Comprehensive Guide to tcpdump (Part 2)

March 19, 2020   By Raj Chandel

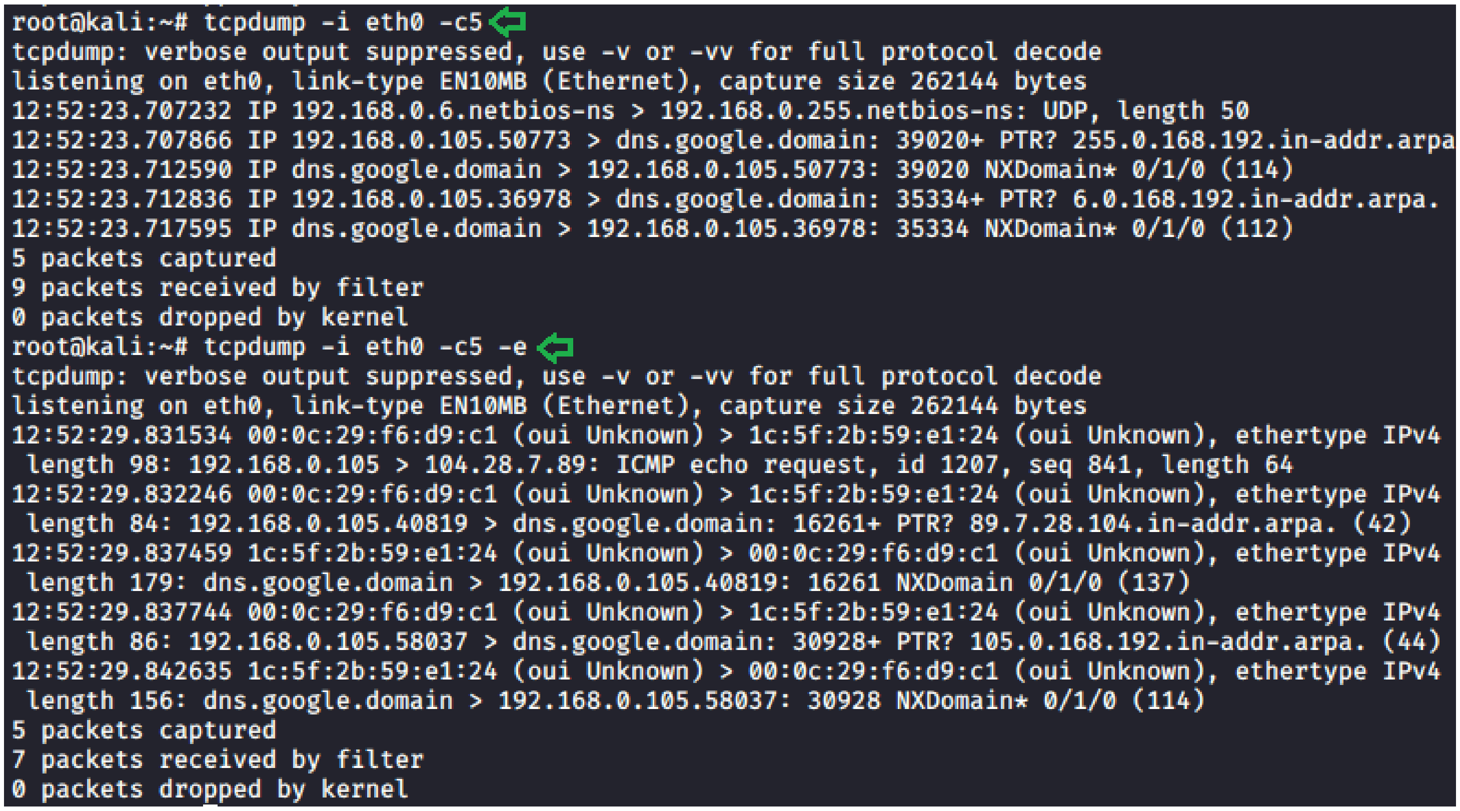
In the previous article of tcpdump, we learned about some basic functionalities of this amazing tool called tcpdump. If you haven't check until now, click here. Hence, in this part, we will cover some of the advance options and data types. So that we can analyze our data traffic in a much faster way.

Table of Content

Link Level Header

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

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

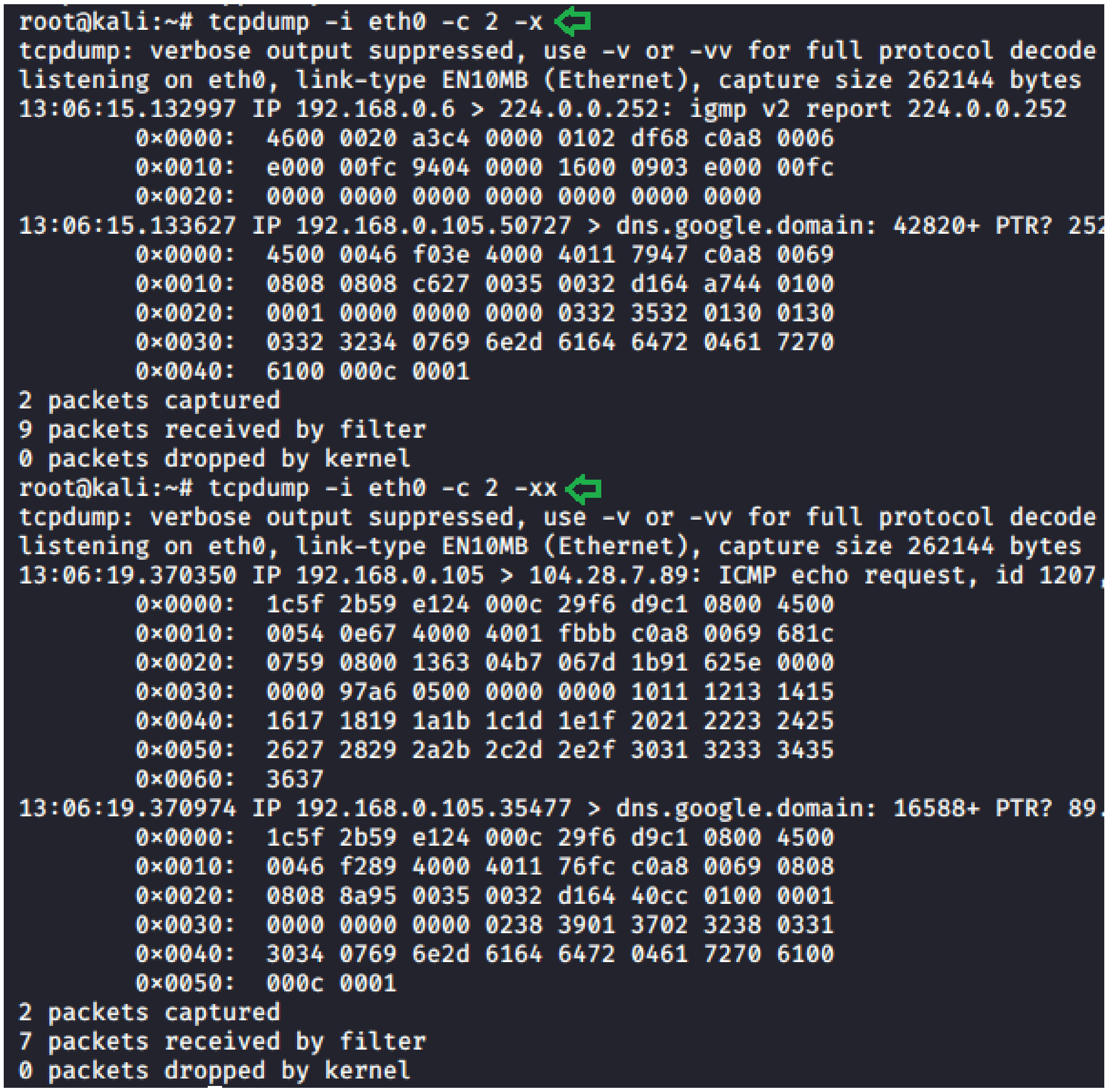


```
root@kali:~# tcpdump -i eth0 -c5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:52:23.707232 IP 192.168.0.6.netbios-ns > 192.168.0.255.netbios-ns: UDP, length 50
12:52:23.707866 IP 192.168.0.105.50773 > 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.

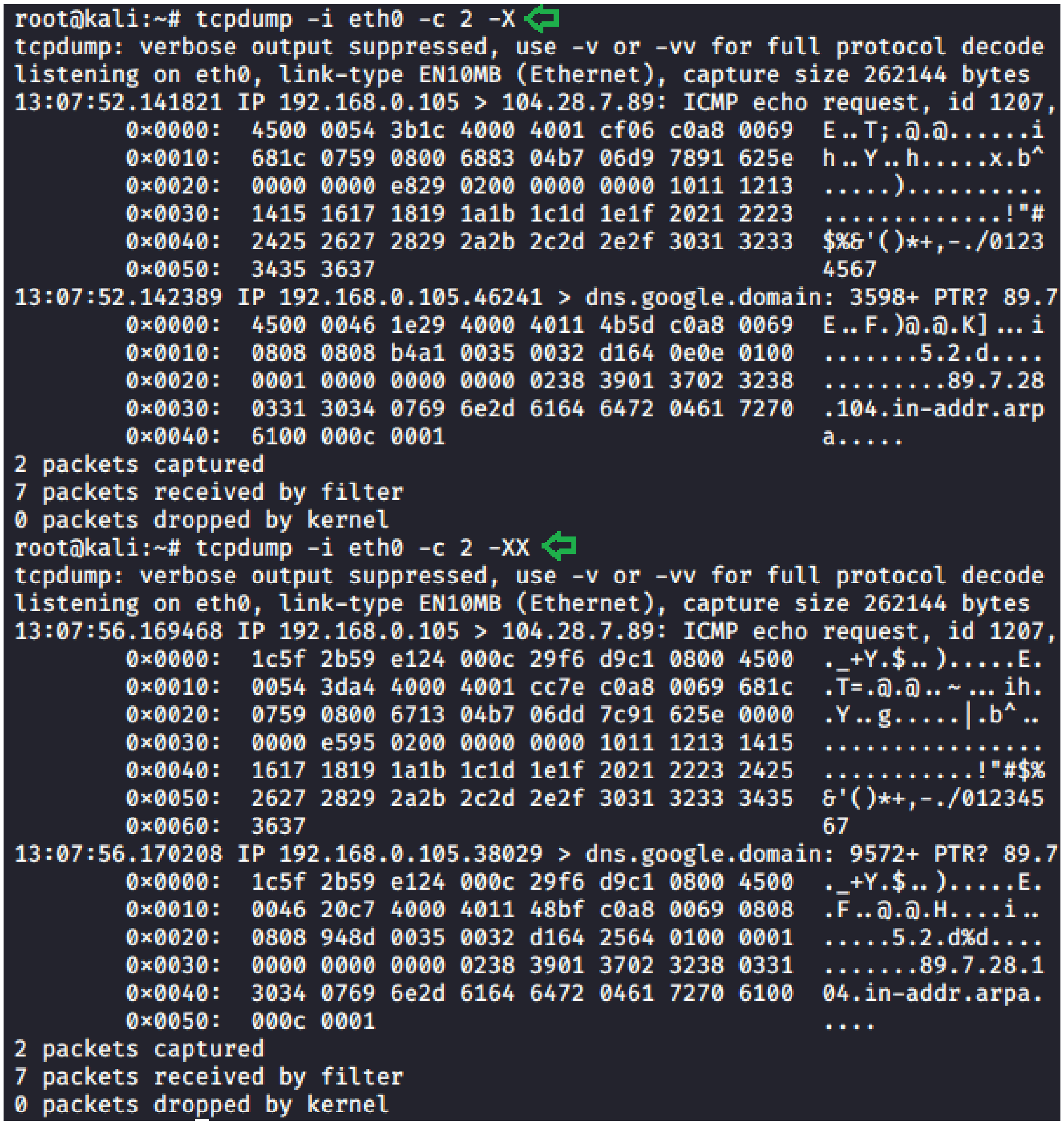
```
tcpdump -i eth0 -c 2 -x
tcpdump -i eth0 -c 2 -xx
```



```
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
```



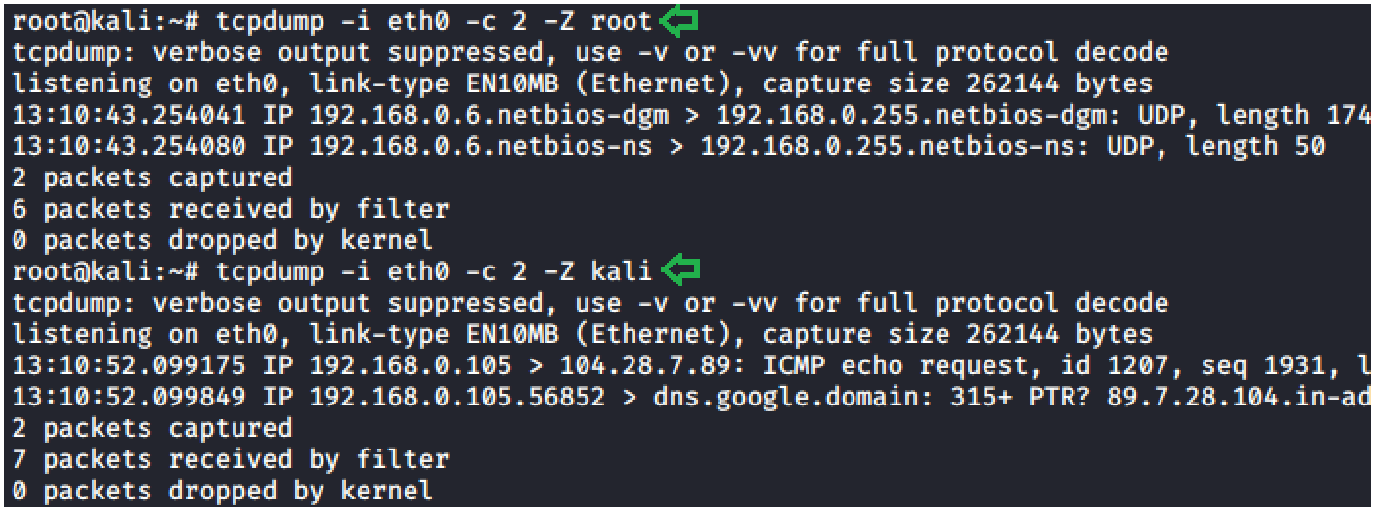
```
root@kali:~# tcpdump -i eth0 -c 2 -X
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:07:52.141821 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207,
        0x0000:  4500 0054 3b1c 4000 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

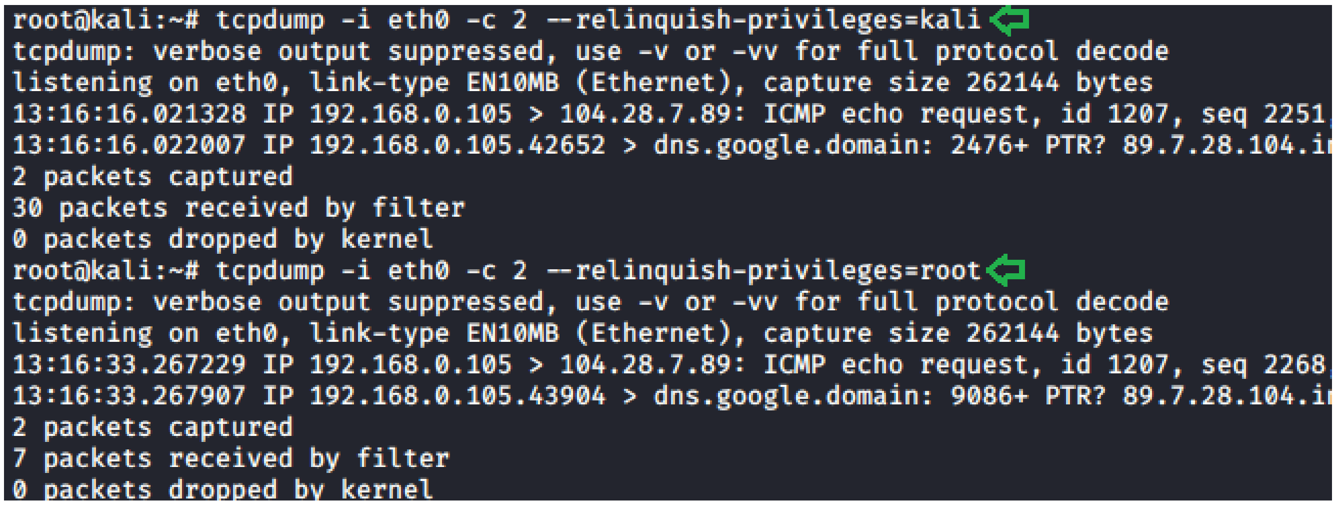
If we are running tcpdump as root then before opening any saved file for analysis, you will observe that it changes the user ID to the user and the group IDs to the primary group of its users.

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
```

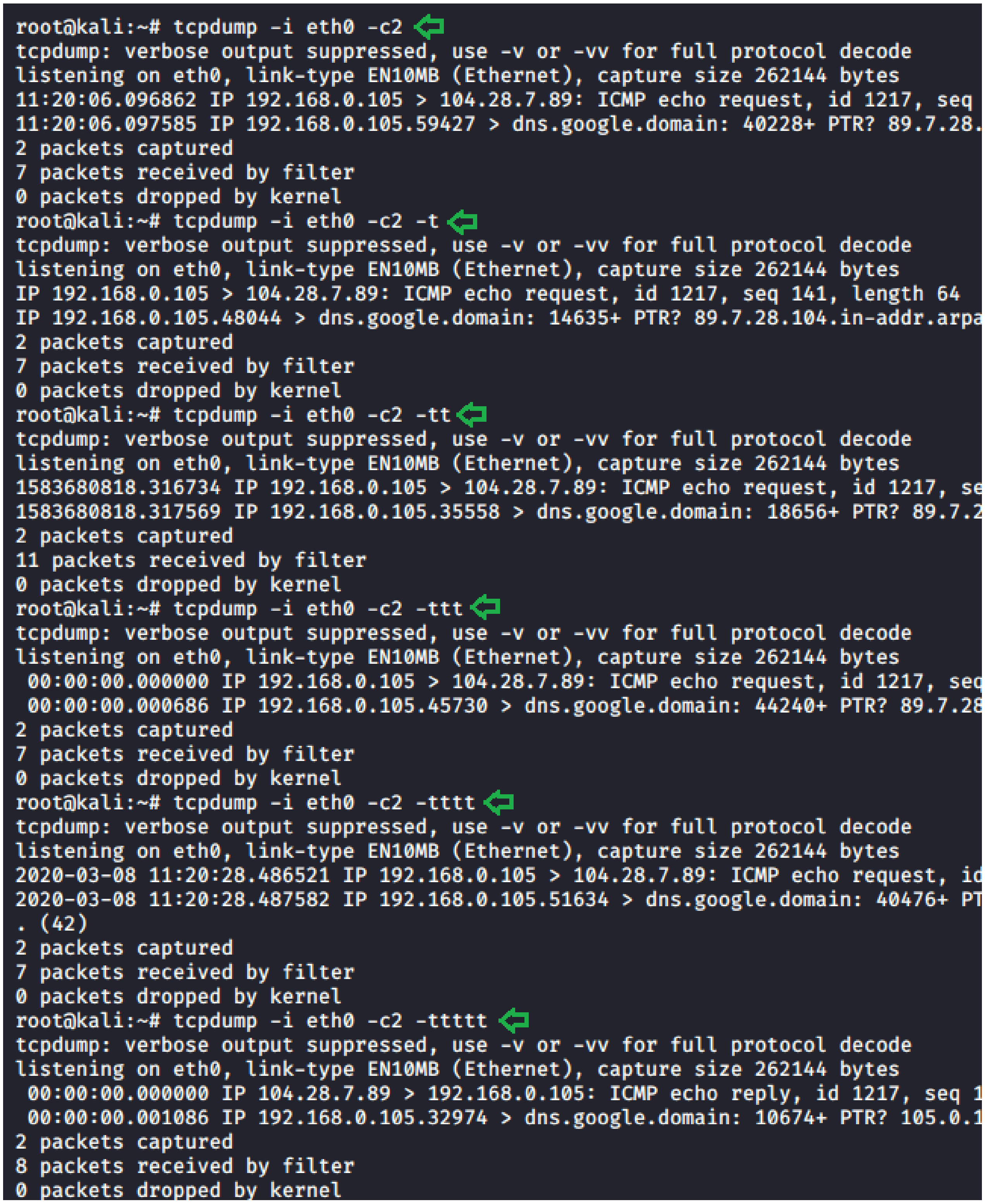
Timestamp Precision

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

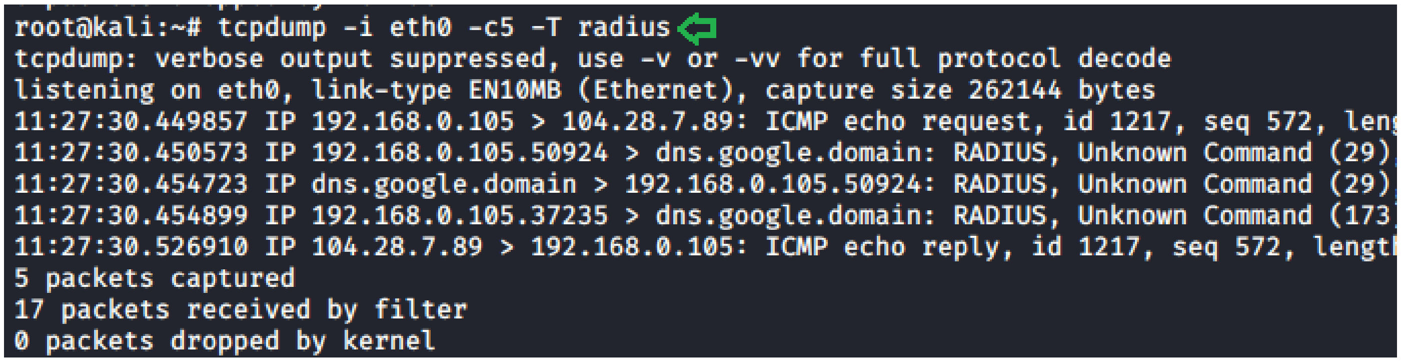
In tcpdump, we can force our scan of data traffic to show some particular protocol. When using the force packet feature, defined by selected any "expression" we can interpret specified type. With the help of the -T parameter, we can force data packets to show only the desired protocol results.

The basic syntax of all force packets will remain the same as other parameters -T followed by the desired protocol. 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
```

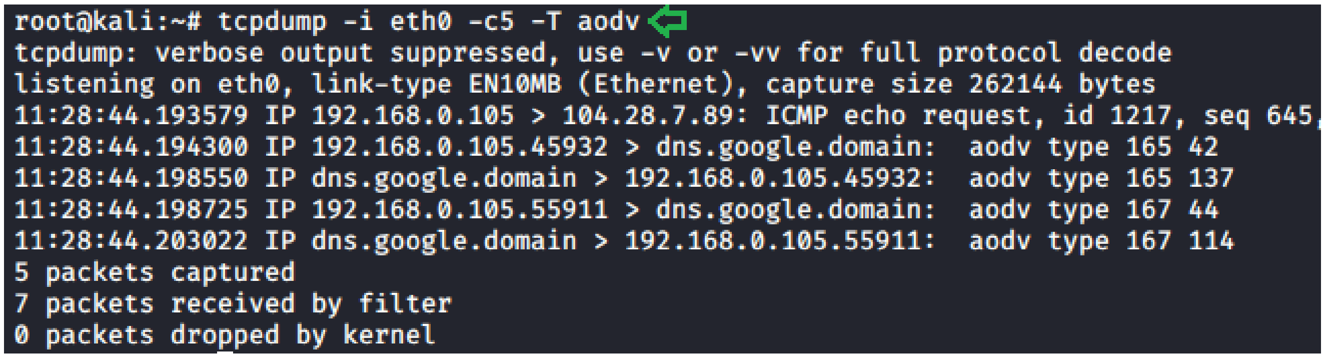


```
root@kali:~# tcpdump -i eth0 -c5 -T radius
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:27:30.449857 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, seq 572, len
11:27:30.450573 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
```

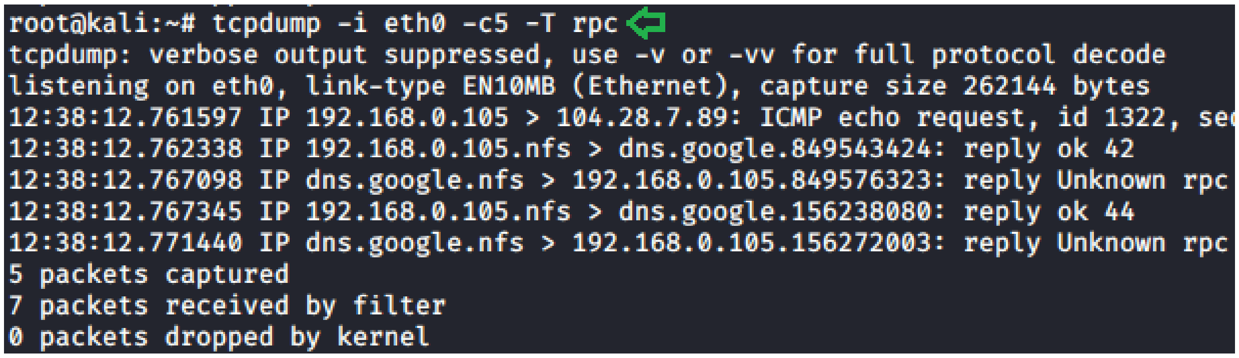


```
root@kali:~# tcpdump -i eth0 -c5 -T aodv
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:28:44.193579 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217, seq 645,
11:28:44.194300 IP 192.168.0.105.45932 > 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
```

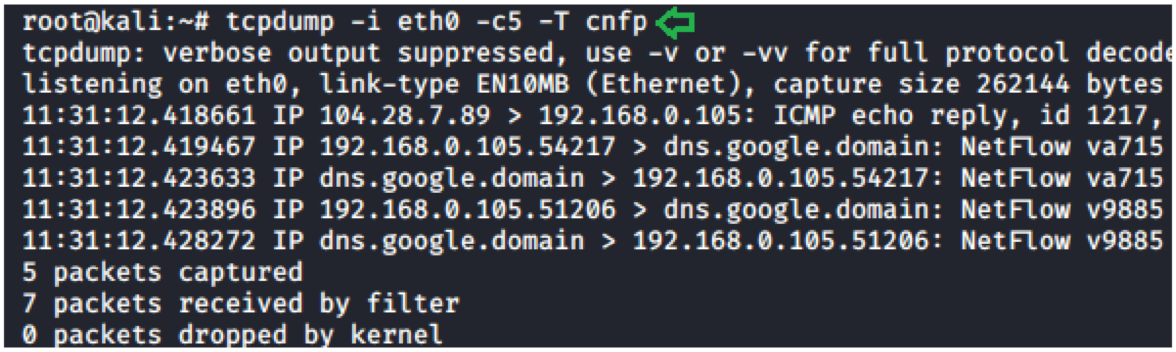


```
root@kali:~# tcpdump -i eth0 -c5 -T rpc
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:38:12.761597 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, se
12:38:12.762338 IP 192.168.0.105.nfs > 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
```

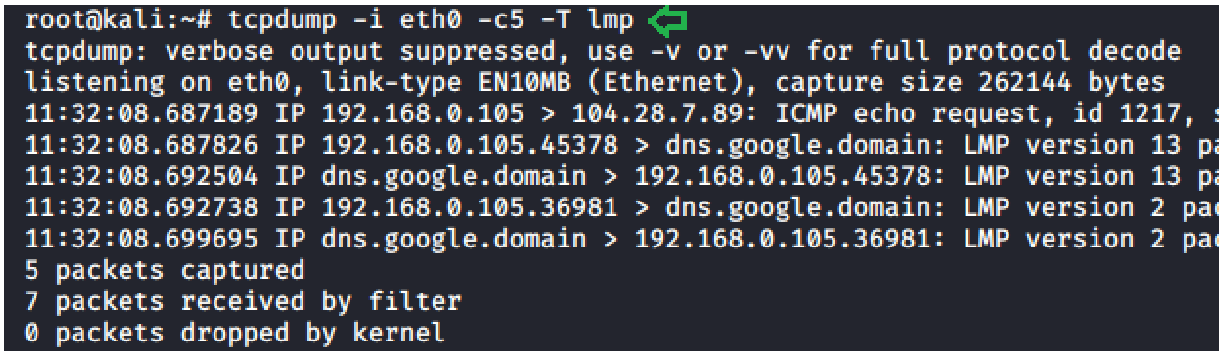


```
root@kali:~# tcpdump -i eth0 -c5 -T cnfp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:31:12.418661 IP 104.28.7.89 > 192.168.0.105: ICMP echo reply, id 1217,
11:31:12.419467 IP 192.168.0.105.54217 > 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
```

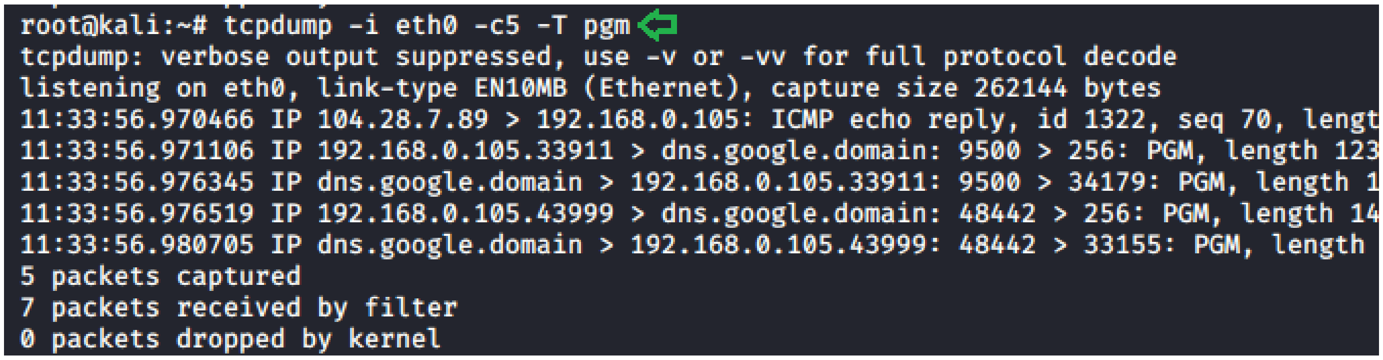


```
root@kali:~# tcpdump -i eth0 -c5 -T lmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:32:08.687189 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1217,
11:32:08.687826 IP 192.168.0.105.45378 > 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
```

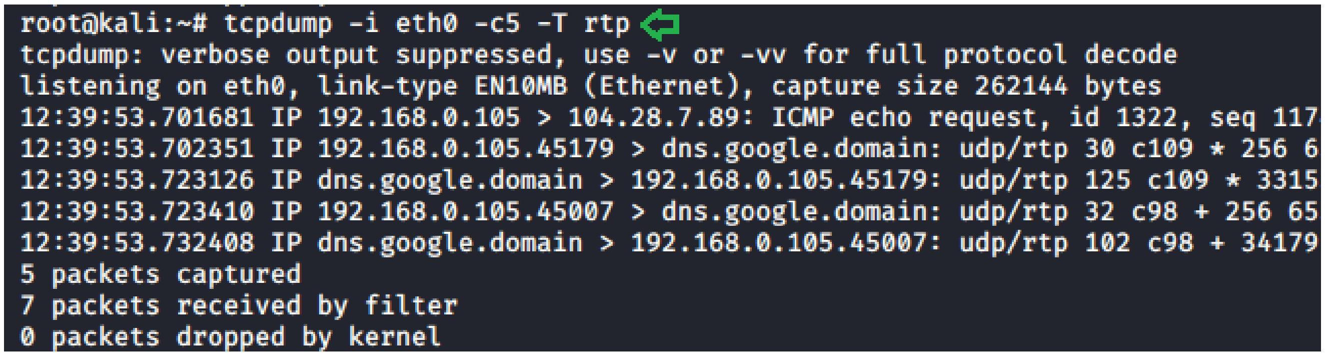


```
root@kali:~# tcpdump -i eth0 -c5 -T pgm
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:33:56.970466 IP 104.28.7.89 > 192.168.0.105: ICMP echo reply, id 1322, seq 70, lengt
11:33:56.971106 IP 192.168.0.105.33911 > dns.google.domain: 9500 > 256: PGM, length 123
11:33:56.976345 IP dns.google.domain > 192.168.0.105.33911: 9500 > 34179: PGM, length 1
11:33:56.976519 IP 192.168.0.105.43999 > dns.google.domain: 48442 > 256: PGM, length 14
11:33:56.980705 IP dns.google.domain > 192.168.0.105.43999: 48442 > 33155: PGM, length
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

RTP

Real-time application protocol, it 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
```

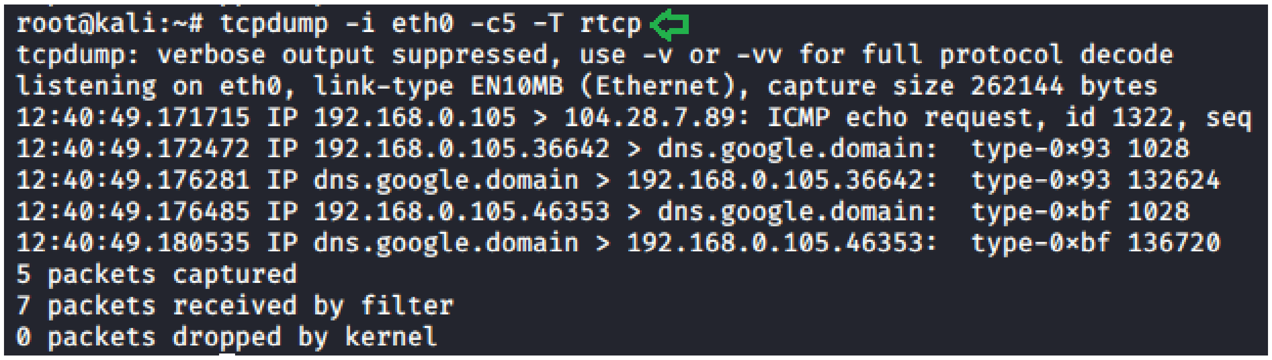


```
root@kali:~# tcpdump -i eth0 -c5 -T rtp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:39:53.701681 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq 117
12:39:53.702351 IP 192.168.0.105.45179 > dns.google.domain: udp/rtp 30 c109 * 256 6
12:39:53.723126 IP dns.google.domain > 192.168.0.105.45179: udp/rtp 125 c109 * 3315
12:39:53.723410 IP 192.168.0.105.45007 > dns.google.domain: udp/rtp 32 c98 + 256 65
12:39:53.732408 IP dns.google.domain > 192.168.0.105.45007: udp/rtp 102 c98 + 34179
5 packets captured
7 packets received by filter
0 packets dropped by kernel
```

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
```

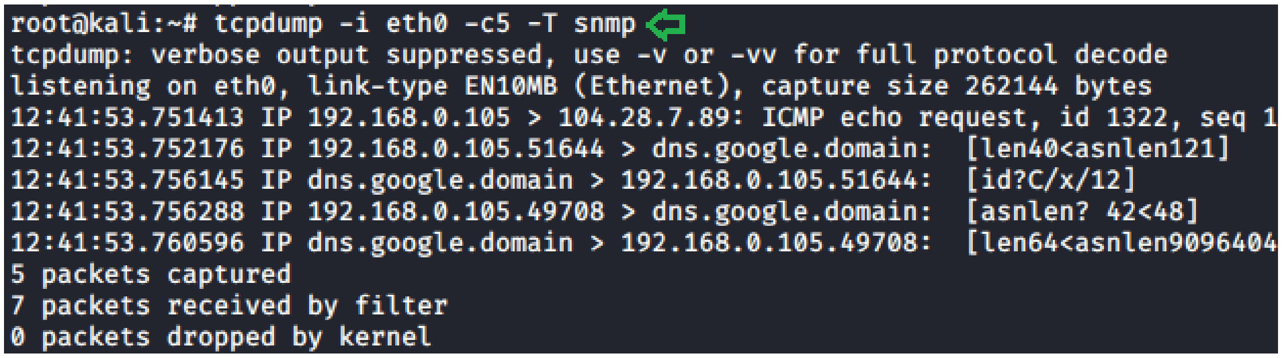


```
root@kali:~# tcpdump -i eth0 -c5 -T rtcp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:40:49.171715 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq
12:40:49.172472 IP 192.168.0.105.36642 > dns.google.domain:  type-0×93 1028
12:40:49.176281 IP dns.google.domain > 192.168.0.105.36642:  type-0×93 132624
12:40:49.176485 IP 192.168.0.105.46353 > dns.google.domain:  type-0×bf 1028
12:40:49.180535 IP dns.google.domain > 192.168.0.105.46353:  type-0×bf 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
```

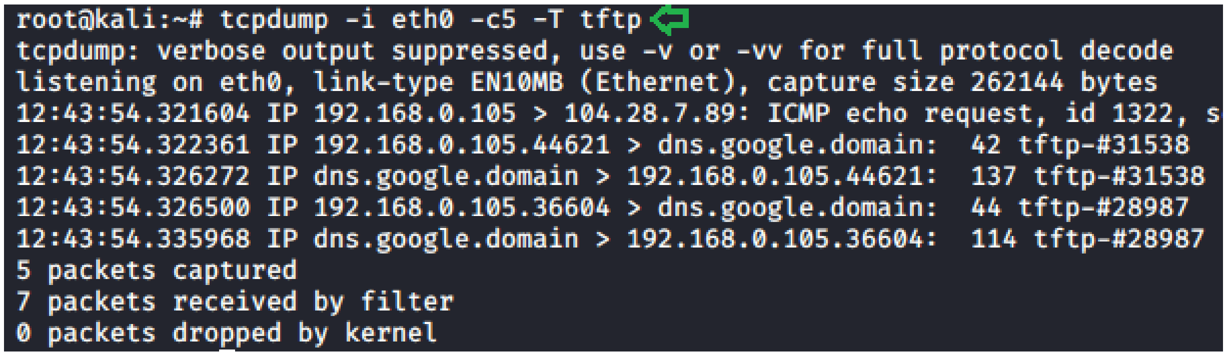


```
root@kali:~# tcpdump -i eth0 -c5 -T snmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:41:53.751413 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq 1
12:41:53.752176 IP 192.168.0.105.51644 > 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
```

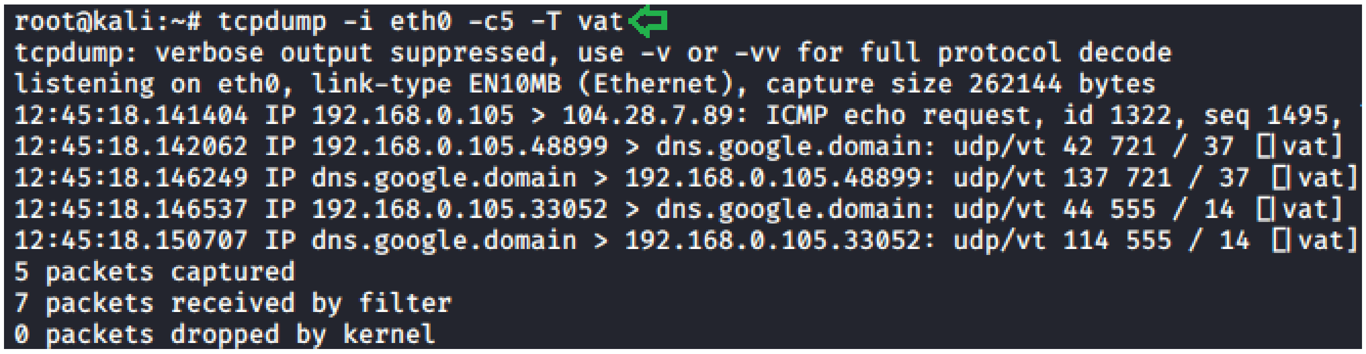


```
root@kali:~# tcpdump -i eth0 -c5 -T tftp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:43:54.321604 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, s
12:43:54.322361 IP 192.168.0.105.44621 > 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 a visual component. To understand its data packets in our traffic we need to apply these commands.

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

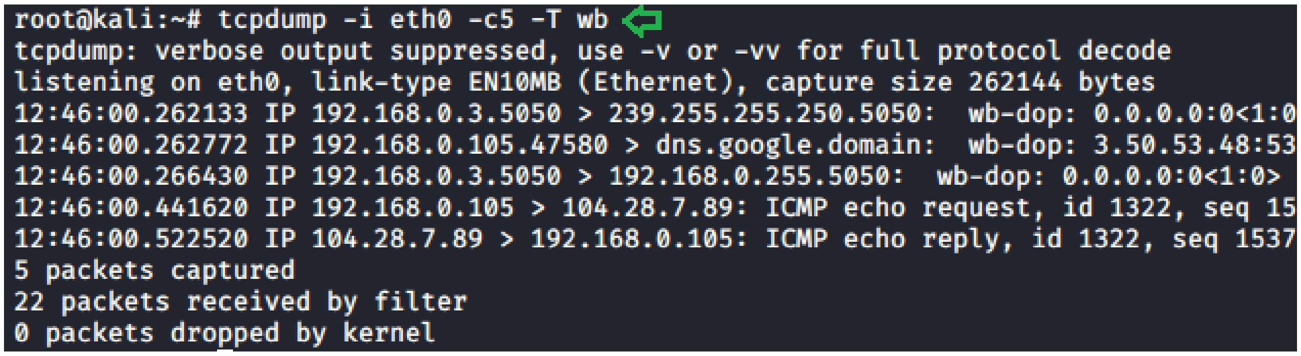


```
root@kali:~# tcpdump -i eth0 -c5 -T vat
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:45:18.141404 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1322, seq 1495,
12:45:18.142062 IP 192.168.0.105.48899 > 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

Distributed whiteboard, the program allows its users to draw and type the messages onto canvas, this should be synchronized to every other user that is on the same overlay network for the applications. New users should also receive everything that is already stored on the whiteboard when they connect. To understand its data packets, follow this command.

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

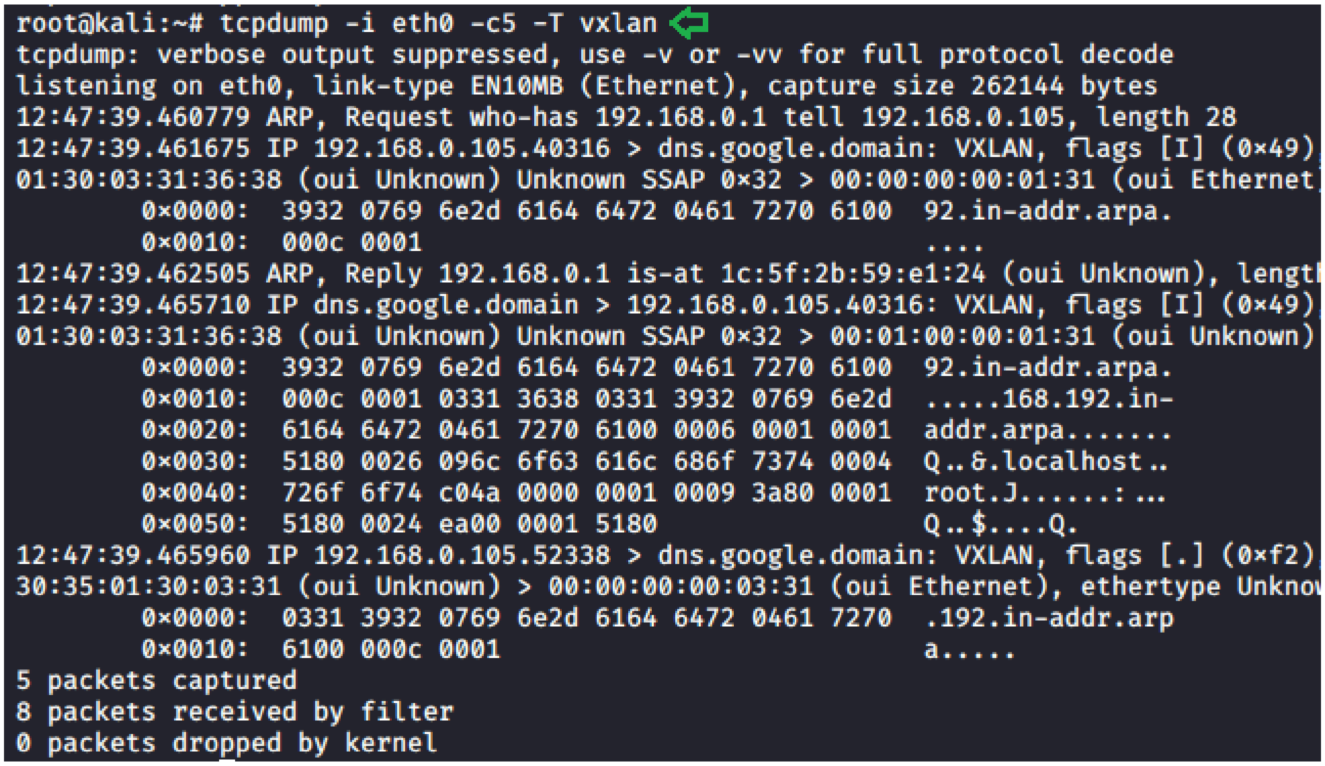


```
root@kali:~# tcpdump -i eth0 -c5 -T wb
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:46:00.262133 IP 192.168.0.3.5050 > 239.255.255.250.5050:  wb-dop: 0.0.0.0:0<1:0
12:46:00.262772 IP 192.168.0.105.47580 > 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 understands 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] (0×49)
01:30:03:31:36:38 (oui Unknown) Unknown SSAP 0×32 > 00:00:00:00:01:31 (oui Ethernet
        0×0000:  3932 0769 6e2d 6164 6472 0461 7270 6100  92.in-addr.arpa.
        0×0010:  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] (0×49)
01:30:03:31:36:38 (oui Unknown) Unknown SSAP 0×32 > 00:01:00:00:01:31 (oui Unknown)
        0×0000:  3932 0769 6e2d 6164 6472 0461 7270 6100  92.in-addr.arpa.
        0×0010:  000c 0001 0331 3638 0331 3932 0769 6e2d  .....168.192.in-
        0×0020:  6164 6472 0461 7270 6100 0006 0001 0001  addr.arpa.......
        0×0030:  5180 0026 096c 6f63 616c 686f 7374 0004  Q..&.localhost..
        0×0040:  726f 6f74 c04a 0000 0001 0009 3a80 0001  root.J......:...
        0×0050:  5180 0024 ea00 0001 5180                 Q..$....Q.
12:47:39.465960 IP 192.168.0.105.52338 > dns.google.domain: VXLAN, flags [.] (0×f2)
30:35:01:30:03:31 (oui Unknown) > 00:00:00:00:03:31 (oui Ethernet), ethertype Unknow
        0×0000:  0331 3932 0769 6e2d 6164 6472 0461 7270  .192.in-addr.arp
        0×0010:  6100 000c 0001                           a.....
5 packets captured
8 packets received by filter
0 packets dropped by kernel
```

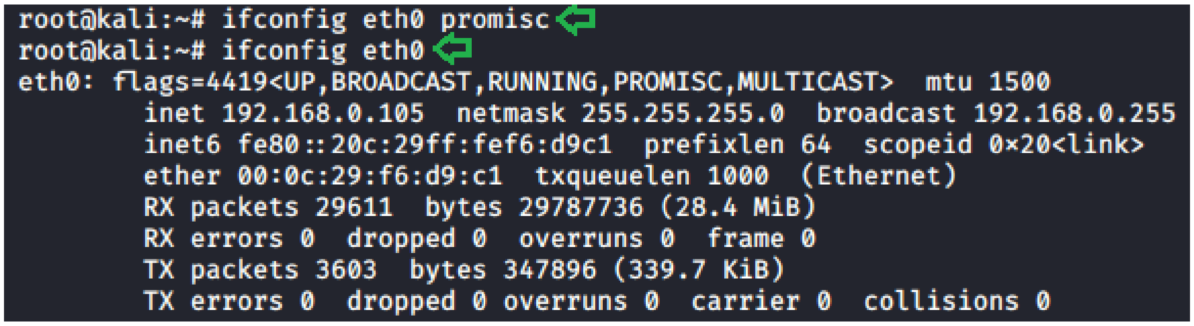
These are some of the protocol which is used under forced packets parameter to get the fixed desired data traffic from scan.

Promiscuous Mode

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

To configure promiscuous mode by following these commands.

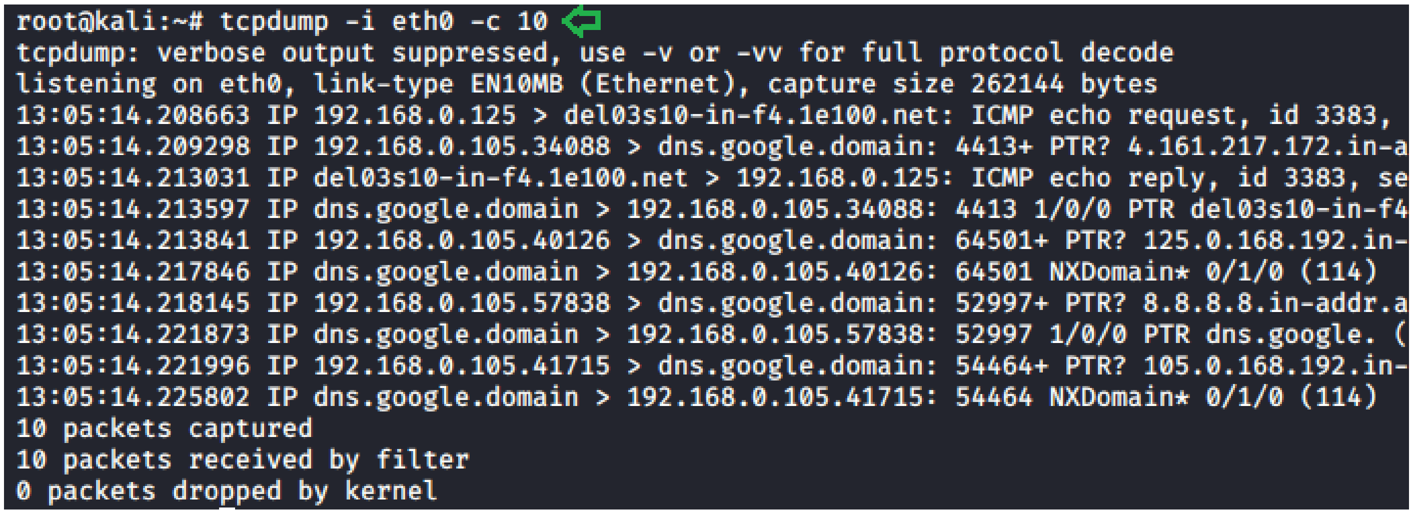
```
ifconfig eth0 promisc
ifconfig eth0
```



```
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
```

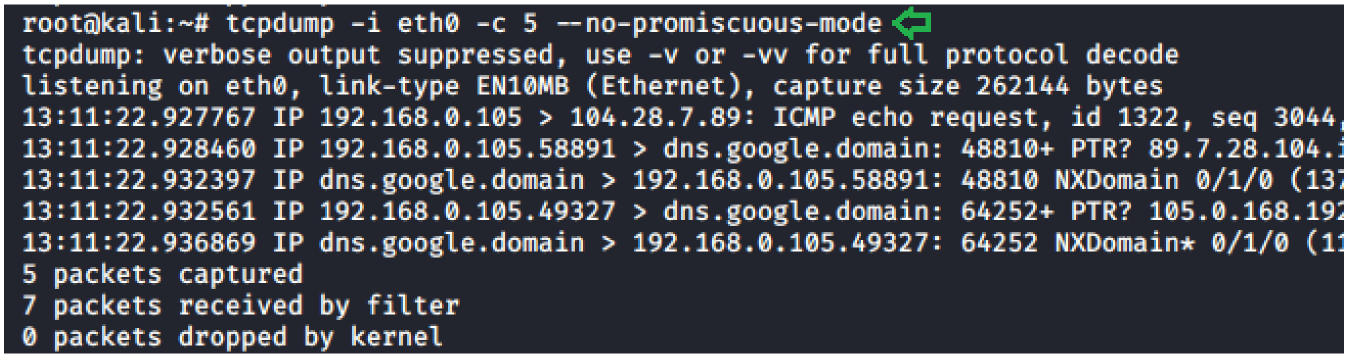


```
root@kali:~# tcpdump -i eth0 -c 10
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:05:14.208663 IP 192.168.0.125 > del03s10-in-f4.1e100.net: ICMP echo request, id 3383,
13:05:14.209298 IP 192.168.0.105.34088 > 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 that means a network interface card will pass all frames received to the OS for processing versus the traditional operation where only frames destined for the NIC's MAC address or a broadcast address will be passed up to the OS. Generally, promiscuous mode is used to "sniff" all traffic on the wire. But if we want to switch to multicast mode against the 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
```

This is the second part of the series. So, get familiar with these features and stay tuned for some advance features of tcpdump in our next article.