Sniff Your own Networks with Tcpdump

by Boris Loza, PhD, CISSP

A sniffer is any device, software, or hardware that listens to all packets traveling along a network. Bad guys use sniffers to breach your security by capturing and analyzing all network traffic. Good guys use sniffers to protect the network.

Today you can find many different types of this software. Some sniffers are used by crackers, and others by network administrators. Some sniffers are free, and some are expensive. Further, some are simple. There are companies who produce entire suites of sniffer applications designed to diagnose network problems. In this article, we'll show you how to use a freely available sniffer--`tcpdump`.

About tcpdump

The `tcpdump` program was written by Van Jacobson, Craig Leres, and Steven McCanne, all of the Lawrence Berkeley Laboratory at the University of California at Berkeley. Like any other sniffers, `tcpdump` captures packets on the network by placing a local interface in promiscuous mode. Normally, your network interface will ignore any pockets that aren't addressed to your system. By placing your interface in promiscuous mode, `tcpdump` captures all packets, regardless of address, and allows you to examine their headers. The `tcpdump` program also allows you to extract particular types of network traffic based on header information.

Installing tcpdump

`You can download tcpdump` from tcpdump version found here was 3.4. You can also download `libpcap.tar.Z` from the same destination (*url;ftp://ftp.ee.lbl.gov/'>ftp://ftp.ee.lbl.gov/tcpdump.tar.Z*eurl. The latest `tcpdump` version found here was 3.4. You can also download `libpcap.tar.Z` from the same destination (*url;ftp://ftp.ee.lbl.gov/). This application is a system-independent interface for user-level packet capture, and will be used in conjunction with `tcpdump`. You must install `libpcap` first. After downloading files, type:

```
#zcat libpcap.tar.Z | tar xvf -
#zcat tcpdump.tar.Z | tar xvf -
```

This extracts the `tcpdump` and `libpcap` distributions. Then, go to the `libpcap-0.4` directory and read the `INSTALL` file. In most cases, it's enough just to type `./configure` followed by `make install, make install-incl,` and optionally `make install-man`. Doing so installs

libpcap and the manual entry. Now you can go to the tcpdump-3.4 directory to start building tcpdump. Type ./configure and build tcpdump by running make.

If everything builds okay, become root and type make install and make install-man. Doing so installs tcpdump and the manual pages. If you decide to install tcpdump in a directory other than /usr/local/sbin, edit the BINDEST path in Makefile.in and run ./configure.

Tcpdump output format

Running tcpdump is easy. After you type tcpdump at root, you'll see the output shown in **Listing A**.

Listing A: *Sample output from tcpdump*

```
#tcpdump
tcpdump: listening on hme0
11:17:36.965387  192.168.10.1.1028 > 192.168.10.5.700: udp 82
11:17:42.645580 pine.tree.com.34342 > birch.tree.com.telnet:
=>S 1819099388:1819099388(0)
win 8760 <mss 1460> (DF)
11:17:50.011072 pine.tree.com > oak.tree.com: icmp: echo request (DF)
11:17:50.011091 oak.tree.com > pine.tree.com: icmp: echo reply (DF)
11:17:55.870599 arp who-has 192.168.10.1 tell 192.168.10.55
^C
```

You'll find that tcpdump generally produces one line of output for each packet that it sees. For each connection, tcpdump will always display (except in very special cases) a timestamp, a source IP address, a destination IP address, and some additional information about the packet (such as protocol and port information). Further, tcpdump has a default standard output based on the protocol. On the tcpdump output in **Listing A**, we can see the UDP, TCP, ICMP, and ARP protocol information.

Table A: *Each record generated in our sample has the following fields*

Field	Data
Timestamp	11:17:36.965387
Source.Port > Dest.Port:	192.168.10.1.1028 > 192.168.10.5.700:
Protocol	udp
Bytes of data	82

In this example, tcpdump is reporting traffic from host 192.168.10.1 on port 1028, to host 192.168.10.5 on port 700, using the UDP protocol. The direction of the traffic is indicated by the greater than symbol (>) between the source and destination address. The timestamp is in the format of hour:minutes:seconds.millionth_of_seconds. UDP packets may or may not have udp in the output.

win 8760 <mss 1460> (DF) **Table B** shows the fields that are present for a TCP packet.

Table B: *tcpdump generates the following record for the TCP protocol*

Timestamp	11:17:42.645580
Source.Port > Dest.Port	pine.tree.com.34342 > birch.tree.com.telnet:
Flags	S
Begin-Seq#:End-Seq#(Bytes)	1819099388:1819099388(0)
Options	

Table B shows the fields that are present for a TCP packet. This is identical to the UDP record as far as timestamp, source and destination host, and port. What distinguishes the TCP format from the others are the TCP flags, sequence numbers, acknowledgements, acknowledgement numbers, and TCP options.

In this record, we see the flag `SYN` or `S` set following the destination port telnet. The `SYN` flag indicates the beginning of a `telnet` session. Other possible flags are `P` for `PUSH` that sends data, `R` `RESET` that aborts a connection, and `F` `FIN` that terminates a connection. If you see a period '.' in the flag field, it simply means that none of the `PUSH`, `RESET`, `SYN,` or `FIN` flags are set.

You'll see that 1819099388:1819099388 is the beginning:ending set of sequence numbers. The ending sequence number is the sum of the initial sequence number plus the number of TCP data bytes sent in this segment (in this case 0, which is why both numbers are equal).

Finally, there is a TCP options field. We see that pine.tree.com is advertising a window size of 8760 bytes, a maximum segment size of 1460 bytes, and no fragmentation required (`DF` Do not Fragment option).

Table C shows a sample `tcpdump` ICMP output. ICMP is the protocol used for error control and message handling. There are many different types of ICMP records that have different messages. In the first ICMP record following the timestamp, we can see the source pine.tree.com. Following the greater than sign, we see the destination oak.tree.com. Because ICMP doesn't use ports to communicate like TCP and UDP, we won't see this information. The ICMP message type that

follows the destination host in the first record is an ICMP echo request or ping. In the second record, ICMP message type is an ICMP message reply.

Table C: *tcpdump generates the following record for the ICMP protocol.*

Timestamp	Source	>	Destination:	icmp:	icmp Message
11:17:50.011072	pine.tree.com	>	oak.tree.com:	icmp:	echo request (DF)
11:17:50.011091	oak.tree.com	>	pine.tree.com:	icmp:	echo reply (DF)

The last row of `tcpdump` output is the ARP request. This shows the IP address of 192.168.10.55 as the target IP address and the 192.168.10.1 IP address as the sender IP address.

There are also command-line options for `tcpdump` that will alter the default behavior, either by collecting specified records, printing in a more verbose mode, printing in hexadecimal, or writing records as raw packets to a file instead of printing as standard output. For all of `tcpdump`'s options, consult the `tcpdump(1)` man pages.

Filters for tcpdump

As we mentioned earlier, `tcpdump` allows you to extract particular kinds of network traffic. The basic syntax for a `tcpdump` filter is as follows:

```
<header>[<offset>:<length>] <relation> <value>
```

So, to use this filter to detect `telnet` traffic, you can write the following:

```
#tcpdump tcp and dst port 23
```

The filter is `tcp and dst port 23`. Only those packets that satisfy the filter requirements will be captured. This filter selects those packets that have protocol type TCP and destination port number 23. Since the `telnet` protocol uses port 23, this filter selects `telnet` packets.

`The` filter syntax for `tcpdump` is very robust. You can employ the filters to extract connections involving specific networks, hosts, and ports. The simplest way to do this is to use the macros that `tcpdump` supplies for `<header>`, such as `src`, `dst`, `host`, `net`, and `port`. However, if you wish to specify a range of networks, hosts, or ports, you need to use relational operators in conjunction with the individual header byte specifications. As an example, you can specify all destination IP addresses belonging to the range 172.16.0.0 172.31.255.255 by writin g the following:

```
dst net 172  and (ip[17] > 15) and (ip[17] < 32)
```

This specifies the first octet and the range for the second octet of the destination IP address. You could alternatively specify the first octet of the destination address by writing `ip[16] = 172`, but `dst net 172` seems a bit more intuitive. Similarly, you could specify the IP address range by writing:

```
(dst net 172.16 or dst net 172.17 or dst net
172.18 or ... or dst net 172.31)
```

However, this can quickly get cumbersome when a large number of networks are involved. `Tcpdump` accepts the regular relational operators for `<relation>`: `= < > <= >= !=` and the logical operators: `and`, `or`, and `not`. Further, `tcpdump` accepts either hostnames or IP addresses, and either port numbers or service names. Note that when you use service names, and `tcpdump` understands the protocol, both the port number and the protocol are checked, instead of just checking the port number.

Let's create several filters. To capture all traffic between pine.tree.com and oak.tree.com machines for a `telnet` port, you can write:

```
#tcpdump host pine.tree.com and host oak.tree.com
and port telnet
```

Or just to detect telnet traffic with the following:

```
#tcpdump "tcp[2:2] = 23"
```

This works because the destination port number is two bytes, beginning at the second byte in the TCP header. `Telnet` uses `23/tcp` for the server port. We use single quotes to protect the square brackets from being interpreted by the UNIX shell.

Following are some more examples:

- ICMP filter. This filter looks for all ICMP packets that are not ping packets:

```
#tcpdump "icmp and icmp[0] !=8 and icmp[0] != 0"
```

- r-utility filter. `rlogin`, `rcp, rsh, rdist` and so forth. It is wise to look for packets with r-utilities from unknown sites:

```
#tcpdump "ip and (tcp dst port 512 or
tcp dst port 513 or dst port 514)"
```

- X11 filter. This filter finds any X11 or Motif traffic:

```
#tcpdump "tcp and (port 6000 or
port 6001 or port 6002)"
```

- Inbound SYN Filter. This filter monitors any inbound SYN connections to ports you are not expecting traffic on:

```
#tcpdump "tcp and (tcp[13] & 0x02 != 0) and
(tcp[13] & 0x10 = 0)and (not dst port 53) and
 (not dst port 80)
and (not dst port 25) and (not dst port 21)"
```

As you can see from these examples, `tcpdump`'s filters also allow the use of parentheses and logical operators. But what if you decide to implement a more complex filter? We've borrowed an example for **Listing B** from Intrusion Detection: SHADOW Style, which is referenced at the end of this article. This is a bad-events filter, which will detect any packets that indicate suspicious activity warranting further attention.

Listing B: *A bad-events filter that can detect potential network threats*

```
#tcpdump "(tcp and (tcp[13] & 3 != 0) and ((dst port 143)
or (dst port 111) or(tcp[13] & 3 != 0 and tcp[13] & 0x10 = 0 and
dst net 172.16 and dst port 1080) or
(dst port 512 or dst port 513 or dst port 514) or
((ip[19] = 0xff) and not (net 172.16/16 or net 192.168/16))
or (ip[12:4] = ip[16:4]))) or (not tcp and
igrp and not dst port 520 and
((dst port 111) or (udp port 2049) or ((ip[19] = 0xff)
and not (net 172.16/16 or net 192.168/16))
or (ip[12:4] = ip[16:4])))"
```

Obviously, it isn't reasonable to type such complicated filters on the command line. You can save such filters in a file and then read in by `tcpdump` at runtime with the `-F` flag:

```
#tcpdump -F filterfile
```

To understand what the last filter does, you have to master `tcpdump` and network protocols. Read the man pages for `tcpdump(1)`. There are also a couple of excellent books about network protocols for further reading, which you'll find listed at the end of this article.

What to do with tcpdump?

First of all, `tcpdump` is a great utility for studying network protocols. For instance, **Listing C** shows how you can use `tcpdump` to see and analyze the setup phase of a TCP connection, which is generally referred to as a three-way handshake.

Listing C: *Use the output from tcpdump to study how network protocols work*

```
#tcpdump host pine.tree.com and host oak.tree.com and port telnet
tcpdump: listening on hme0
08:16:48.519576 pine.tree.com.33102 > oak.tree.com.telnet: S 2970775184
:2
970775184(0) win 8760 <mss 1460> (DF)
08:16:48.519602 oak.tree.com.telnet > pine.tree.com.33102: S 3013029286
:3
013029286(0) ack 2970775185 win 8760 <mss 1460> (DF)
08:16:48.520242 pine.tree.com.33102 > oak.tree.com.telnet: . ack 1 win
8760 (DF)
08:16:48.522879 pine.tree.com.33102 > oak.tree.com.telnet: P 1:28(27) a
ck 1 win 8760 (DF)
08:16:48.522913 oak.tree.com.telnet > pine.tree.com.33102: . ack 28 win
 8760 (DF)
08:16:48.540494 oak.tree.com.telnet > pine.tree.com.33102: P 1:16(15) a
ck 28 win 8760 (DF)
^C
```

You'll find that `tcpdump` is an excellent network analyzing, troubleshooting, and debugging tool. You can use `it` for troubleshooting and analyzing TCP, DNS, NFS, PPP, RIP, AppleTalk, and other network protocols. For example, to troubleshoot network problems, dump all activity occurring on an interface:

```
#tcpdump -i hme0 -vv
```

For RIP troubleshooting, you can use the following filter:

```
#tcpdump -i hme0 -s 1024 port routed
```

`You can also use tcpdump` to create a powerful, network-based Intrusion Detection System (IDS). SHADOW, which was designed by the Naval Surface Warfare Center, Dahlgren Division, is one of the first freely available toolkits based on `tcpdump` for detecting intruders. Intrusion Detection: SHADOW Style will guide you step by step in building your own very robust network-based IDS. You will also find many useful `tcpdump` filters in this book.

Obviously, you can run `tcpdump` on your hosts to detect network-based attacks, even without building your own IDS. For example, to detect teardrop attacks, you may use the following filter:

```
#tcpdump "udp and (ip[6:1] & 0x20 != 0)"
```

To detect IP packets where the source and destination addresses are equal (classic land attack), you can use:

```
#tcpdump "ip[12:4] = ip[16:4]"
```

If you would like to understand more about packet filtering based on `tcpdump` and network attacks, the SANS Institute (http://www.sansstore.org/) offers excellent courses on this topic.

Conclusion

It's always a good idea to sniff your own networks for any suspicious behavior. Based on our experience, `tcpdump` is the right tool for the job. Working with `tcpdump` will also allow you to understand in detail the various network protocols.

If you periodically sniff your networks, you'll be able to find the network bottleneck. You'll also be able to identify bad-guy network attacks by creating your own tcpdump-based IDS.

Further reading

- Intrusion Detection: SHADOW Style. Stephen Northcutt and the Intrusion Detection Team. SANS Institute (http://www.sansstore.org/).
- TCP/IP Illustrated, Volume I. Richard Stevens.
- Internetworking with TCP/IP, Volume I. Douglas E. Comer.