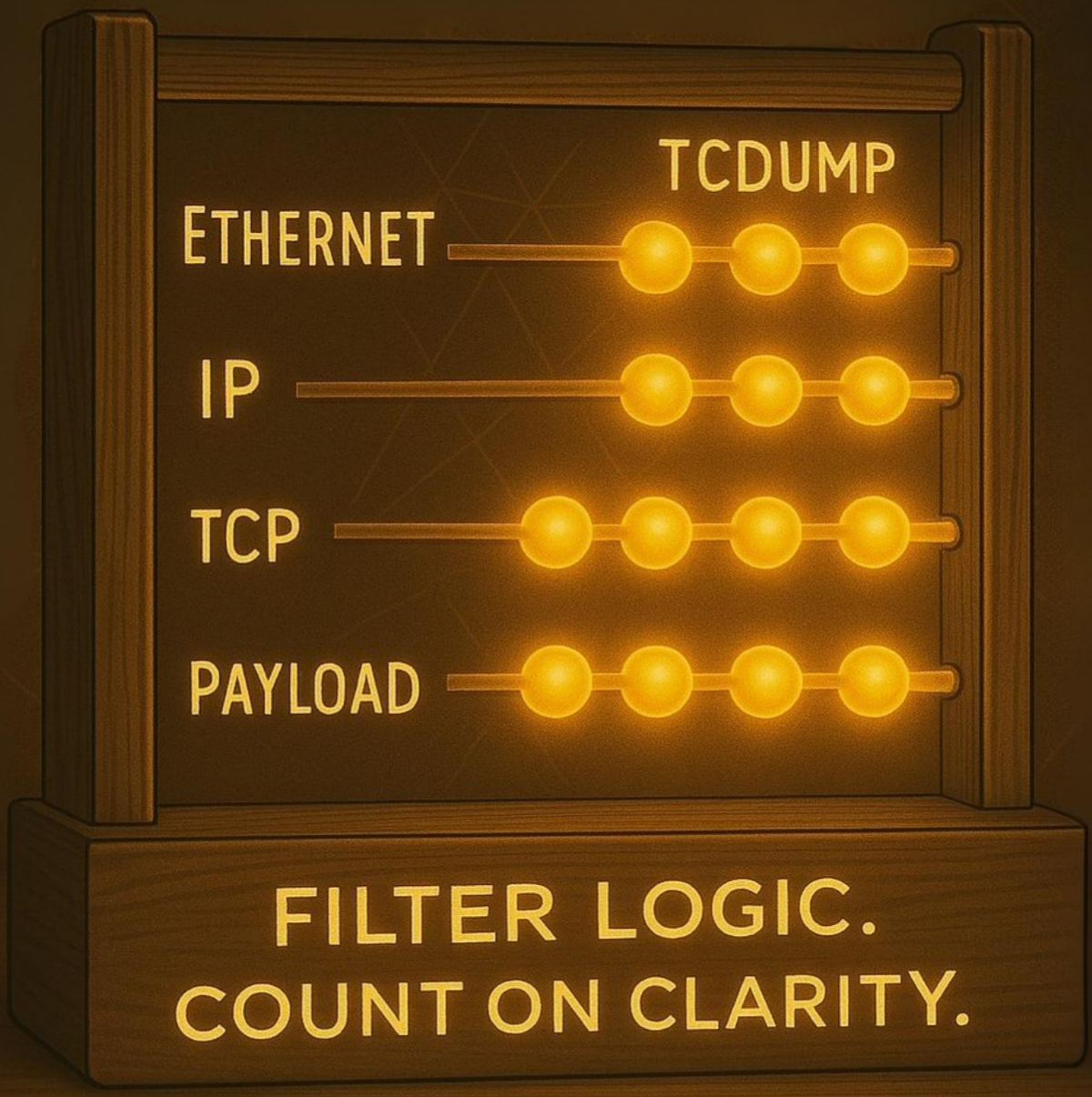
iGNITE Technologies

A DETAILED GUIDE ON :



TCP DUMP

www.hackingarticles.in

Contents

Introduction 4
Introduction to TCPDump 4
Available Options 4
List of interfaces 4
Default Capture 5
Capturing traffic of a particular interface 5
Packet count 6
Verbose mode 6
Printing each packet in ASCII 7
Don't convert address 7
Port filter 8
Host filter 8
The header of each packet 9
TCP sequence number 10
Packet filter 11
Packet directions 12
Live number count 12
Read and write in a file 13
Snapshot length 13
Dump mode 14
Link Level Header 15
Parsing and Printing 15
User scan 17
Timestamp Precision 18
Force Packets 19
RADIUS 20
AODV 20
RPC 20
CNFP 21
LMP 21
PGM 22
RTP 22
RTCP 22
SNMP 23

TFTP 23
VAT 23
WB 24
VXLAN 24
Promiscuous Mode 25
No Promiscuous Mode 26
Version Information 26
Quick Mode 27
Verbose Mode 27
HTTP Requests 28
User Agent 30
Port Range 30
Destination 31
Source 31
Network 31
TCP Packets 32
Tcpdump to Wireshark 33
Conclusion 36

Introduction

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.

Introduction to TCPDump

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. Tcpdump 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 capture traffic using the ethernet network known as “eth0.” A category 5 cable usually connects this type of interface to the network.

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
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
```

```
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.

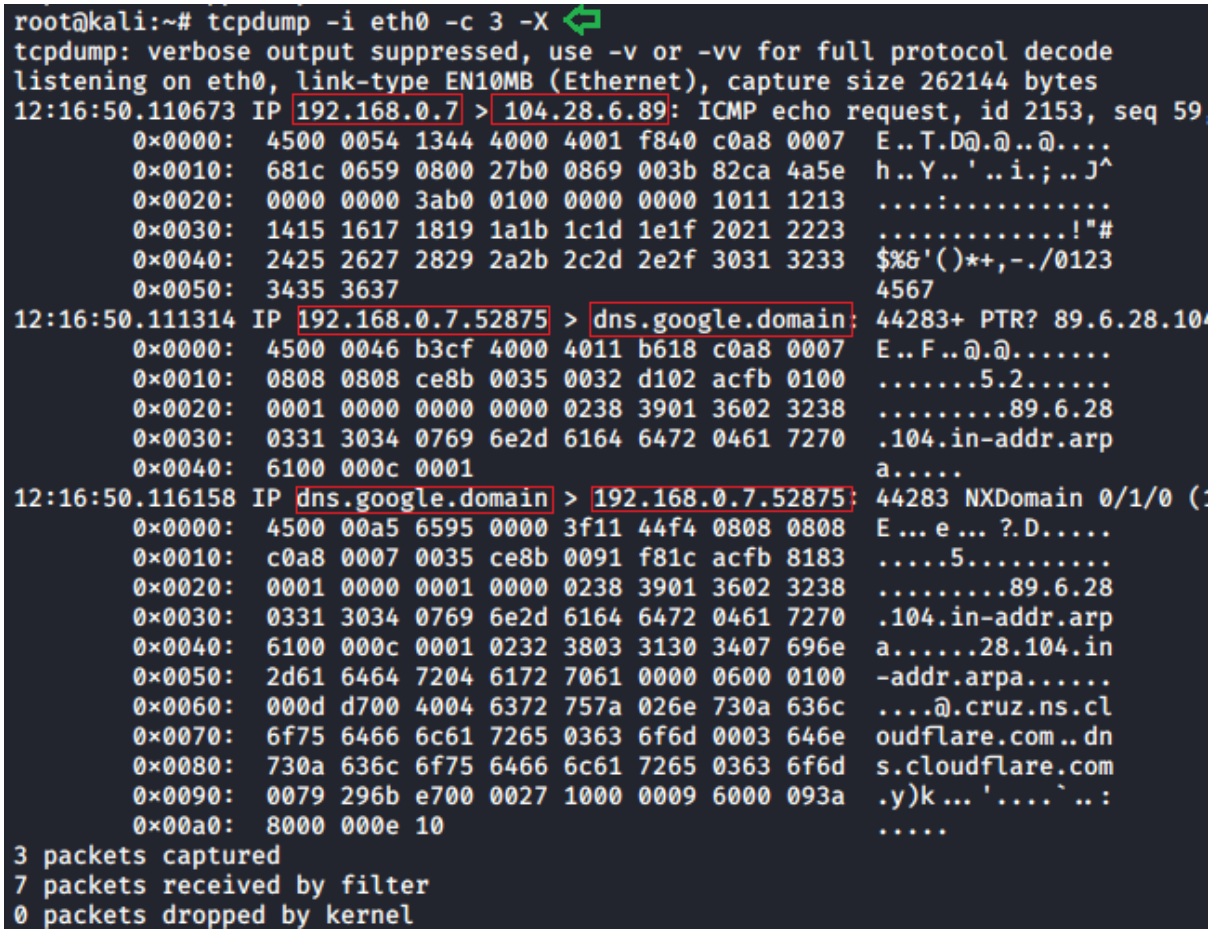
```
tcpdump host 104.28.6.89 -c10 -A -n
```

```
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^.............................. !"#$%&'()*+,-./01234567
14:42:42.353676 IP 104.28.6.89 > 192.168.0.5: ICMP echo reply, id 46758, seq 786, length 64
E..T.f..7..  h..Y.......o....2.I^.............................. !"#$%&'()*+,-./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
```



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, se
        0x0000:  4500 0054 ea26 4000 4001 215e c0a8 0007  E..T.&@.@.!^....
        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  LEHFCEPFFFACACAC
        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, se
        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, 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 the live number count feature to see how many packets the system scanned or captured during the data traffic scans. --The number parameter counts the number of packets that

the system captures 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. Specifically, the write option(-w) allows us to write raw data packets that we receive as output to a standard.pcap file. On the other hand, the read option(-r) helps us read that file. To illustrate, follow the steps below to write the output in .pcap format:

```
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. To begin with, the -d parameter dumps the compiled matching code into a readable output. Furthermore, the -dd parameter dumps the code as a C program fragment. Finally, the -ddd parameter dumps the code as a decimal number with a count. To see these results in our scan, we need to follow the steps below:

```
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
```

Link Level Header

Tcpdump provides us with the option to showcase link-level headers of each data packet. We are using -e parameter to get this information in our data traffic result. Generally, by using this parameter, we will get the MAC address for protocols such as Ethernet and IEEE 802.11.

```
tcpdump -i eth0 -c5
tcpdump -i eth0 -c5 -e
```

```
root@kali:~# tcpdump -i eth0 -c5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:52:23.707232 IP 192.168.0.6.netbios-ns > 192.168.0.255.netbios-ns: UDP, length 50
12:52:23.707866 IP 192.168.0.105.50773 > dns.google.domain: 39020+ PTR? 255.0.168.192.in-addr.arpa
12:52:23.712590 IP dns.google.domain > 192.168.0.105.50773: 39020 NXDomain* 0/1/0 (114)
12:52:23.712836 IP 192.168.0.105.36978 > dns.google.domain: 35334+ PTR? 6.0.168.192.in-addr.arpa.
12:52:23.717595 IP dns.google.domain > 192.168.0.105.36978: 35334 NXDomain* 0/1/0 (112)
5 packets captured
9 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c5 -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:52:29.831534 00:0c:29:f6:d9:c1 (oui Unknown) > 1c:5f:2b:59:e1:24 (oui Unknown), ethertype IPv4
 length 98: 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207, seq 841, length 64
12:52:29.832246 00:0c:29:f6:d9:c1 (oui Unknown) > 1c:5f:2b:59:e1:24 (oui Unknown), ethertype IPv4
 length 84: 192.168.0.105.40819 > dns.google.domain: 16261+ PTR? 89.7.28.104.in-addr.arpa. (42)
12:52:29.837459 1c:5f:2b:59:e1:24 (oui Unknown) > 00:0c:29:f6:d9:c1 (oui Unknown), ethertype IPv4
 length 179: dns.google.domain > 192.168.0.105.40819: 16261 NXDomain 0/1/0 (137)
12:52:29.837744 00:0c:29:f6:d9:c1 (oui Unknown) > 1c:5f:2b:59:e1:24 (oui Unknown), ethertype IPv4
 length 86: 192.168.0.105.58037 > dns.google.domain: 30928+ PTR? 105.0.168.192.in-addr.arpa. (44)
12:52:29.842635 1c:5f:2b:59:e1:24 (oui Unknown) > 00:0c:29:f6:d9:c1 (oui Unknown), ethertype IPv4
 length 156: dns.google.domain > 192.168.0.105.58037: 30928 NXDomain* 0/1/0 (114)
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

Parsing and Printing

As we all know that, the conversation of a concrete syntax to the abstract syntax is known as parsing. The conversation of an abstract syntax to the concrete syntax is called unparsing or printing. Now to parse a data packet we can use -x parameter and to print the abstracted syntax, we can use -xx parameter. In addition to printing the headers of each data packets, we can also print the packet in hex along with its snaplen.

```
tcpdump -i eth0 -c 2 -x
tcpdump -i eth0 -c 2 -xx
```

```
root@kali:~# tcpdump -i eth0 -c 2 -x
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:06:15.132997 IP 192.168.0.6 > 224.0.0.252: igmp v2 report 224.0.0.252
        0x0000:  4600 0020 a3c4 0000 0102 df68 c0a8 0006
        0x0010:  e000 00fc 9404 0000 1600 0903 e000 00fc
        0x0020:  0000 0000 0000 0000 0000 0000 0000
13:06:15.133627 IP 192.168.0.105.50727 > dns.google.domain: 42820+ PTR? 252
        0x0000:  4500 0046 f03e 4000 4011 7947 c0a8 0069
        0x0010:  0808 0808 c627 0035 0032 d164 a744 0100
        0x0020:  0001 0000 0000 0000 0332 3532 0130 0130
        0x0030:  0332 3234 0769 6e2d 6164 6472 0461 7270
        0x0040:  6100 000c 0001
2 packets captured
9 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 2 -xx
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:06:19.370350 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207,
        0x0000:  1c5f 2b59 e124 000c 29f6 d9c1 0800 4500
        0x0010:  0054 0e67 4000 4001 fbbb c0a8 0069 681c
        0x0020:  0759 0800 1363 04b7 067d 1b91 625e 0000
        0x0030:  0000 97a6 0500 0000 0000 1011 1213 1415
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435
        0x0060:  3637
13:06:19.370974 IP 192.168.0.105.35477 > dns.google.domain: 16588+ PTR? 89.
        0x0000:  1c5f 2b59 e124 000c 29f6 d9c1 0800 4500
        0x0010:  0046 f289 4000 4011 76fc c0a8 0069 0808
        0x0020:  0808 8a95 0035 0032 d164 40cc 0100 0001
        0x0030:  0000 0000 0000 0238 3901 3702 3238 0331
        0x0040:  3034 0769 6e2d 6164 6472 0461 7270 6100
        0x0050:  000c 0001
2 packets captured
7 packets received by filter
0 packets dropped by kernel
```

If we want this information provided by -x parameter along with their ASCII code then we need to use -X parameter and if we want the results of -xx parameter along with their ASCII codes then we need to use -XX parameter. To use these parameters in our Data analysis, use the following commands:

```
tcpdump -i eth0 -c 2 -X
tcpdump -i eth0 -c 2 -XX
```

```
root@kali:~# tcpdump -i eth0 -c 2 -X
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:07:52.141821 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207,
        0x0000:  4500 0054 3b1c 4000 4001 cf06 c0a8 0069  E..T;.@.@......i
        0x0010:  681c 0759 0800 6883 04b7 06d9 7891 625e  h..Y..h.....x.b^
        0x0020:  0000 0000 e829 0200 0000 0000 1011 1213  .....).........
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  ............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
13:07:52.142389 IP 192.168.0.105.46241 > dns.google.domain: 3598+ PTR? 89.7
        0x0000:  4500 0046 1e29 4000 4011 4b5d c0a8 0069  E..F.)@.@.K]...i
        0x0010:  0808 0808 b4a1 0035 0032 d164 0e0e 0100  .......5.2.d....
        0x0020:  0001 0000 0000 0000 0238 3901 3702 3238  .........89.7.28
        0x0030:  0331 3034 0769 6e2d 6164 6472 0461 7270  .104.in-addr.arp
        0x0040:  6100 000c 0001                           a.....
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 2 -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:07:56.169468 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207,
        0x0000:  1c5f 2b59 e124 000c 29f6 d9c1 0800 4500  ._+Y.$..)..... E.
        0x0010:  0054 3da4 4000 4001 cc7e c0a8 0069 681c  .T=.@.@..~...ih.
        0x0020:  0759 0800 6713 04b7 06dd 7c91 625e 0000  .Y..g.....|.b^..
        0x0030:  0000 e595 0200 0000 0000 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
13:07:56.170208 IP 192.168.0.105.38029 > dns.google.domain: 9572+ PTR? 89.7
        0x0000:  1c5f 2b59 e124 000c 29f6 d9c1 0800 4500  ._+Y.$..)..... E.
        0x0010:  0046 20c7 4000 4011 48bf c0a8 0069 0808  .F..@.@.H....i..
        0x0020:  0808 948d 0035 0032 d164 2564 0100 0001  .....5.2.d%d....
        0x0030:  0000 0000 0000 0238 3901 3702 3238 0331  .......89.7.28.1
        0x0040:  3034 0769 6e2d 6164 6472 0461 7270 6100  04.in-addr.arpa.
        0x0050:  000c 0001                                ....
2 packets captured
7 packets received by filter
0 packets dropped by kernel
```

User scan

If we are running tcpdump as root then before opening any saved file for analysis, you will observe that it changes the user ID to the user and the group IDs to the primary group of its users.

Tcpdump provides us -Z parameter, through which we can overcome this issue but we need to provide the user name like the following:

```
tcpdump -i eth0 -c 2 -Z root
tcpdump -i eth0 -c 2 -Z kali
```

```
root@kali:~# tcpdump -i eth0 -c 2 -Z root
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:10:43.254041 IP 192.168.0.6.netbios-dgm > 192.168.0.255.netbios-dgm: UDP, length 174
13:10:43.254080 IP 192.168.0.6.netbios-ns > 192.168.0.255.netbios-ns: UDP, length 50
2 packets captured
6 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 2 -Z kali
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:10:52.099175 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207, seq 1931, l
13:10:52.099849 IP 192.168.0.105.56852 > dns.google.domain: 315+ PTR? 89.7.28.104.in-ad
2 packets captured
7 packets received by filter
0 packets dropped by kernel
```

There is one more way to do this, i.e. with the help of **–relinquish-privileges=** parameter.

```
root@kali:~# tcpdump -i eth0 -c 2 --relinquish-privileges=kali
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:16:16.021328 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207, seq 2251
13:16:16.022007 IP 192.168.0.105.42652 > dns.google.domain: 2476+ PTR? 89.7.28.104.i
2 packets captured
30 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 2 --relinquish-privileges=root
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:16:33.267229 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207, seq 2268
13:16:33.267907 IP 192.168.0.105.43904 > dns.google.domain: 9086+ PTR? 89.7.28.104.i
2 packets captured
7 packets received by filter
0 packets dropped by kernel
```

Timestamp Precision

Timestamp is the time registered to a file, log or notification that can record when data is added, removed, modified or transmitted. In tcpdump, there are plenty of parameters that move around timestamp values like -t, -tt, -ttt, -tttt, -ttttt, where each parameter has its unique working and efficiency.

- **-t** parameter which must don't print a timestamp on each dump line.
- **-tt** parameter which can print timestamp till seconds.
- **-ttt** parameter which can print a microsecond or nanosecond resolution depending upon the time stamp precision between the current and previous line on each dump line. Where microsecond is a default resolution.
- **-tttt** parameter which can print a timestamp as hours, minutes, seconds and fractions of seconds since midnight.
- **-ttttt** parameter which is quite similar to the **-ttt** It can able to delta between current and first line on each dump line.

To apply these features in our scan we need to follow these commands:

```
tcpdump -i eth0 -c 2
tcpdump -i eth0 -c 2 -t
tcpdump -i eth0 -c 2 -tt
tcpdump -i eth0 -c 2 -ttt
tcpdump -i eth0 -c 2 -tttt
tcpdump -i eth0 -c 2 -ttttt
```

```
root@kali:~# tcpdump -i eth0 -c2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:20:06.096862 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, seq
11:20:06.097585 IP 192.168.0.105.59427 > dns.google.domain: 40228+ PTR? 89.7.28.
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c2 -t
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, seq 141, length 64
IP 192.168.0.105.48044 > dns.google.domain: 14635+ PTR? 89.7.28.104.in-addr.arpa
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c2 -tt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
1583680818.316734 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, se
1583680818.317569 IP 192.168.0.105.35558 > dns.google.domain: 18656+ PTR? 89.7.2
2 packets captured
11 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c2 -ttt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
 00:00:00.000000 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, seq
 00:00:00.000686 IP 192.168.0.105.45730 > dns.google.domain: 44240+ PTR? 89.7.28
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c2 -tttt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
2020-03-08 11:20:28.486521 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id
2020-03-08 11:20:28.487582 IP 192.168.0.105.51634 > dns.google.domain: 40476+ PT
. (42)
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c2 -ttttt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
 00:00:00.000000 IP 104.28.7.89 > 192.168.0.105: ICMP echo reply, id 1217, seq 1
 00:00:00.001086 IP 192.168.0.105.32974 > dns.google.domain: 10674+ PTR? 105.0.1
2 packets captured
8 packets received by filter
0 packets dropped by kernel
```

Force Packets

In tcpdump, we can force our scan of data traffic to show some particular protocol. When using the force packet feature, defined by selected any “expression” we can interpret specified type. With the help of the -T parameter, we can force data packets to show only the desired protocol results.

The basic syntax of all force packets will remain the same as other parameters -T followed by the desired protocol. The following are some protocols of force packets:

RADIUS

RADIUS stands for Remote Authentication Dial-in User Service. It is a network protocol, which has its unique port number 1812, provides centralized authentication along with authorization and accounting management for its users who connect and use the network services. We can use this protocol for our scan.

```
tcpdump -i eth0 -c5 -T radius
```

```
root@kali:~# tcpdump -i eth0 -c5 -T radius
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:27:30.449857 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, seq 572, leng
11:27:30.450573 IP 192.168.0.105.50924 > dns.google.domain: RADIUS, Unknown Command (29)
11:27:30.454723 IP dns.google.domain > 192.168.0.105.50924: RADIUS, Unknown Command (29)
11:27:30.454899 IP 192.168.0.105.37235 > dns.google.domain: RADIUS, Unknown Command (173
11:27:30.526910 IP 104.28.7.89 > 192.168.0.105: ICMP echo reply, id 1217, seq 572, lengt
5 packets captured
17 packets received by filter
0 packets dropped by kernel
```

AODV

Adhoc On-demand Distance Vector protocol is a routing protocol for mobile ad hoc networks and other wireless networks. It is a routing protocol that is used for a low power and low data rate for wireless networks. To see these results in our scan follow.

```
tcpdump -i eth0 -c5 -T aodv
```

```
root@kali:~# tcpdump -i eth0 -c5 -T aodv
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:28:44.193579 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, seq 645,
11:28:44.194300 IP 192.168.0.105.45932 > dns.google.domain:  aodv type 165 42
11:28:44.198550 IP dns.google.domain > 192.168.0.105.45932:  aodv type 165 137
11:28:44.198725 IP 192.168.0.105.55911 > dns.google.domain:  aodv type 167 44
11:28:44.203022 IP dns.google.domain > 192.168.0.105.55911:  aodv type 167 114
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

RPC

A remote procedure call, it is a protocol that one program can use to request service from a program located in another computer on a network without having to understand the network details. A procedure call is also known as a function call. For getting this protocol in our scan use the following command:

```
tcpdump -i eth0 -c5 -T rpc
```

```
root@kali:~# tcpdump -i eth0 -c5 -T rpc
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:38:12.761597 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, se
12:38:12.762338 IP 192.168.0.105.nfs > dns.google.849543424: reply ok 42
12:38:12.767098 IP dns.google.nfs > 192.168.0.105.849576323: reply Unknown rpc
12:38:12.767345 IP 192.168.0.105.nfs > dns.google.156238080: reply ok 44
12:38:12.771440 IP dns.google.nfs > 192.168.0.105.156272003: reply Unknown rpc
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

CNFP

Cisco NetFlow protocol, it is a network protocol developed by cisco for the collection and monitoring of network traffic, flow data generated by NetFlow enabled routers and switches. It exports traffic statistics as they record which are then collected by its collector. To get these detailed scans follow this command.

```
tcpdump -i eth0 -c5 -T cnfp
```

```
root@kali:~# tcpdump -i eth0 -c5 -T cnfp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:31:12.418661 IP 104.28.7.89 > 192.168.0.105: ICMP echo reply, id 1217,
11:31:12.419467 IP 192.168.0.105.54217 > dns.google.domain: NetFlow va715
11:31:12.423633 IP dns.google.domain > 192.168.0.105.54217: NetFlow va715
11:31:12.423896 IP 192.168.0.105.51206 > dns.google.domain: NetFlow v9885
11:31:12.428272 IP dns.google.domain > 192.168.0.105.51206: NetFlow v9885
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

LMP

Link Management Protocol, it is designed to ease the configuration and management of optical network devices. To understand the working of LMP in our network, we need to apply this protocol in our scan.

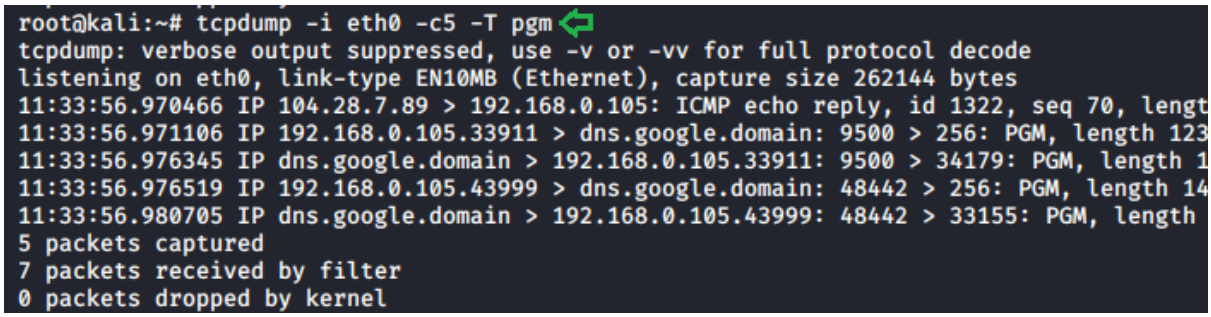
```
tcpdump -i eth0 -c5 -T lmp
```

```
root@kali:~# tcpdump -i eth0 -c5 -T lmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:32:08.687189 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217,
11:32:08.687826 IP 192.168.0.105.45378 > dns.google.domain: LMP version 13 pa
11:32:08.692504 IP dns.google.domain > 192.168.0.105.45378: LMP version 13 pa
11:32:08.692738 IP 192.168.0.105.36981 > dns.google.domain: LMP version 2 pac
11:32:08.699695 IP dns.google.domain > 192.168.0.105.36981: LMP version 2 pac
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

PGM

Pragmatic general multicast, it is a reliable multicast network transport protocol. It can provide a reliable sequence of packets to multiple recipients simultaneously. Which further makes it suitable for a multi-receiver file-transfer. To understand its working in our data traffic follows.

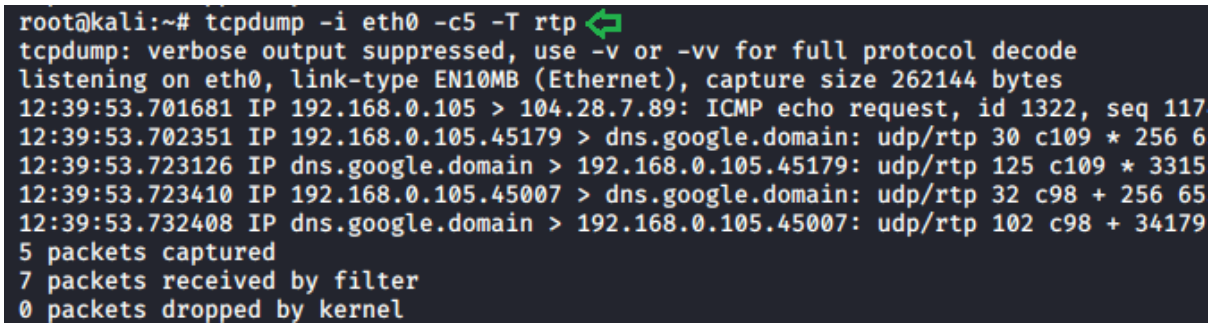
```
tcpdump -i eth0 -c5 -T pgm
```



RTP

Real-time application protocol can code multimedia data streams such as audio or video. It divides them into packets and transmits them over an IP network. To analyze this protocol in our traffic, we need to follow this command:

```
tcpdump -i eth0 -c5 -T rtp
```



RTCP

Real-time application control protocol. This protocol has all the capabilities of RTP along with additional control. With the help of this feature, we can control its working in our network environment. To understand the working of this protocol in our data traffic, apply these commands.

```
tcpdump -i eth0 -c5 -T rtcp
```

```
root@kali:~# tcpdump -i eth0 -c5 -T rtcp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:40:49.171715 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq
12:40:49.172472 IP 192.168.0.105.36642 > dns.google.domain:  type-0x93 1028
12:40:49.176281 IP dns.google.domain > 192.168.0.105.36642:  type-0x93 132624
12:40:49.176485 IP 192.168.0.105.46353 > dns.google.domain:  type-0xbf 1028
12:40:49.180535 IP dns.google.domain > 192.168.0.105.46353:  type-0xbf 136720
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

SNMP

Simple Network Management Protocol is an Internet standard protocol for collecting and organizing information about managed devices on IP networks for modifying that information to change device behavior. To see its working in our traffic, apply this command.

```
tcpdump -i eth0 -c5 -T snmp
```

```
root@kali:~# tcpdump -i eth0 -c5 -T snmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:41:53.751413 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq 1
12:41:53.752176 IP 192.168.0.105.51644 > dns.google.domain:  [len40<asnlen121]
12:41:53.756145 IP dns.google.domain > 192.168.0.105.51644:  [id?C/x/12]
12:41:53.756288 IP 192.168.0.105.49708 > dns.google.domain:  [asnlen? 42<48]
12:41:53.760596 IP dns.google.domain > 192.168.0.105.49708:  [len64<asnlen9096404
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

TFTP

Trivial File Transfer Protocol is a simple lockstep File transfer protocol that allows its client to get a file from a remote host. It is used in the early stages of node booting from a local area network. To understand its traffic, follow this command.

```
tcpdump -i eth0 -c5 -T tftp
```

```
root@kali:~# tcpdump -i eth0 -c5 -T tftp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:43:54.321604 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, s
12:43:54.322361 IP 192.168.0.105.44621 > dns.google.domain:  42 tftp-#31538
12:43:54.326272 IP dns.google.domain > 192.168.0.105.44621:  137 tftp-#31538
12:43:54.326500 IP 192.168.0.105.36604 > dns.google.domain:  44 tftp-#28987
12:43:54.335968 IP dns.google.domain > 192.168.0.105.36604:  114 tftp-#28987
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

VAT

Visual Audio Tool is developed by Van Jacobson and Steven McCanne. It is an electronic media processing for both sound and visual components. To understand its data packets in our traffic we need to apply these commands.

```
tcpdump -i eth0 -c5 -T vat
```

```
root@kali:~# tcpdump -i eth0 -c5 -T vat
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:45:18.141404 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq 1495,
12:45:18.142062 IP 192.168.0.105.48899 > dns.google.domain: udp/vt 42 721 / 37 [vat]
12:45:18.146249 IP dns.google.domain > 192.168.0.105.48899: udp/vt 137 721 / 37 [vat]
12:45:18.146537 IP 192.168.0.105.33052 > dns.google.domain: udp/vt 44 555 / 14 [vat]
12:45:18.150707 IP dns.google.domain > 192.168.0.105.33052: udp/vt 114 555 / 14 [vat]
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

WB

The program allows its users to draw and type the messages onto the canvas in a distributed whiteboard, synchronizing this to every other user on the same overlay network for the applications. New users also receive everything that other users have already stored on the whiteboard when they connect. To understand its data packets, follow this command.

```
tcpdump -i eth0 -c5 -T wb
```

```
root@kali:~# tcpdump -i eth0 -c5 -T wb
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:46:00.262133 IP 192.168.0.3.5050 > 239.255.255.250.5050:  wb-dop: 0.0.0.0:0<1:0
12:46:00.262772 IP 192.168.0.105.47580 > dns.google.domain:  wb-dop: 3.50.53.48:53
12:46:00.266430 IP 192.168.0.3.5050 > 192.168.0.255.5050:  wb-dop: 0.0.0.0:0<1:0>
12:46:00.441620 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq 15
12:46:00.522520 IP 104.28.7.89 > 192.168.0.105: ICMP echo reply, id 1322, seq 1537
5 packets captured
22 packets received by filter
0 packets dropped by kernel
```

VXLAN

Virtual Xtensible Local Area Network is a network virtualization tech that attempts to address the scalability problems associated with a large cloud computing area. It is a proposed Layer 3 encapsulation protocol that will make it easier for *network* engineers to scale-out cloud computing. To understand its data traffic follows these commands.

```
tcpdump -i eth0 -c5 -T vxlan
```

```
root@kali:~# tcpdump -i eth0 -c5 -T vxlan
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:47:39.460779 ARP, Request who-has 192.168.0.1 tell 192.168.0.105, length 28
12:47:39.461675 IP 192.168.0.105.40316 > dns.google.domain: VXLAN, flags [I] (0x49)
01:30:03:31:36:38 (oui Unknown) Unknown SSAP 0x32 > 00:00:00:00:01:31 (oui Ethernet
        0x0000:  3932 0769 6e2d 6164 6472 0461 7270 6100  92.in-addr.arpa.
        0x0010:  000c 0001                                ....
12:47:39.462505 ARP, Reply 192.168.0.1 is-at 1c:5f:2b:59:e1:24 (oui Unknown), lengt
12:47:39.465710 IP dns.google.domain > 192.168.0.105.40316: VXLAN, flags [I] (0x49)
01:30:03:31:36:38 (oui Unknown) Unknown SSAP 0x32 > 00:01:00:00:01:31 (oui Unknown)
        0x0000:  3932 0769 6e2d 6164 6472 0461 7270 6100  92.in-addr.arpa.
        0x0010:  000c 0001 0331 3638 0331 3932 0769 6e2d  .....168.192.in-
        0x0020:  6164 6472 0461 7270 6100 0006 0001 0001  addr.arpa.......
        0x0030:  5180 0026 096c 6f63 616c 686f 7374 0004  Q..&.localhost..
        0x0040:  726f 6f74 c04a 0000 0001 0009 3a80 0001  root.J......:...
        0x0050:  5180 0024 ea00 0001 5180                 Q..$....Q.
12:47:39.465960 IP 192.168.0.105.52338 > dns.google.domain: VXLAN, flags [.] (0xf2)
30:35:01:30:03:31 (oui Unknown) > 00:00:00:00:03:31 (oui Ethernet), ethertype Unknow
        0x0000:  0331 3932 0769 6e2d 6164 6472 0461 7270  .192.in-addr.arp
        0x0010:  6100 000c 0001                           a.....
5 packets captured
8 packets received by filter
0 packets dropped by kernel
```

These protocols use the forced packets parameter to get the fixed desired data traffic from the scan.

Promiscuous Mode

In computer networks, an interface controller uses promiscuous mode to cause tcpdump to pass on the traffic it receives to the CPU rather than passing it to the promiscuous mode, and it normally facilitates packet sniffing that can take place on a part of the LAN or router.

To configure promiscuous mode by following these commands.

```
ifconfig eth0 promisc
ifconfig eth0
```

```
root@kali:~# ifconfig eth0 promisc
root@kali:~# ifconfig eth0
eth0: flags=4419<UP,BROADCAST,RUNNING,PROMISC,MULTICAST>  mtu 1500
        inet 192.168.0.105  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::20c:29ff:fef6:d9c1  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:f6:d9:c1  txqueuelen 1000  (Ethernet)
        RX packets 29611  bytes 29787736 (28.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3603  bytes 347896 (339.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

After enabling the promiscuous mode in our network, let us capture some packets with the help of this by applying these commands.

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

```
root@kali:~# tcpdump -i eth0 -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
13:05:14.208663 IP 192.168.0.125 > del03s10-in-f4.1e100.net: ICMP echo request, id 3383,
13:05:14.209298 IP 192.168.0.105.34088 > dns.google.domain: 4413+ PTR? 4.161.217.172.in-a
13:05:14.213031 IP del03s10-in-f4.1e100.net > 192.168.0.125: ICMP echo reply, id 3383, se
13:05:14.213597 IP dns.google.domain > 192.168.0.105.34088: 4413 1/0/0 PTR del03s10-in-f4
13:05:14.213841 IP 192.168.0.105.40126 > dns.google.domain: 64501+ PTR? 125.0.168.192.in-
13:05:14.217846 IP dns.google.domain > 192.168.0.105.40126: 64501 NXDomain* 0/1/0 (114)
13:05:14.218145 IP 192.168.0.105.57838 > dns.google.domain: 52997+ PTR? 8.8.8.8.in-addr.a
13:05:14.221873 IP dns.google.domain > 192.168.0.105.57838: 52997 1/0/0 PTR dns.google. (
13:05:14.221996 IP 192.168.0.105.41715 > dns.google.domain: 54464+ PTR? 105.0.168.192.in-
13:05:14.225802 IP dns.google.domain > 192.168.0.105.41715: 54464 NXDomain* 0/1/0 (114)
10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

No Promiscuous Mode

In the previous parameter, we learned about the promiscuous mode, which allows a network interface card to pass all frames it receives to the OS for processing, unlike the traditional operation, where it only passes frames destined for the NIC's MAC address or a broadcast address to the OS. Generally, users utilize promiscuous mode to "sniff" all traffic on the wire. But if we want to switch to multicast mode instead of promiscuous mode. Then we need to use –no-promiscuous-mode parameter, which helps us to which the mode without changing the network settings.

```
tcpdump -i eth0 -c 5 --no-promiscuous-mode
```

```
root@kali:~# tcpdump -i eth0 -c 5 --no-promiscuous-mode
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:11:22.927767 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq 3044,
13:11:22.928460 IP 192.168.0.105.58891 > dns.google.domain: 48810+ PTR? 89.7.28.104.i
13:11:22.932397 IP dns.google.domain > 192.168.0.105.58891: 48810 NXDomain 0/1/0 (137
13:11:22.932561 IP 192.168.0.105.49327 > dns.google.domain: 64252+ PTR? 105.0.168.192
13:11:22.936869 IP dns.google.domain > 192.168.0.105.49327: 64252 NXDomain* 0/1/0 (11
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

Version Information

Let's begin with one of the simplest commands so that we can understand and relate all the practicals during the article. We can use this parameter to print the tcpdump, libpcap and OpenSSL string.

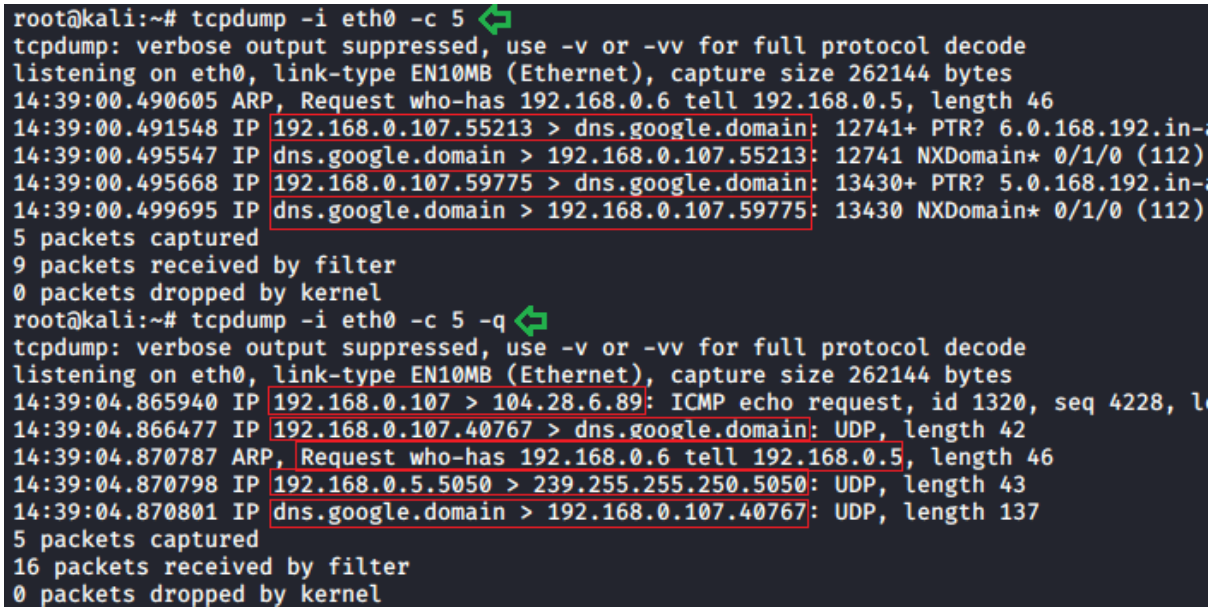
```
tcpdump -version
```

```
root@kali:~# tcpdump --version
tcpdump version 4.9.3
libpcap version 1.9.1 (with TPACKET_V3)
OpenSSL 1.1.1d  10 Sep 2019
```

Quick Mode

Arguably if the network is very quiet, performing any operation during that time will take more time than usual. The person who developed tcpdump thought of this conundrum and gave us the way to speed up the process by using the "-q" parameter. It will print less information about protocols and data packets to save time.

```
tcpdump -i eth0 -c 5
tcpdump -i eth0 -c 5 -q
```



Verbose Mode

The verbose mode is famous to provide extra information regarding operations. in TCPDump, verbose mode provides such the information too. For instance, time to live, identification, total length. It can also enable additional packet integrity checks such as verifying the IP and ICMP header checksum values.`

TCPDump provides us with plenty of parameters that we move around this mode like -v, -vv, -vvv, where each parameter has its unique efficiency.

- **-v** parameter is the traditional verbose mode.
- **-vv** parameter surpasses the traditional verbose mode, printing additional fields from NFS (Network File System) reply packets and fully decoding SMB packets.
- **-vvv** parameter has something more to provide like tenet options etc.

```
tcpdump -i eth0 -c 2
tcpdump -i eth0 -c 2 -v
tcpdump -i eth0 -c 2 -vv
tcpdump -i eth0 -c 2 -vvv
```

```
root@kali:~# tcpdump -i eth0 -c 2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:21:47.829457 IP 192.168.0.107 > 104.28.7.89: ICMP echo request, id 1301, seq
14:21:47.832213 IP 192.168.0.107.52206 > dns.google.domain: 42610+ PTR? 89.7.28.
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 2 -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 byt
14:21:50.921740 IP (tos 0x0, ttl 55, id 13449, offset 0, flags [none], proto ICM
    104.28.7.89 > 192.168.0.107: ICMP echo reply, id 1301, seq 63, length 64
14:21:50.922187 IP (tos 0x0, ttl 64, id 41410, offset 0, flags [DF], proto UDP (
    192.168.0.107.50629 > dns.google.domain: 35210+ PTR? 107.0.168.192.in-addr.a
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 2 -vv
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 byt
14:21:54.852840 IP (tos 0x0, ttl 64, id 38498, offset 0, flags [DF], proto ICMP
    192.168.0.107 > 104.28.7.89: ICMP echo request, id 1301, seq 67, length 64
14:21:54.853266 IP (tos 0x0, ttl 64, id 41583, offset 0, flags [DF], proto UDP (
    192.168.0.107.38281 > dns.google.domain: [bad udp cksum 0xd166 → 0x395b!] 2
2 packets captured
7 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 2 -vvv
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 byt
14:21:58.867572 IP (tos 0x0, ttl 64, id 38938, offset 0, flags [DF], proto ICMP
    192.168.0.107 > 104.28.7.89: ICMP echo request, id 1301, seq 71, length 64
14:21:58.868097 IP (tos 0x0, ttl 64, id 41750, offset 0, flags [DF], proto UDP (
    192.168.0.107.60041 > dns.google.domain: [bad udp cksum 0xd166 → 0xae3c!] 4
2 packets captured
7 packets received by filter
0 packets dropped by kernel
```

HTTP Requests

As we all know, HTTP Requests is an information message from the client to a server over the hypertext transfer protocol (HTTP). It has various methods to deliver this information. These methods are case-sensitive and always mentioned in the UPPERCASE. Through tcpdump, we can capture these requests to analyze the traffic sent over the said protocol traffic.

The methods which we can capture through tcpdump are the following:

- **GET-** This method is used to retrieve the information from the given server using a given URL. Requests using GET should only retrieve data and have no other effect on it. We can also capture this request with the help of tcpdump.

```
tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420'
```

```
root@kali:~# tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0×f0) >> 2):4] = 0×47455420'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:30:01.926242 IP (tos 0×0, ttl 64, id 42167, offset 0, flags [DF], proto TCP (6),
    192.168.0.107.48234 > rs202995.rs.hosteurope.de.http: Flags [P.], cksum 0×a554
gth 345: HTTP, length: 345
        GET / HTTP/1.1
        Host: testphp.vulnweb.com
        User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Connection: keep-alive
        Upgrade-Insecure-Requests: 1
        Cache-Control: max-age=0

E.....@.@......k..2..j.P....N.Pd.....T.....
•–.$....GET / HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

- **POST-** This request is used to send data to the server. Like customer information, file upload, etc. using HTML forms. Traffic over this protocol can analyzed using the following command :

```
tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x504f5354'
```

```
root@kali:~# tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0×f0) >> 2):4] = 0×504f5354'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:32:35.944089 IP (tos 0×0, ttl 64, id 59541, offset 0, flags [DF], proto TCP (6)
    192.168.0.107.33312 > 117.18.237.29.http: Flags [P.], cksum 0×24dd (incorrect
TTP, length: 371
        POST / HTTP/1.1
        Host: ocsp.digicert.com
        User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefo
        Accept: */*
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Content-Type: application/ocsp-request
        Content-Length: 83
        Connection: keep-alive

E.....@.@•–x ... ku.... .P.....Ex.....$......
 ... ?/ ... POST / HTTP/1.1
Host: ocsp.digicert.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/ocsp-request
Content-Length: 83
Connection: keep-alive
```

- **Request-URL-** It is a uniform resource identifier, which identifies the resource on which we need to apply requests. The most common form of this is used to identify a resource on a server. If a client wants to retrieve the data directly from the server, where it originated, then

it would create a connection to port 80 of the host and send the request. These requests can be captured using the following commands:

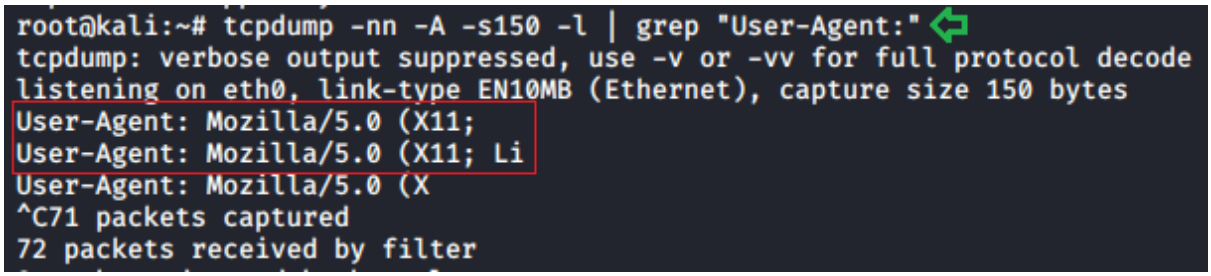
```
tcpdump -v -n -l | egrep -i "POST /|GET /|Host:"
```



User Agent

With TCPDump, you can also see which traffic is generated from which application. We can also find the user agents in our data traffic by using the following command :

```
tcpdump -nn -A -s150 -l | grep "User-Agent:"
```



Port Range

Some ordinary port filters help us to analyze the traffic on a particular port. But in tcpdump, we give our scan a range of ports through which it can monitor the destination of TCP/UDP or other port-based network protocols.

```
tcpdump -i eth0 portrange 21-80
```

```
root@kali:~# tcpdump -i eth0 portrange 21-80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:57:54.698585 IP 192.168.0.107.48999 > dns.google.domain: 23191+ A? ib.adnxs.com. (30)
14:57:54.699026 IP 192.168.0.107.54455 > dns.google.domain: 10778+ PTR? 8.8.8.8.in-addr.a
14:57:54.699576 IP 192.168.0.107.35047 > dns.google.domain: 11848+ A? apex.go.sonobi.com.
14:57:54.708128 IP dns.google.domain > 192.168.0.107.48999: 23191 10/0/0 CNAME g.geogslb.
3, A 103.43.90.20 (219)
14:57:54.710840 IP dns.google.domain > 192.168.0.107.54455: 10778 1/0/0 PTR dns.google. (
14:57:54.710949 IP 192.168.0.107.50063 > dns.google.domain: 65122+ PTR? 107.0.168.192.in-
14:57:54.715651 IP dns.google.domain > 192.168.0.107.35047: 11848 2/0/0 CNAME lax-1-apex.
14:57:54.715669 IP dns.google.domain > 192.168.0.107.50063: 65122 NXDomain* 0/1/0 (114)
14:57:54.715717 IP 192.168.0.107.35047 > dns.google.domain: 43083+ AAAA? apex.go.sonobi.c
14:57:54.719375 IP dns.google.domain > 192.168.0.107.35047: 43083 1/1/0 CNAME lax-1-apex.
14:57:55.062444 IP 192.168.0.107.42637 > dns.google.domain: 27452+ A? csi.gstatic.com. (3
14:57:55.066502 IP dns.google.domain > 192.168.0.107.42637: 27452 1/0/0 A 172.217.167.163
14:57:56.179622 IP 192.168.0.107.38126 > dns.google.domain: 26680+ A? testphp.vulnweb.com
14:57:56.179721 IP 192.168.0.107.38126 > dns.google.domain: 9789+ AAAA? testphp.vulnweb.c
14:57:56.184376 IP dns.google.domain > 192.168.0.107.38126: 26680 1/0/0 A 176.28.50.165 (
14:57:56.184408 IP dns.google.domain > 192.168.0.107.38126: 9789 0/1/0 (99)
14:57:56.184742 IP 192.168.0.107.49870 > rs202995.rs.hosteurope.de.http: Flags [S], seq 1
```

Destination

To check the flow of data in network traffic towards a particular destination, use the following command for this:

```
tcpdump -i eth0 dst google.com
```

```
root@kali:~# tcpdump -i eth0 dst google.com
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:01:00.616734 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.622475 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.623480 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.629658 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.630288 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.632857 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.636334 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.636682 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
15:01:00.636755 IP 192.168.0.107.40862 > del11s05-in-f14.1e100.net.https:
```

Source

To check the data traffic coming from a particular source, we can follow the command given below :

tcpdump -i eth0 src google.com

```
root@kali:~# tcpdump -i eth0 src google.com
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:01:59.119776 IP del11s05-in-f14.1e100.net.https > 192.168.0.107.40862: Flags [.]
15:01:59.120836 IP del11s05-in-f14.1e100.net.https > 192.168.0.107.40862: Flags [P.
```

Network

To find the packets going to or from in a particular network, we can use the following function to analyze this traffic:

```
tcpdump net 192.168.0.1 -c5
```

```
root@kali:~# tcpdump net 192.168.0.1 -c5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:12:07.034733 IP 192.168.0.107 > 192.168.0.1: ICMP echo request, id 1862, seq 20, length 64
04:12:07.040277 IP 192.168.0.1 > 192.168.0.107: ICMP echo reply, id 1862, seq 20, length 64
04:12:08.038253 IP 192.168.0.107 > 192.168.0.1: ICMP echo request, id 1862, seq 21, length 64
04:12:08.043569 IP 192.168.0.1 > 192.168.0.107: ICMP echo reply, id 1862, seq 21, length 64
04:12:09.042053 IP 192.168.0.107 > 192.168.0.1: ICMP echo request, id 1862, seq 22, length 64
5 packets captured
5 packets received by filter
0 packets dropped by kernel
```

TCP Packets

TCP packet is the format consists of the fields such as source port and destination port field. Through these fields, we can identify the endpoints of the connections and can also capture these TCP packets in its various flag format. i.e. SYN, RST and ACK.

- **SYN-** The SYN flag synchronizes sequence numbers to initiate a TCP connection. We can capture this packet from traffic with the help of tcpdump.

```
tcpdump 'tcp[tcpflags] == tcp-syn'
```

```
root@kali:~# tcpdump 'tcp[tcpflags] == tcp-syn'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:14:11.261944 IP 192.168.0.107.57644 > rs202995.rs.hosteurope.de.http: Flags [S], seq 1623426266,
04:14:19.454764 IP 192.168.0.107.54374 > ec2-54-208-84-166.compute-1.amazonaws.com.https: Flags [S],
04:14:30.969669 IP 192.168.0.107.57648 > rs202995.rs.hosteurope.de.http: Flags [S], seq 4038658565,
04:14:36.543910 IP 192.168.0.107.54378 > ec2-54-208-84-166.compute-1.amazonaws.com.https: Flags [S],
04:14:36.811148 IP 192.168.0.107.42458 > ec2-54-208-84-166.compute-1.amazonaws.com.http: Flags [S],
04:14:36.822624 IP 192.168.0.107.54382 > ec2-54-208-84-166.compute-1.amazonaws.com.https: Flags [S],
04:14:37.055320 IP 192.168.0.107.42462 > ec2-54-208-84-166.compute-1.amazonaws.com.http: Flags [S],
04:14:40.809317 IP 192.168.0.107.54386 > ec2-54-208-84-166.compute-1.amazonaws.com.https: Flags [S],
```

- **RST-** The system knows the RST flag as the reset flag. If a packet is sent to a particular host that expects it, the receiver sends this flag to the sender. The system uses the RST flag to re-establish a TCP end-to-end connection. We can capture this flag from our data traffic with the help of tcpdump.

```
tcpdump 'tcp[tcpflags] == tcp-rst'
```

```
root@kali:~# tcpdump 'tcp[tcpflags] == tcp-rst'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:15:58.610080 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.610178 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917610 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917684 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917698 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917711 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:16:03.730516 IP 192.168.0.107.57670 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:03.730792 IP 192.168.0.107.57670 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:05.983486 IP 192.168.0.107.57674 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:05.983838 IP 192.168.0.107.57674 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:07.853537 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:16:07.853847 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:16:08.021311 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:08.021497 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:08.022655 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:08.023500 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
```

- **ACK-** The Acknowledgement flag refers to the ACK flag. We use this flag to acknowledge that our data packet has been successfully received. We can capture these flags with tcpdump to study our data traffic.

```
tcpdump 'tcp[tcpflags] == tcp-ack' -c5
```

```
root@kali:~# tcpdump 'tcp[tcpflags] == tcp-ack' -c5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:18:11.472063 IP 192.168.0.107.51778 > 103.95.86.61.https: Flags [.], ack 31208648, win 50
 val 405310300 ecr 2371920174], length 0
04:18:11.472362 IP 192.168.0.107.45016 > a23-32-28-63.deploy.static.akamaitechnologies.com.h
92513610, win 501, options [nop,nop,TS val 1560136604 ecr 3342172352], length 0
04:18:11.482409 IP 103.95.86.61.https > 192.168.0.107.51778: Flags [.], ack 1, win 243, opti
71930412 ecr 405259543], length 0
04:18:11.484155 IP a23-32-28-63.deploy.static.akamaitechnologies.com.http > 192.168.0.107.45
 win 235, options [nop,nop,TS val 3342182590 ecr 1560085786], length 0
04:18:11.737741 IP 192.168.0.107.34024 > cloudproxy10024.sucuri.net.http: Flags [.], ack 305
ons [nop,nop,TS val 3082716552 ecr 793487062], length 0
```

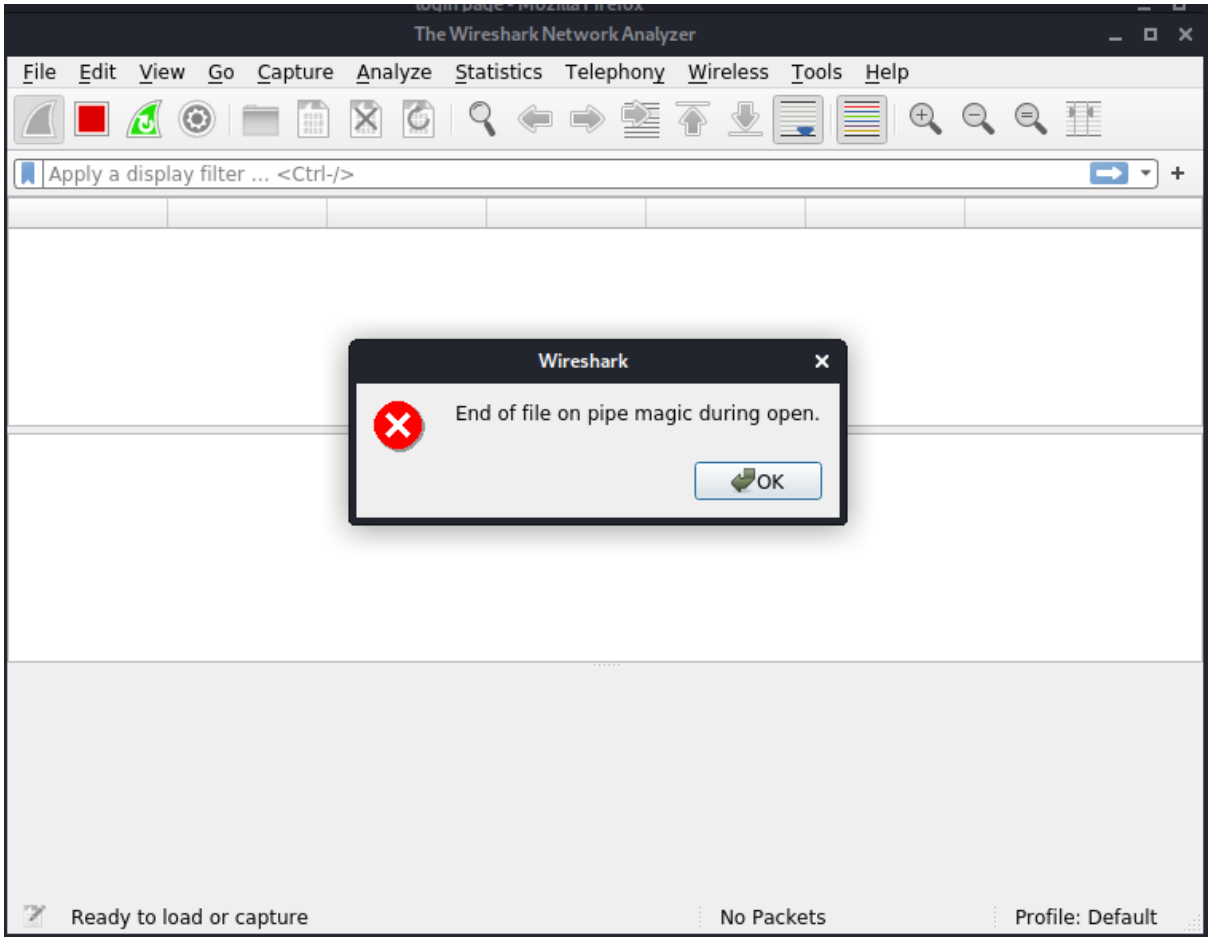
Tcpdump to Wireshark

The only difference between the Wireshark and TCPDump is that Wireshark is GUI while tcpdump is a command-line tool. But with the help of a few sources, we use a command on tcpdump and view our data traffic results in Wireshark which, we find is the best way to analyze our traffic. This can be done using the following command :

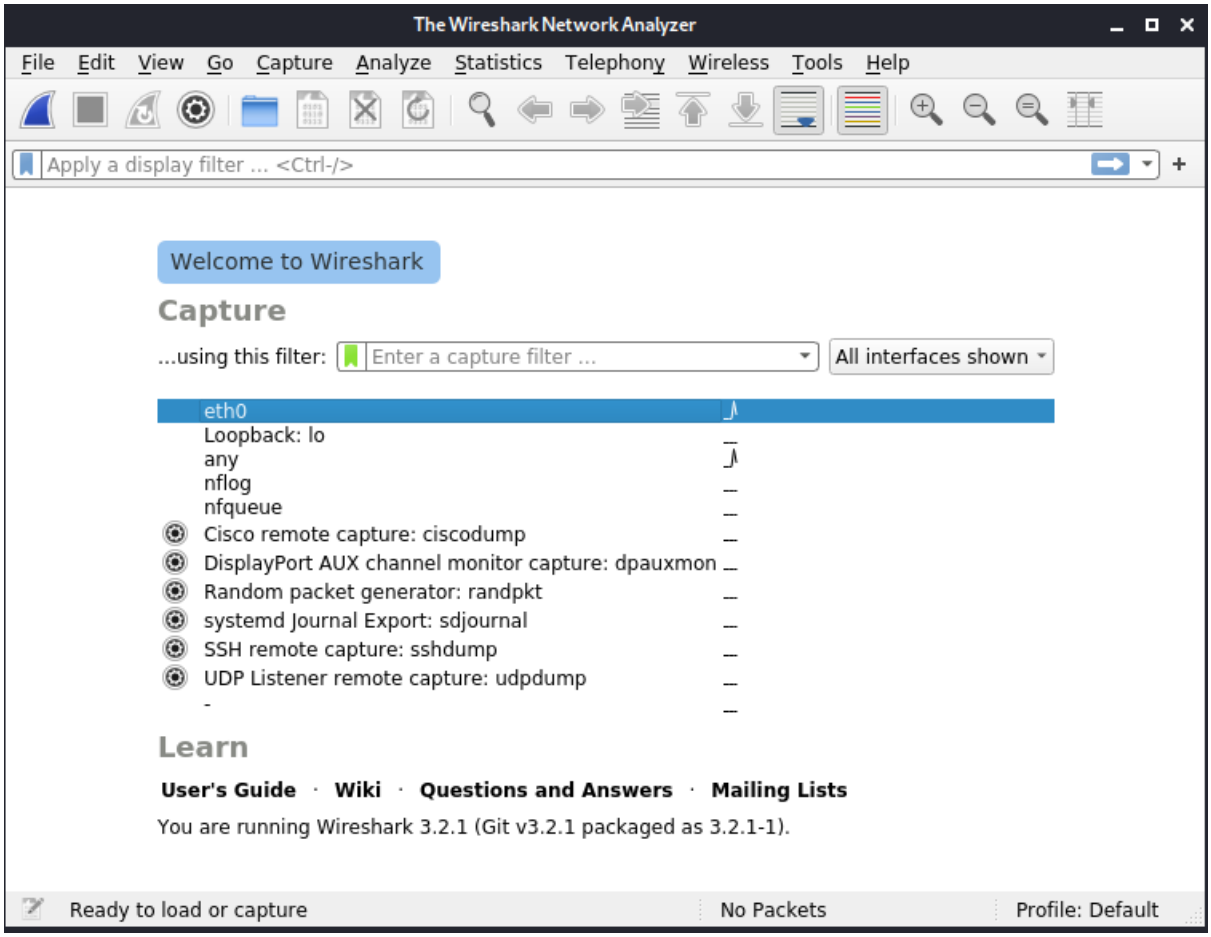
```
ssh root@remotesystem 'tcpdump -c20 -nn -w - not port 22' | wireshark -k -i -
```

```
root@kali:~# ssh root@remotesystem 'tcpdump -c 20 -nn -w - not port 22' | wireshark -k -i -
ssh: Could not resolve hostname remotesystem: Name or service not known
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
04:43:13.310     Main Warn QXcbConnection: XCB error: 3 (BadWindow), sequence: 5551, resource id:
10543229, major code: 40 (TranslateCoords), minor code: 0
```

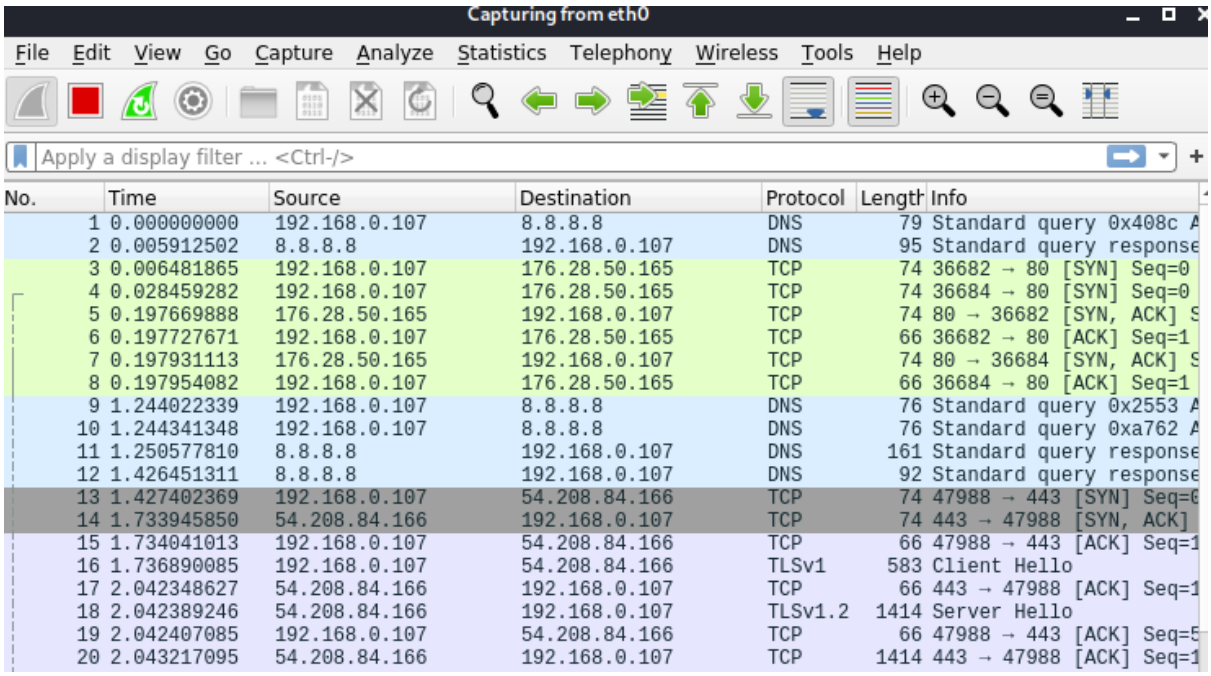
After running this command, it will immediately open the Wireshark and will ask a few questions about our scan. Press **OK** to move further.



After this, it will ask you which network interface we want to capture the data packets. In our case it will be eth0, so we are selecting that network interface.



After completing all the formalities our live data capture screen will appear with our captured data packets.



By following these steps, we can run a command for tcpdump and capture its results in Wireshark.

Conclusion

In conclusion, Tcpdump is a powerful command-line tool for network packet analysis, allowing users to troubleshoot network issues and analyze security tools. With its various parameters and options, Tcpdump provides a wide range of features, including capturing traffic on specific interfaces, filtering packets, and printing packet contents in ASCII. Additionally, Tcpdump can be used to capture HTTP requests, analyze TCP packets, and even export data to Wireshark for further analysis. Overall, Tcpdump is a valuable tool for network administrators and security professionals, providing detailed insight into network traffic and helping to identify and resolve issues.

iGNITE Technologies

FOLLOW US ON *social media*



TWITTER



DISCORD



GITHUB



LINKEDIN

CONTACT US

FOR MORE DETAILS

+91 95993-87841

www.ignitetechnologies.in

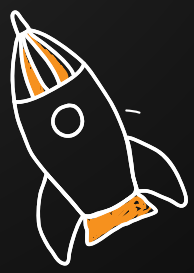
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials

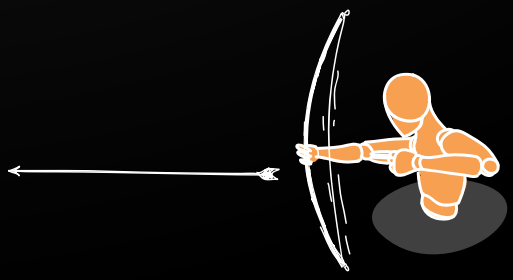


ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics



EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux