Packets

We will now test a different IDS signature. To be specific, we will see if Snort will indeed detect LSRR (Loose Source Record Route, commonly known as loose source routed) packets as it is supposed to.

*Hping* syntax as noted is below:

```
monkeylabs:/home/don # hping -S --lsrr 192.168.1.108 192.168.1.102 -p 25 -c 1 HPING
192.168.1.102 (eth0 192.168.1.102): S set, 40 headers + 0 data bytes ---
192.168.1.102 hping statistic --- 1 packets tramitted, 0 packets received, 100%
packet loss round-trip min/avg/max = 0.0/0.0/0.0 ms
```

The packets as they looked like when leaving the hping machine:

```
monkeylabs:/home/don # tcpdump -nXvs 0 ip and host 192.168.1.100 and 192.168.1.108
tcpdump: listening on eth0 09:23:35.134313 192.168.1.100.1636 > 192.168.1.108.25: S
[tcp sum ok] 384787509:384787509(0) win 512 (ttl 64, id 57275, len 48, optlen=8
LSRR{192.168.1.102#} EOL) 0x0000 4700 0030 dfbb 0000 4006 7b22 c0a8 0164
G..0.....@.{"....d 0x0010 c0a8 016c 8307 08c0 a801 6600 0664 0019
...l.....f..d.. 0x0020 16ef 6435 3ac2 97be 5002 0200 d59f 0000
..d5:...P.....
```

The Snort output below is shown as Snort saw the packet:

```
03/03-09:22:33.600331 0:C:6E:8C:D4:61 -> 0:50:DA:C5:9D:8B type:0x800 len:0x3E
192.168.1.100:1636 -> 192.168.1.108:25 TCP TTL:64 TOS:0x0 ID:57275 IpLen:28
DgmLen:48 IP Options (1) => LSRR *****S* Seq: 0x16EF6435 Ack: 0x3AC297BE Win:
0x200 TcpLen: 20
```

Taken from the alert file in `/var/log/snort/`, we see the output below:

```
[**] [1:501:2] MISC source route lssre [**] [Classification: Potentially Bad
Traffic] [Priority: 2] 03/03-09:22:33.600331 0:C:6E:8C:D4:61 -> 0:50:DA:C5:9D:8B
type:0x800 len:0x3E 192.168.1.100:1636 -> 192.168.1.108:25 TCP TTL:64 TOS:0x0
ID:57275 IpLen:28 DgmLen:48 IP Options (1) => LSRR *****S* Seq: 0x16EF6435 Ack:
0x3AC297BE Win: 0x200 TcpLen: 20 [Xref =>
http://www.whitehats.com/info/IDS420][Xref => http://cve.mitre.org/cgi-
bin/cvename.cgi?name=CVE-1999-0909][Xref => http://www.securityfocus.com/bid/646]
```

And here is what was received on the test machine:

```
linux:/home/don # tcpdump -nXvs 0 ip and host 192.168.1.108 and 192.168.1.100
tcpdump: listening on eth0 09:22:33.600331 192.168.1.100.1636 > 192.168.1.108.25: S
[tcp sum ok] 384787509:384787509(0) win 512 (ttl 64, id 57275, len 48, optlen=8
LSRR{192.168.1.102#} EOL) 0x0000 4700 0030 dfbb 0000 4006 7b22 c0a8 0164
G..0.....@.{"....d 0x0010 c0a8 016c 8307 08c0 a801 6600 0664 0019
...l.....f..d.. 0x0020 16ef 6435 3ac2 97be 5002 0200 d59f 0000
..d5:...P.....
```

So as we can see, the Snort IDS definitely picked up the receipt of our LSRR packets. Now we can be confident that even Snort's default ruleset is working as advertised.

## Dispelling Some Myths

Finally, I would like to take the time to dispel some myths here about packet crafting. The first myth is about buffer overflows. You cannot send buffer overflows using a packet crafting tool because the three-

# Packet Crafting for Firewall and IDS Audits

Don Parker

way TCP/IP handshake needs to be completed for an exploit to work. This is why the code needs to be compiled, for within the code itself are the systems calls to handle the 3 way TCP/IP handshake.

On most stacks today the sequence numbers are pseudo-random and therefore difficult to predict, making the ISN prediction attack impractical, though not impossible. So in light of this any data you inject into a packet will be ignored by the destination machine unless you have successfully completed the TCP/IP handshake.

## Conclusion

Hopefully you can now see the clear benefits of crafting packets, and what it will bring to you and your security posture. Not only that, but along the way you will also learn about the all-important subject of low level TCP/IP. The examples that were shown in this article, are just a starting point -- each person's network or setup can be unique, and the firewall rulesets or IDS signatures will be different. It will be up to you to use the knowledge gained to actively probe your own defenses.

I sincerely hope you found this article series of use to you. Feel free to contact me, if you like, with regards to it.