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

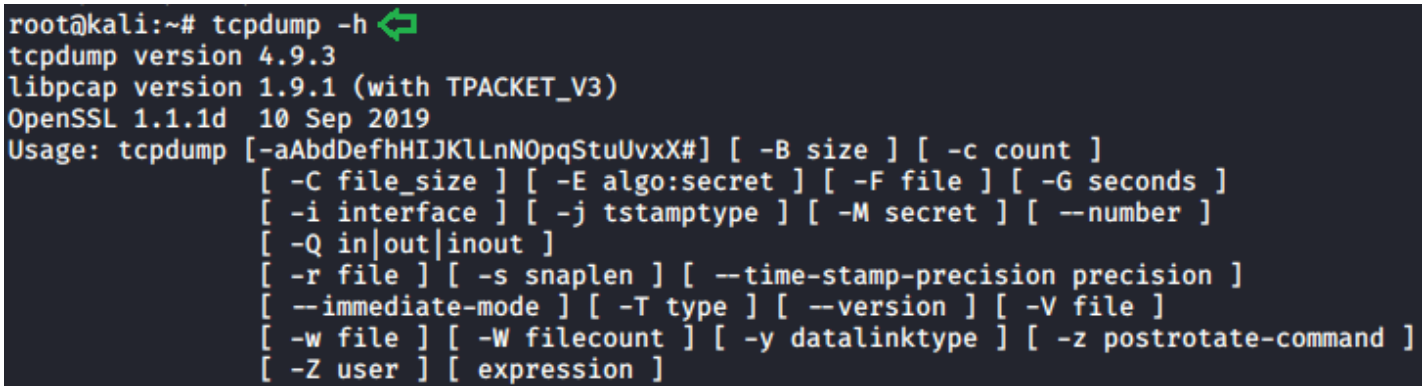
Tcpdump was originally developed in 1988 by Van Jacobson, Sally Floyd, Vern Paxson, and Steven McCanne. They worked at the Lawrence Berkeley Laboratory Network Research Group.

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
```

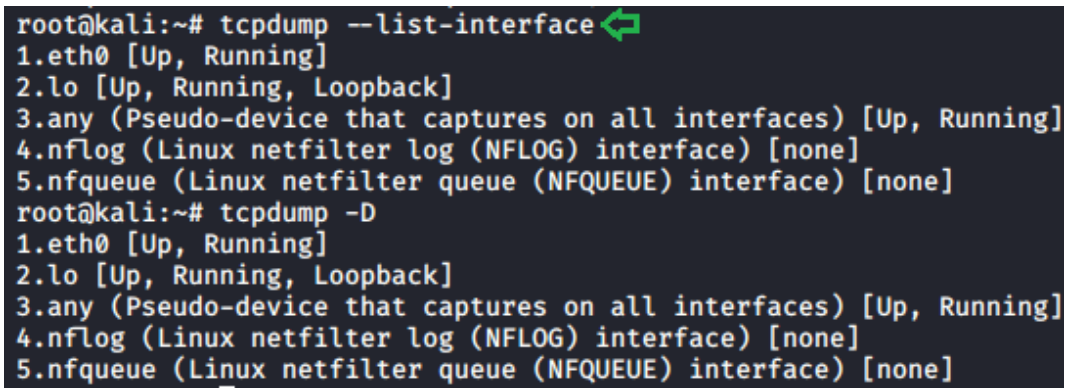


List of interfaces

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).

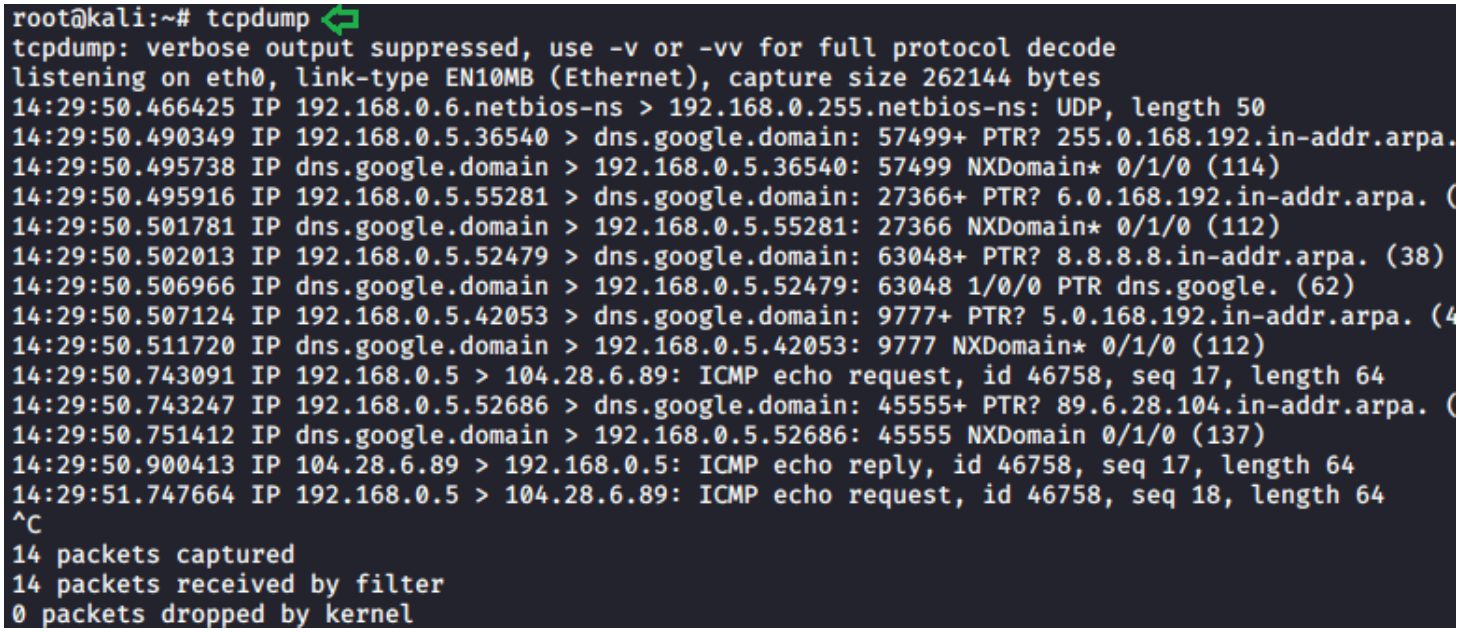
```
tcpdump –list-interface
tcpdump -D
```



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
```

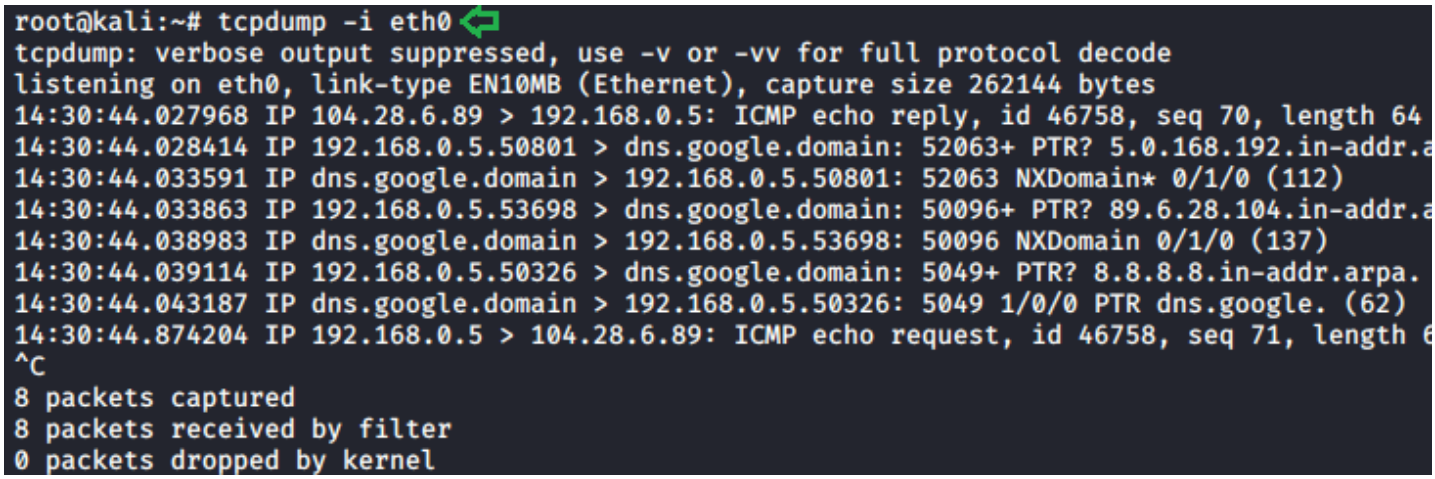


Capturing traffic of a particular interface

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
```



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
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
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. (
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.

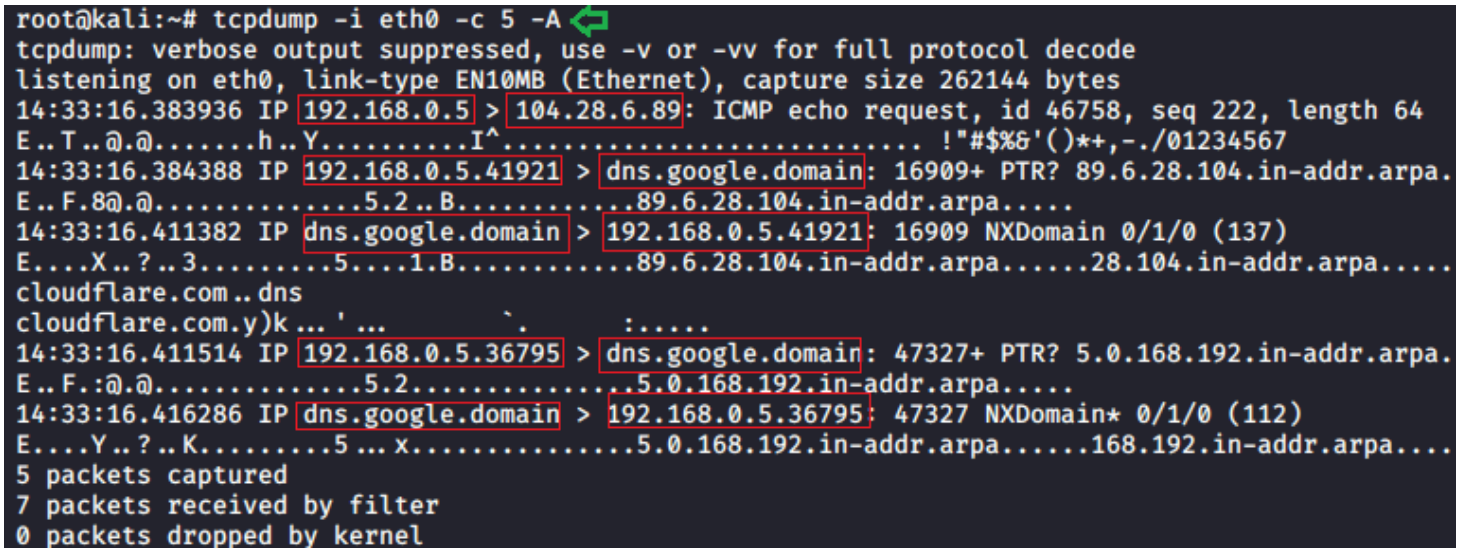
```
tcpdump -i eth0 -c 5 -v
```

```
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
```



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.

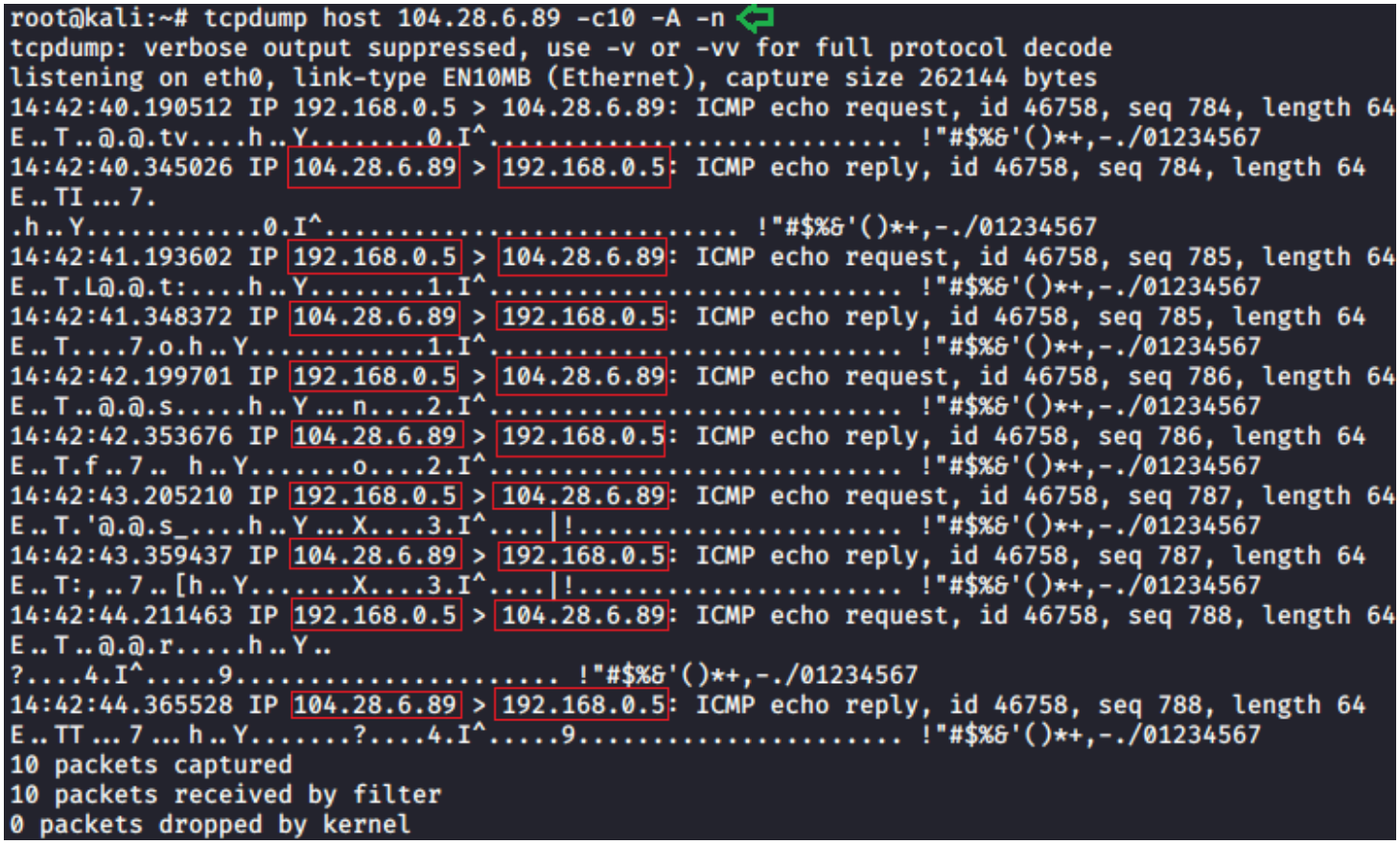
```
tcpdump -i eth0 -c 5 -v port 80
```

```
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.

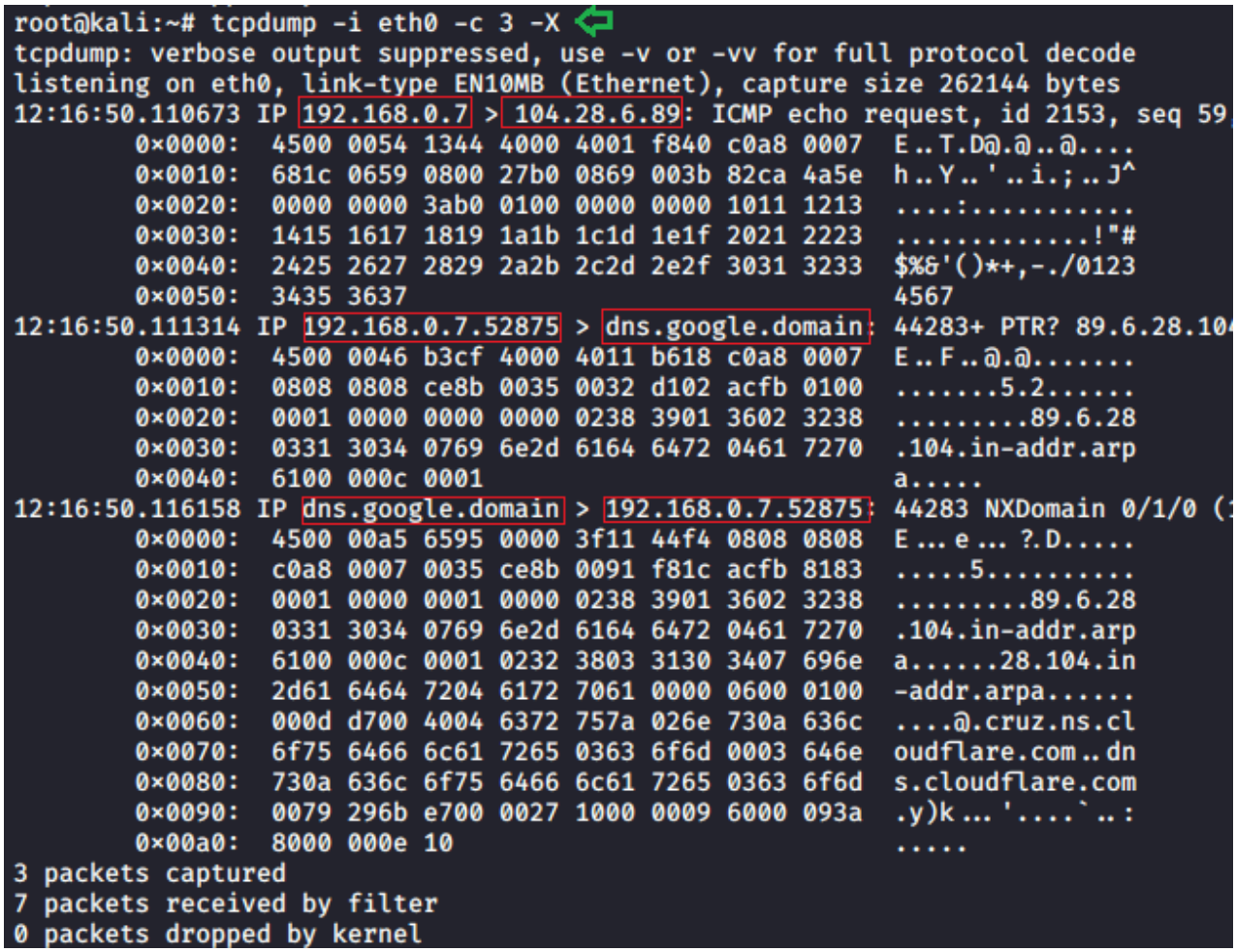
```
tcpdump host 104.28.6.89 -c10 -A -n
```



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.

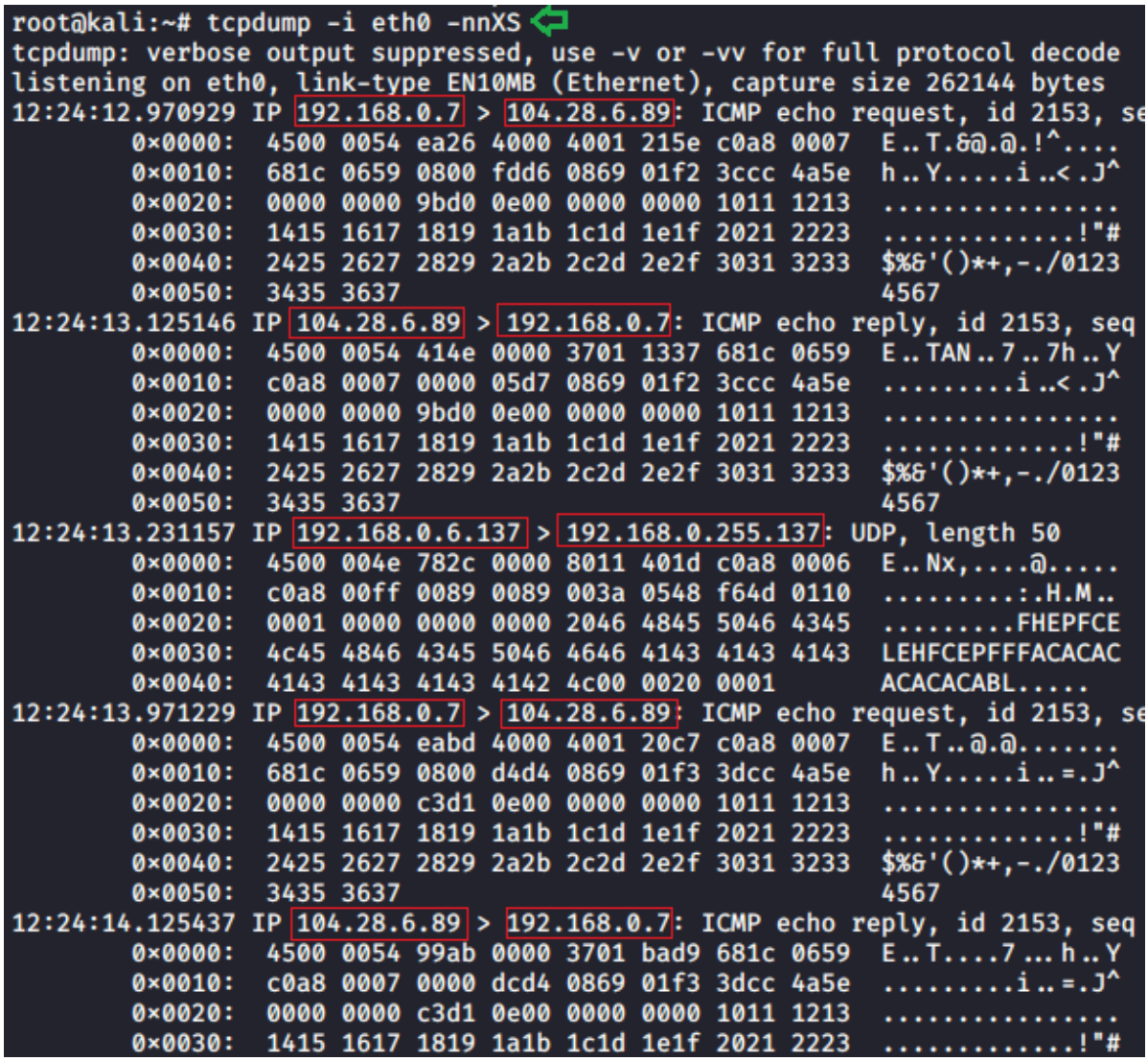
```
tcpdump -i eth0 -c 3 -X
```



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
```



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, l
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, l
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, l
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, l
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, l
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,
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,
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,
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,
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,
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.