

# Comprehensive Guide to tcpdump (Part 1)

March 19, 2020 By Raj Chandel

In this article, we are going to learn about tcpdump. It is a powerful command-line tool for network packet analysis. Tcpdump helps us troubleshoot the network issues as well as help us analyze the working of some security tools.

## Table of Content

- **Introduction**
- **Available Options**
- **List of interfaces**
- **Default working**
- **Capturing traffic of a particular interface**
- **Packet count**
- **Verbose mode**
- **Printing each packet in ASCII**
- **Don't convert address**
- **Port filter**
- **Host filter**
- **The header of each packet**
- **TCP sequence number**
- **Packet filter**
- **Packet Direction**
- **Live number count**
- **Read and Write in a file**
- **Snapshot length**
- **Dump mode**

## Introduction

Tcpdump was originally developed in 1988 by Van Jacobson, Sally Floyd, Vern Paxson, and Steven McCanne. They worked at the Lawrence Berkeley Laboratory Network Research Group.

It allows its users to display the TCP/IP and other packets being received and transmitted over the network. It works on most of the Linux based operating systems. It uses the libpcap library to capture packets, which is a C/C++ based library. Tcpdump has a windows equivalent as well. It is named windump. It uses a winpcap for its library.

## Available Options

We can use the following parameter to print the tcpdump and libpcap version strings. Also, we can print a usage message that shows all the available options.

```
tcpdump -h
tcpdump --help
```

```
root@kali:~# tcpdump -h ↩
tcpdump version 4.9.3
libpcap version 1.9.1 (with TPACKET_V3)
OpenSSL 1.1.1d 10 Sep 2019
Usage: tcpdump [-aAbdDefhHIJKlLnNOpqStuUvxX#] [-B size] [-c count]
               [-C file_size] [-E algo:secret] [-F file] [-G seconds]
               [-i interface] [-j tstamptype] [-M secret] [--number]
               [-Q in|out|inout]
               [-r file] [-s snaplen] [--time-stamp-precision precision]
               [--immediate-mode] [-T type] [--version] [-V file]
               [-w file] [-W filecount] [-y datalinktype] [-z postrotate-command]
               [-Z user] [expression]
```

## List of interfaces

An interface is the point of interconnection between a computer and a network. We can use the following parameter to print the list of the network interfaces available on the system. It can also detect interfaces on which tcpdump can capture packets. For each network interface, a number is assigned. This number can be used with the ‘-i’ parameter to capture packets on that particular interface.

There might be a scenario where the machine that we are working on, is unable to list the network interfaces it is running. This can be a compatibility issue or something else hindering the execution of some specific commands (ifconfig -a).

```
tcpdump -list-interface
tcpdump -D
```

```
root@kali:~# tcpdump -list-interface ↩
1.eth0 [Up, Running]
2.lo [Up, Running, Loopback]
3.any (Pseudo-device that captures on all interfaces) [Up, Running]
4.nflog (Linux netfilter log (NFLOG) interface) [none]
5.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
root@kali:~# tcpdump -D
1.eth0 [Up, Running]
2.lo [Up, Running, Loopback]
3.any (Pseudo-device that captures on all interfaces) [Up, Running]
4.nflog (Linux netfilter log (NFLOG) interface) [none]
5.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
```

## Default Capture

Before moving onto to advanced options and parameters of this network traffic capture tool let's first do a capture with the default configurations.

tcpdump

```
root@kali:~# tcpdump ↵
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:29:50.466425 IP 192.168.0.6.netbios-ns > 192.168.0.255.netbios-ns: UDP, length 50
14:29:50.490349 IP 192.168.0.5.36540 > dns.google.domain: 57499+ PTR? 255.0.168.192.in-addr.arpa.
14:29:50.495738 IP dns.google.domain > 192.168.0.5.36540: 57499 NXDomain* 0/1/0 (114)
14:29:50.495916 IP 192.168.0.5.55281 > dns.google.domain: 27366+ PTR? 6.0.168.192.in-addr.arpa. (
14:29:50.501781 IP dns.google.domain > 192.168.0.5.55281: 27366 NXDomain* 0/1/0 (112)
14:29:50.502013 IP 192.168.0.5.52479 > dns.google.domain: 63048+ PTR? 8.8.8.8.in-addr.arpa. (38)
14:29:50.506966 IP dns.google.domain > 192.168.0.5.52479: 63048 1/0/0 PTR dns.google. (62)
14:29:50.507124 IP 192.168.0.5.42053 > dns.google.domain: 9777+ PTR? 5.0.168.192.in-addr.arpa. (4
14:29:50.511720 IP dns.google.domain > 192.168.0.5.42053: 9777 NXDomain* 0/1/0 (112)
14:29:50.743091 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 17, length 64
14:29:50.743247 IP 192.168.0.5.52686 > dns.google.domain: 45555+ PTR? 89.6.28.104.in-addr.arpa. (
14:29:50.751412 IP dns.google.domain > 192.168.0.5.52686: 45555 NXDomain 0/1/0 (137)
14:29:50.900413 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 17, length 64
14:29:51.747664 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 18, length 64
^C
14 packets captured
14 packets received by filter
0 packets dropped by kernel
```

## Capturing traffic of a particular interface

We will be capturing traffic using the ethernet network which is known as “eth0”. This type of interface is usually connected to the network by a category 5 cable.

To select this interface we need to use -i parameter.

tcpdump -i eth0

```
root@kali:~# tcpdump -i eth0 ↵
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:30:44.027968 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 70, length 64
14:30:44.028414 IP 192.168.0.5.50801 > dns.google.domain: 52063+ PTR? 5.0.168.192.in-addr.a
14:30:44.033591 IP dns.google.domain > 192.168.0.5.50801: 52063 NXDomain* 0/1/0 (112)
14:30:44.033863 IP 192.168.0.5.53698 > dns.google.domain: 50096+ PTR? 89.6.28.104.in-addr.a
14:30:44.038983 IP dns.google.domain > 192.168.0.5.53698: 50096 NXDomain 0/1/0 (137)
14:30:44.039114 IP 192.168.0.5.50326 > dns.google.domain: 5049+ PTR? 8.8.8.8.in-addr.arpa.
14:30:44.043187 IP dns.google.domain > 192.168.0.5.50326: 5049 1/0/0 PTR dns.google. (62)
14:30:44.874204 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 71, length 6
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

## Packet count

Tcpdump has some amazing features which we can use to make our traffic analysis more efficient. We can access some of these features using various parameters. We use the -c parameter, it will help us to capture the exact amount of data that we need and display those. It refines the amount of data we captured.

```
tcpdump -i eth0 -c10
```

```
root@kali:~# tcpdump -i eth0 -c10
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:31:39.015544 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 125, length 64
14:31:39.016097 IP 192.168.0.5.38399 > dns.google.domain: 30357+ PTR? 89.6.28.104.in-addr.arpa. (42)
14:31:39.020973 IP dns.google.domain > 192.168.0.5.38399: 30357 NXDomain 0/1/0 (137)
14:31:39.021153 IP 192.168.0.5.42657 > dns.google.domain: 39318+ PTR? 5.0.168.192.in-addr.arpa. (42)
14:31:39.025541 IP dns.google.domain > 192.168.0.5.42657: 39318 NXDomain* 0/1/0 (112)
14:31:39.025780 IP 192.168.0.5.59288 > dns.google.domain: 37268+ PTR? 8.8.8.8.in-addr.arpa. (42)
14:31:39.030209 IP dns.google.domain > 192.168.0.5.59288: 37268 1/0/0 PTR dns.google. (62)
14:31:39.171652 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 125, length 64
14:31:40.018238 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 126, length 64
14:31:40.172954 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 126, length 64
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

## Verbose mode

The verbose mode provides information regarding the traffic scan. For example, time to live(TTL), identification of data, total length and available options in IP packets. It enables additional packet integrity checks such as verifying the IP and ICMP headers.

To get extra information from our scan we need to use -v parameter.

```
tcpdump -i eth0 -c 5 -v
```

```
root@kali:~# tcpdump -i eth0 -c 5 -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:32:29.460679 IP (tos 0x0, ttl 55, id 43396, offset 0, flags [none], proto ICMP (1), length 84)
104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 175, length 64
14:32:29.461119 IP (tos 0x0, ttl 64, id 26941, offset 0, flags [DF], proto UDP (17), length 70)
192.168.0.5.59232 > dns.google.domain: 13145+ PTR? 5.0.168.192.in-addr.arpa. (42)
14:32:29.466903 IP (tos 0x0, ttl 63, id 2765, offset 0, flags [none], proto UDP (17), length 140)
dns.google.domain > 192.168.0.5.59232: 13145 NXDomain* 0/1/0 (112)
14:32:29.467083 IP (tos 0x0, ttl 64, id 26942, offset 0, flags [DF], proto UDP (17), length 70)
192.168.0.5.51101 > dns.google.domain: 61260+ PTR? 89.6.28.104.in-addr.arpa. (42)
14:32:29.472455 IP (tos 0x0, ttl 63, id 2766, offset 0, flags [none], proto UDP (17), length 165)
dns.google.domain > 192.168.0.5.51101: 61260 NXDomain 0/1/0 (137)
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

## Printing each packet in ASCII



ASCII is the abbreviation of the American Standard Code for Information Interchange. It is a character encoding standard for electronic communication. ASCII codes represent the text in computers and other devices. Most of the modern character encoding techniques were based on the ASCII codes. To print each packet in ASCII code we need to use -A parameter.

```
tcpdump -i eth0 -c 5 -A
```

```
root@kali:~# tcpdump -i eth0 -c 5 -A
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:33:16.383936 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 222, length 64
E..T..@.@.....h..Y.....I^.....! "$%&'()*+,-./01234567
14:33:16.384388 IP 192.168.0.5.41921 > dns.google.domain: 16909+ PTR? 89.6.28.104.in-addr.arpa.
E..F.8@.@.....5.2..B.....89.6.28.104.in-addr.arpa.....
14:33:16.411382 IP dns.google.domain > 192.168.0.5.41921: 16909 NXDomain 0/1/0 (137)
E....X..?..3.....5.....1.B.....89.6.28.104.in-addr.arpa.....28.104.in-addr.arpa.....
cloudflare.com..dns
cloudflare.com.y)k... '...`.:.....
14:33:16.411514 IP 192.168.0.5.36795 > dns.google.domain: 47327+ PTR? 5.0.168.192.in-addr.arpa.
E..F.:@.@.....5.2.....5.0.168.192.in-addr.arpa.....
14:33:16.416286 IP dns.google.domain > 192.168.0.5.36795: 47327 NXDomain* 0/1/0 (112)
E....Y..?..K.....5...x.....5.0.168.192.in-addr.arpa.....168.192.in-addr.arpa....
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

## Don't convert address

With the help of the tcpdump -nn parameter, we can see the actual background address without any filters. This feature helps us to understand the data traffic better without any filters.

```
tcpdump -i eth0 -c 5
tcpdump -i eth0 -c 5 -nn
```

```

root@kali:~# tcpdump -i eth0 -c 5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:34:05.572309 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 271, length 64
14:34:05.572854 IP 192.168.0.5.41236 > dns.google.domain: 21985+ PTR? 89.6.28.104.in-addr.arpa
14:34:05.583880 IP dns.google.domain > 192.168.0.5.41236: 21985 NXDomain 0/1/0 (137)
14:34:05.584066 IP 192.168.0.5.58639 > dns.google.domain: 20259+ PTR? 5.0.168.192.in-addr.arpa
14:34:05.589816 IP dns.google.domain > 192.168.0.5.58639: 20259 NXDomain* 0/1/0 (112)
5 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 5 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:34:13.594032 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 279, length 64
14:34:13.748095 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 279, length 64
14:34:14.598811 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 280, length 64
14:34:14.753531 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 280, length 64
14:34:15.601282 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 281, length 64
5 packets captured
5 packets received by filter
0 packets dropped by kernel

```

## Port filter

Port filter helps us to analyze the data traffic of a particular port. It helps us to monitor the destination ports of the TCP/UDP or other port-based network protocols.

```
tcpdump -i eth0 -c 5 -v port 80
```

```

root@kali:~# tcpdump -i eth0 -c 5 -v port 80
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:38:01.583365 IP (tos 0x0, ttl 64, id 39362, offset 0, flags [DF], proto TCP (6), length 52)
192.168.0.5.53668 > rs202995.rs.hosteurope.de.http: Flags [.], cksum 0xa395 (incorrect → 0x7
178), ack 303151253, win 501, options [nop,nop,TS val 468436003 ecr 495166143], length 0
14:38:01.585604 IP (tos 0x0, ttl 61, id 6081, offset 0, flags [DF], proto TCP (6), length 52)
rs202995.rs.hosteurope.de.http > 192.168.0.5.53668: Flags [.], cksum 0x771c (correct), ack 1,
win 65, options [nop,nop,TS val 495167167 ecr 468433970], length 0
14:38:04.511843 IP (tos 0x0, ttl 60, id 6082, offset 0, flags [DF], proto TCP (6), length 52)
rs202995.rs.hosteurope.de.http > 192.168.0.5.53668: Flags [F.], cksum 0x75f6 (correct), seq 1
, ack 1, win 65, options [nop,nop,TS val 495167460 ecr 468433970], length 0
14:38:04.512010 IP (tos 0x0, ttl 64, id 39363, offset 0, flags [DF], proto TCP (6), length 52)
192.168.0.5.53668 > rs202995.rs.hosteurope.de.http: Flags [F.], cksum 0xa395 (incorrect → 0x
60df), seq 1, ack 2, win 501, options [nop,nop,TS val 468438932 ecr 495167460], length 0
14:38:04.514669 IP (tos 0x0, ttl 61, id 6083, offset 0, flags [DF], proto TCP (6), length 52)
rs202995.rs.hosteurope.de.http > 192.168.0.5.53668: Flags [.], cksum 0x6293 (correct), ack 2,
win 65, options [nop,nop,TS val 495167460 ecr 468438932], length 0
5 packets captured
5 packets received by filter
0 packets dropped by kernel

```

## Host filter

This filter helps us to analyze the data traffic of a particular host. It also allows us to stick to a particular host through which further makes our analyzing better. Multiple parameters can also be applied, such as -v, -c, -A, -n, to get extra information about that host.

```
tcpdump host 104.28.6.89 -c10 -A -n
```

```
root@kali:~# tcpdump host 104.28.6.89 -c10 -A -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:42:40.190512 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 784, length 64
E..T..@.@.tv....h..Y.....0.I^.....!#$%&'()*+,-./01234567
14:42:40.345026 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 784, length 64
E..TI...7..
.h..Y.....0.I^.....!#$%&'()*+,-./01234567
14:42:41.193602 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 785, length 64
E..T.L@.@.t:....h..Y.....1.I^.....!#$%&'()*+,-./01234567
14:42:41.348372 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 785, length 64
E..T....7.o.h..Y.....1.I^.....!#$%&'()*+,-./01234567
14:42:42.199701 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 786, length 64
E..T..@.@.s....h..Y...n....2.I^.....!#$%&'()*+,-./01234567
14:42:42.353676 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 786, length 64
E..T.f..7..h..Y.....0....2.I^.....!#$%&'()*+,-./01234567
14:42:43.205210 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 787, length 64
E..T.'@.@.s_....h..Y...X....3.I^.....!#$%&'()*+,-./01234567
14:42:43.359437 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 787, length 64
E..T:,..7..[h..Y.....X....3.I^.....!#$%&'()*+,-./01234567
14:42:44.211463 IP 192.168.0.5 > 104.28.6.89: ICMP echo request, id 46758, seq 788, length 64
E..T..@.@.r....h..Y..
?....4.I^.....9.....!#$%&'()*+,-./01234567
14:42:44.365528 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 788, length 64
E..TT...7...h..Y.....?....4.I^.....9.....!#$%&'()*+,-./01234567
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

## The header of each packet

The header contains all the instructions given to the individual packet about the data carried by them. These instructions can be packet length, advertisement, synchronization, ASCII code, hex values, etc. We can use -X parameter to see this information on our data packets.

```
tcpdump -i eth0 -c 3 -X
```



```

root@kali:~# tcpdump -i eth0 -c 3 -X
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:16:50.110673 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 59
    0x0000: 4500 0054 1344 4000 4001 f840 c0a8 0007  E..T.D@.@..@....
    0x0010: 681c 0659 0800 27b0 0869 003b 82ca 4a5e  h..Y..'.i.;..J^
    0x0020: 0000 0000 3ab0 0100 0000 0000 1011 1213  ....:.....
    0x0030: 1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .....!"#
    0x0040: 2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
    0x0050: 3435 3637                                4567
12:16:50.111314 IP 192.168.0.7.52875 > dns.google.domain: 44283+ PTR? 89.6.28.104
    0x0000: 4500 0046 b3cf 4000 4011 b618 c0a8 0007  E..F..@.@.....
    0x0010: 0808 0808 ce8b 0035 0032 d102 acfb 0100  .....5.2.....
    0x0020: 0001 0000 0000 0000 0238 3901 3602 3238  .....89.6.28
    0x0030: 0331 3034 0769 6e2d 6164 6472 0461 7270  .104.in-addr.arp
    0x0040: 6100 000c 0001                                a.....
12:16:50.116158 IP dns.google.domain > 192.168.0.7.52875: 44283 NXDomain 0/1/0 (
    0x0000: 4500 00a5 6595 0000 3f11 44f4 0808 0808  E...e...?.D.....
    0x0010: c0a8 0007 0035 ce8b 0091 f81c acfb 8183  .....5.....
    0x0020: 0001 0000 0001 0000 0238 3901 3602 3238  .....89.6.28
    0x0030: 0331 3034 0769 6e2d 6164 6472 0461 7270  .104.in-addr.arp
    0x0040: 6100 000c 0001 0232 3803 3130 3407 696e  a.....28.104.in
    0x0050: 2d61 6464 7204 6172 7061 0000 0600 0100  -addr.arpa.....
    0x0060: 000d d700 4004 6372 757a 026e 730a 636c  ....@.cruz.ns.cl
    0x0070: 6f75 6466 6c61 7265 0363 6f6d 0003 646e  oudflare.com..dn
    0x0080: 730a 636c 6f75 6466 6c61 7265 0363 6f6d  s.cloudflare.com
    0x0090: 0079 296b e700 0027 1000 0009 6000 093a  .y)k...'.....`..:
    0x00a0: 8000 000e 10                                .....
3 packets captured
7 packets received by filter
0 packets dropped by kernel

```

## TCP sequence number

All bytes in TCP connections has there sequence number which is a randomly chosen initial sequence number (ISN). SYN packets have one sequence number, so data will begin at ISN+1. The sequence number is the byte number of data in the TCP packet that is sent forward. -S parameter is used to see these data segments of captured packets.

```
tcpdump -i eth0 -nnXS
```



```

root@kali:~# tcpdump -i eth0 -nnXS
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:24:12.970929 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq
0x0000: 4500 0054 ea26 4000 4001 215e c0a8 0007 E..T.6@.@.!^....
0x0010: 681c 0659 0800 fdd6 0869 01f2 3ccc 4a5e h..Y.....i..<.J^
0x0020: 0000 0000 9bd0 0e00 0000 0000 1011 1213 .....
0x0030: 1415 1617 1819 1a1b 1c1d 1e1f 2021 2223 .....!"#
0x0040: 2425 2627 2829 2a2b 2c2d 2e2f 3031 3233 $%&'()*+,-./0123
0x0050: 3435 3637 4567
12:24:13.125146 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq
0x0000: 4500 0054 414e 0000 3701 1337 681c 0659 E..TAN..7..7h..Y
0x0010: c0a8 0007 0000 05d7 0869 01f2 3ccc 4a5e .....i..<.J^
0x0020: 0000 0000 9bd0 0e00 0000 0000 1011 1213 .....
0x0030: 1415 1617 1819 1a1b 1c1d 1e1f 2021 2223 .....!"#
0x0040: 2425 2627 2829 2a2b 2c2d 2e2f 3031 3233 $%&'()*+,-./0123
0x0050: 3435 3637 4567
12:24:13.231157 IP 192.168.0.6.137 > 192.168.0.255.137: UDP, length 50
0x0000: 4500 004e 782c 0000 8011 401d c0a8 0006 E..Nx,...@.....
0x0010: c0a8 00ff 0089 0089 003a 0548 f64d 0110 .....:..H.M..
0x0020: 0001 0000 0000 0000 2046 4845 5046 4345 .....FHEPFCE
0x0030: 4c45 4846 4345 5046 4646 4143 4143 4143 LEHFCEPFFACACAC
0x0040: 4143 4143 4143 4142 4c00 0020 0001 ACACACABL.....
12:24:13.971229 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq
0x0000: 4500 0054 eabd 4000 4001 20c7 c0a8 0007 E..T..@.@.....
0x0010: 681c 0659 0800 d4d4 0869 01f3 3dcc 4a5e h..Y.....i..=.J^
0x0020: 0000 0000 c3d1 0e00 0000 0000 1011 1213 .....
0x0030: 1415 1617 1819 1a1b 1c1d 1e1f 2021 2223 .....!"#
0x0040: 2425 2627 2829 2a2b 2c2d 2e2f 3031 3233 $%&'()*+,-./0123
0x0050: 3435 3637 4567
12:24:14.125437 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq
0x0000: 4500 0054 99ab 0000 3701 bad9 681c 0659 E..T....7...h..Y
0x0010: c0a8 0007 0000 dcd4 0869 01f3 3dcc 4a5e .....i..=.J^
0x0020: 0000 0000 c3d1 0e00 0000 0000 1011 1213 .....
0x0030: 1415 1617 1819 1a1b 1c1d 1e1f 2021 2223 .....!"#

```

## Packet filter

Another feature that is provided by tcpdump is packet filtering. This helps us to see the packet results on a particular data packet in our scan. If we want to apply this filter in our scan we just need to add the desired packet in our scan.

```
tcpdump -i eth0 icmp -c 10
```

```

root@kali:~# tcpdump -i eth0 icmp -c 10 ↩
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:27:16.083611 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 680,
12:27:16.237620 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 680,
12:27:17.085065 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 681,
12:27:17.239283 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 681,
12:27:18.086692 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 682,
12:27:18.241746 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 682,
12:27:19.091601 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 683,
12:27:19.245784 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 683,
12:27:20.093698 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 684,
12:27:20.247890 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 684,
10 packets captured
10 packets received by filter
0 packets dropped by kernel

```

## Packet directions

To the direction of data flow in our traffic, we can use the following parameter :

```
tcpdump -i eth0 icmp -c 5 -Q in
```

```

root@kali:~# tcpdump -i eth0 icmp -c 5 -Q in ↩
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:52:28.074499 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2182
12:52:29.075941 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2183
12:52:30.078928 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2184
12:52:31.086693 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2185
12:52:32.088288 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2186
5 packets captured
10 packets received by filter
0 packets dropped by kernel

```

To see all the requests which we are sending to the server following (- Q out) parameter can be used:

```
tcpdump -i eth0 icmp -c 5 -Q out
```

```

root@kali:~# tcpdump -i eth0 icmp -c 5 -Q out ↩
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:52:38.948852 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2193
12:52:39.951930 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2194
12:52:40.955149 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2195
12:52:41.958088 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2196
12:52:42.959333 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2197
5 packets captured
9 packets received by filter
0 packets dropped by kernel

```

## Live number count

We can apply live number count feature to see how many packets were scanned or captured during the data traffic scans. `--number` parameter is used to count the number of packets that are being captured in a live scan. We also compared packet count to live number count to see its accuracy.

## Read and write in a file

In `tcpdump`, we can write and read into a `.pcap` extension file. Write (`-w`) allow us to write raw data packets that we have as an output to a standard `.pcap` extension file. Where as read option (`-r`) helps us to read that file. To write output in `.pcap` follow:

```
tcpdump -i eth0 icmp -c 10 -w file.pcap
```

To read this `.pcap` file we follow:

```
tcpdump -r file.pcap
```

```
root@kali:~# tcpdump -i eth0 icmp -c 10 -w file.pcap ↩
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
10 packets captured
10 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -r file.pcap
reading from file file.pcap, link-type EN10MB (Ethernet)
13:03:24.157259 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2836, l
13:03:24.313600 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2836, l
13:03:25.160549 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2837, l
13:03:25.314956 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2837, l
13:03:26.163092 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2838, l
13:03:26.317409 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2838, l
13:03:27.165332 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2839, l
13:03:27.319381 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2839, l
13:03:28.168736 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq 2840, l
13:03:28.323582 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 2840, l
```

## Snapshot length

Snapshot length/`snaplen` is referred to as the bytes of data from each packet. It is by default set on the 262144 bytes. With `tcpdump`, we can adjust this limit to our requirement to better understand it in each snap length. `-s` parameter helps us to do it just apply `-s` parameter along with the length of bytes.

```
tcpdump -i eth0 icmp -s10 -c2
tcpdump -i eth0 icmp -s25 -c2
tcpdump -i eth0 icmp -s40 -c2
tcpdump -i eth0 icmp -s45 -c2
```



```

root@kali:~# tcpdump -i eth0 icmp -s 10 -c2 ↵
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 10 bytes
13:10:13.681893 [ether]
13:10:13.836929 [ether]
2 packets captured
3 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 icmp -s 25 -c2 ↵
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 25 bytes
13:10:23.719663 IP [ip]
13:10:23.873639 IP [ip]
2 packets captured
3 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 icmp -s 40 -c2 ↵
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 40 bytes
13:10:31.741402 IP 192.168.0.7 > 104.28.6.89: [icmp]
13:10:31.895856 IP 104.28.6.89 > 192.168.0.7: [icmp]
2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 icmp -s 45 -c2 ↵
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 45 bytes
13:10:36.754051 IP 192.168.0.7 > 104.28.6.89: ICMP echo request, id 2153, seq
13:10:36.908383 IP 104.28.6.89 > 192.168.0.7: ICMP echo reply, id 2153, seq 32
2 packets captured
2 packets received by filter
0 packets dropped by kernel

```

## Dump mode

Dump mode has multiple parameters like -d, -dd, -ddd. Where -d parameter, dumps the compiled matching code into a readable output, -dd parameter, dumps the code as a C program fragment. -ddd parameter and dumps code as a decimal number with a count. To see these results in our scan we need to follow:

```

tcpdump -i eth0 -c 5 -d
tcpdump -i eth0 -c 5 -dd
tcpdump -i eth0 -c 5 -ddd

```

```

root@kali:~# tcpdump -i eth0 -c 5 -d ↵
(000) ret      #262144
root@kali:~# tcpdump -i eth0 -c 5 -dd ↵
{ 0x6, 0, 0, 0x00040000 },
root@kali:~# tcpdump -i eth0 -c 5 -ddd ↵
1
6 0 0 262144

```

This is our first article in the series of a comprehensive guide to tcpdump. Which is based on some basic commands of tcpdump. Stay tuned for more advance option in this amazing tool.