POST CATEGORY : Penetration Testing

Comprehensive Guide to tcpdump (Part 2)

posted in PENETRATION TESTING on MARCH 19, 2020 by RAJ CHANDEL with 1 COMMENT

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
- Parsing and printing
- User scan
- Timestamp precision
- Force packets
    - RADIUS (Remote Authentication Dial-in User Service)
    - AODV (Ad-hoc On-demand Distance Vector protocol)
    - RPC (Remote Procedure Call)
    - CNFP (Cisco NetFlow Protocol)
    - LMP (Link Management Protocol)
    - PGM (Pragmatic General Multicast)
    - RTP (Real-Time Application Protocol)
    - RTCP (Real-Time Application Control Protocol)
    - SNMP (Simple Network Management Protocol)
    - TFTP (Trivial File Transfer Protocol)
    - VAT (Visual Audio Tool)
    - WB (Distributed White Board)
    - VXLAN (Virtual Xtensible Local Area Network)
- Promiscuous mode
- No promiscuous mode

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
```

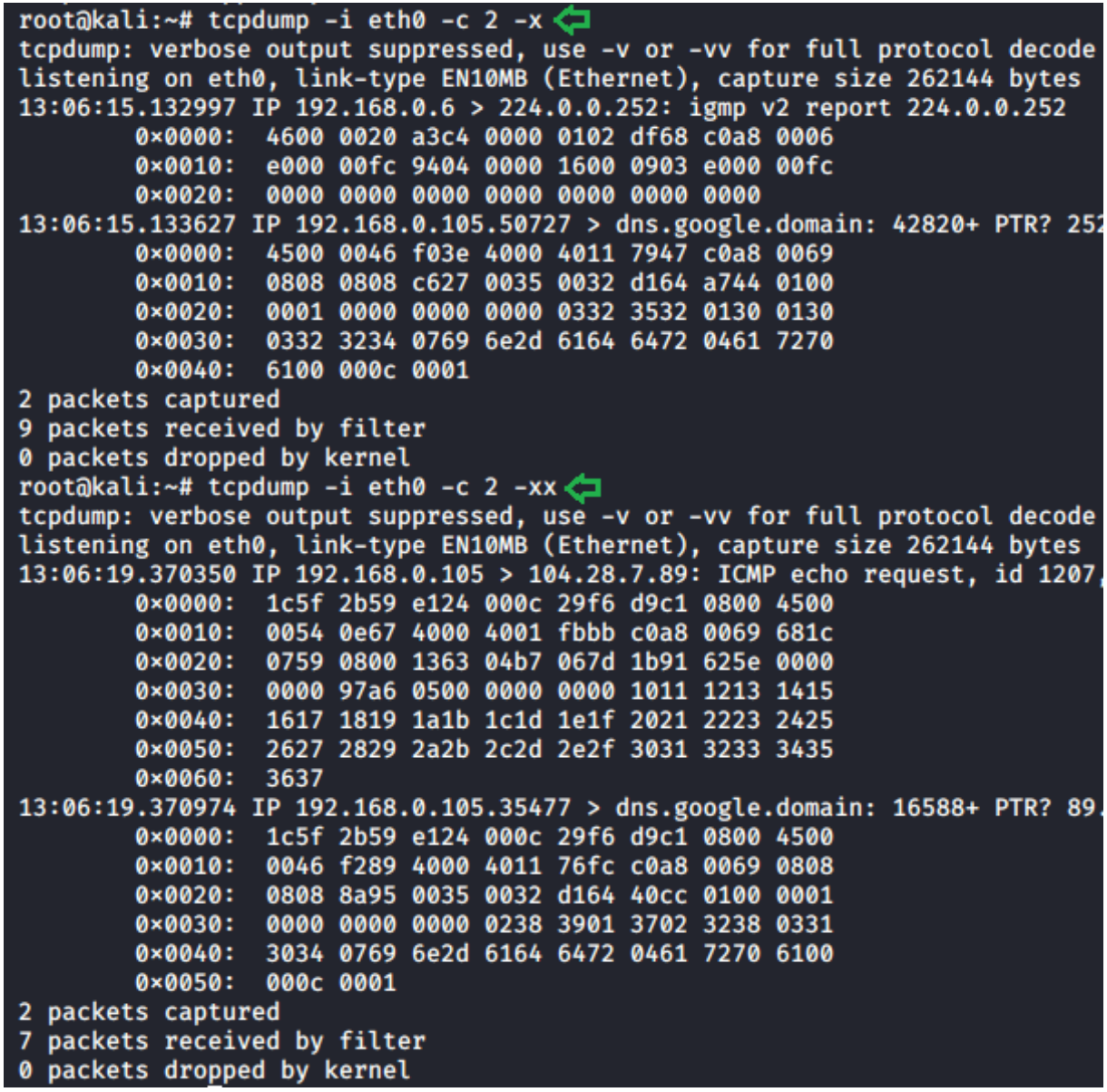
Parsing and Printing

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
```


Categories

- BackTrack 5 Tutorials
- Cryptography & Stegnography
- CTF Challenges
- Cyber Forensics
- Database Hacking
- Footprinting
- Hacking Tools
- Kali Linux
- Nmap
- Others
- Penetration Testing



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

- Privilege Escalation
- Red Teaming
- Social Engineering Toolkit
- Trojans & Backdoors
- Uncategorized
- Website Hacking
- Window Password Hacking
- Wireless Hacking

Articles

Select Month

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
```

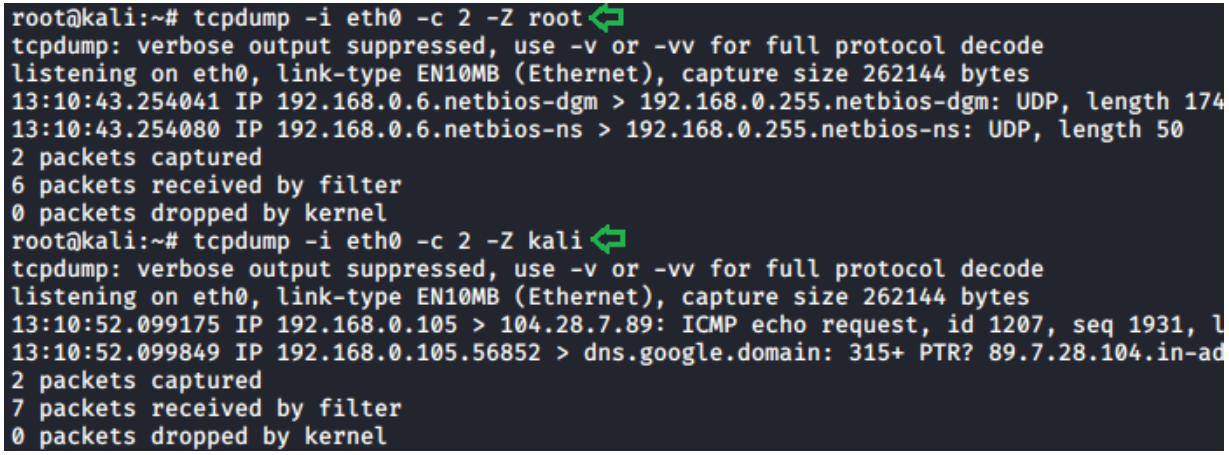
```
root@kali:~# tcpdump -i eth0 -c 2 -X
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:07:52.141821 IP 192.168.0.105 > 104.28.7.89: ICMP echo request, id 1207,
        0x0000:  4500 0054 3b1c 4000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

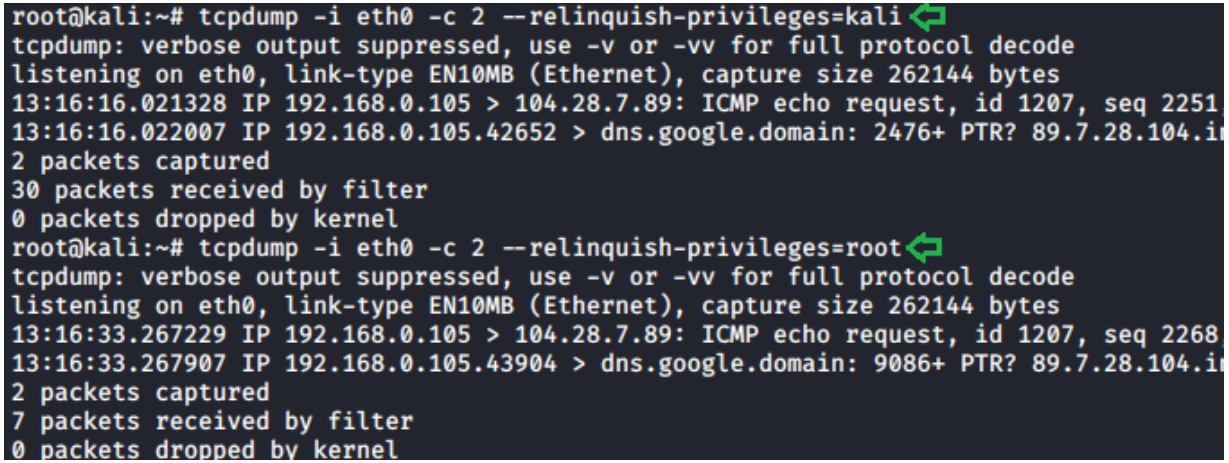
If we are running tcpdump as root then before opening any saved file for analysis, you will observe that it changes the user ID to the user and the group IDs to the primary group of its users.

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
```



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



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.

```
1 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.

```
1 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.

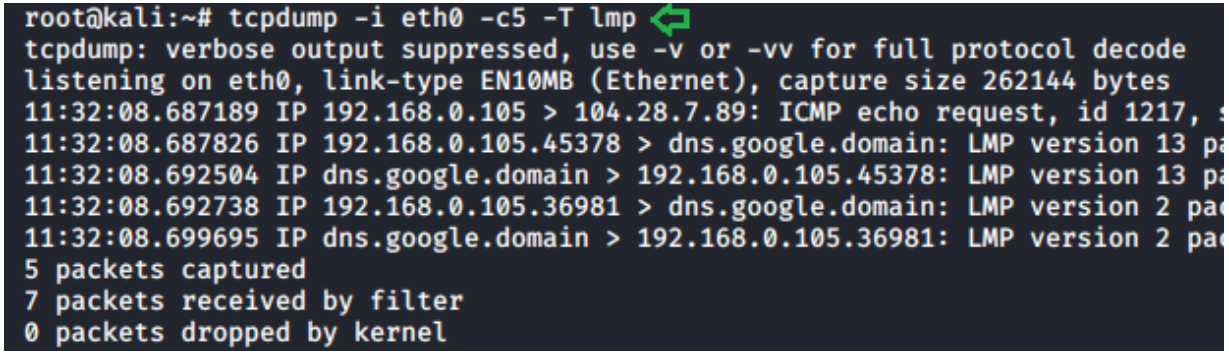
```
tcpdump -i eth0 -c5 -T cnfp
```

```
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.

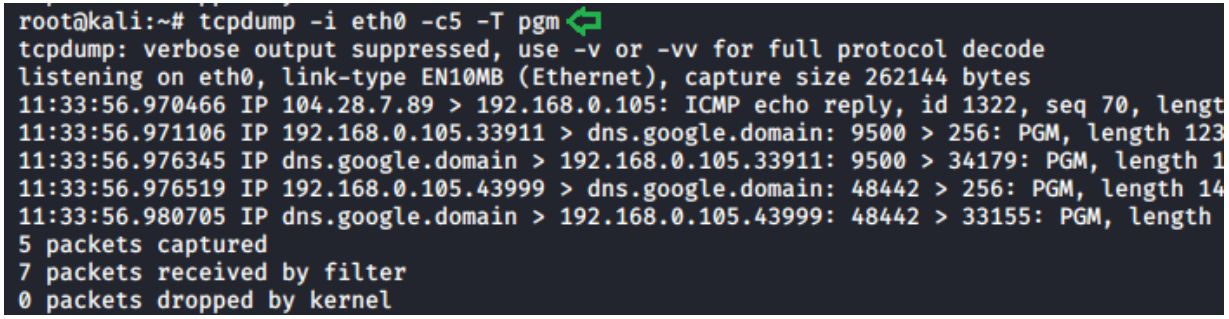
```
tcpdump -i eth0 -c5 -T lmp
```



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.

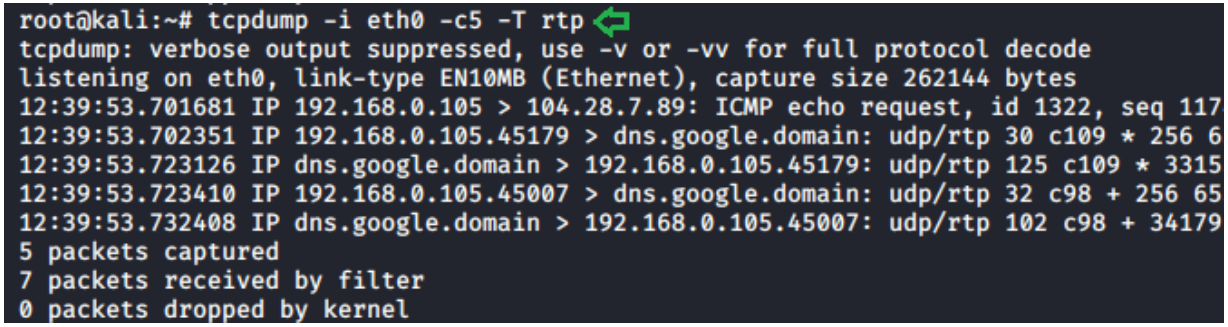
```
tcpdump -i eth0 -c5 -T pgm
```



RTP

Real-time application protocol, 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
```



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.

```
1 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] (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 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.

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

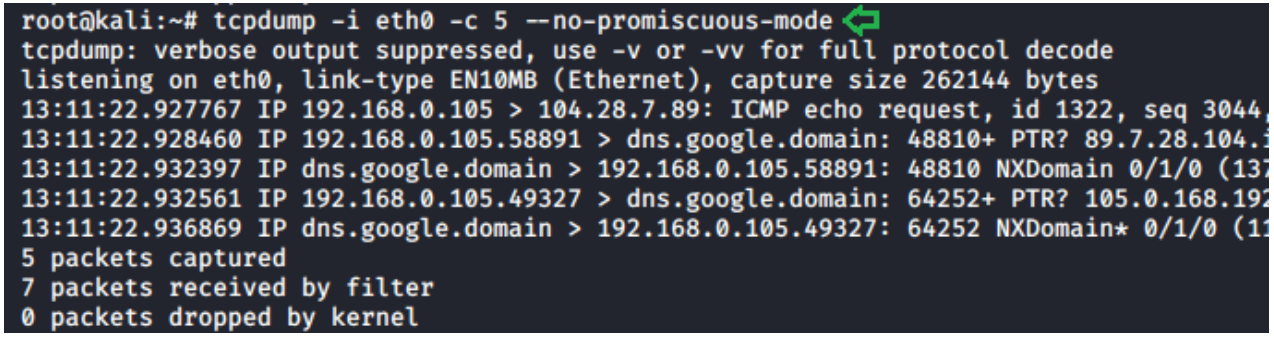
```
root@kali:~# tcpdump -i eth0 -c 10
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
13:05:14.208663 IP 192.168.0.125 > del03s10-in-f4.1e100.net: ICMP echo request, id 3383,
13:05:14.209298 IP 192.168.0.105.34088 > 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
```



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.

Author: Shubham Sharma is a Pentester, Cybersecurity Researcher, Contact Linkedin and twitter.

Comprehensive Guide to tcpdump (Part 1)

posted in PENETRATION TESTING on MARCH 19, 2020 by RAJ CHANDEL with 0 COMMENT

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

Table of Content

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

Introduction

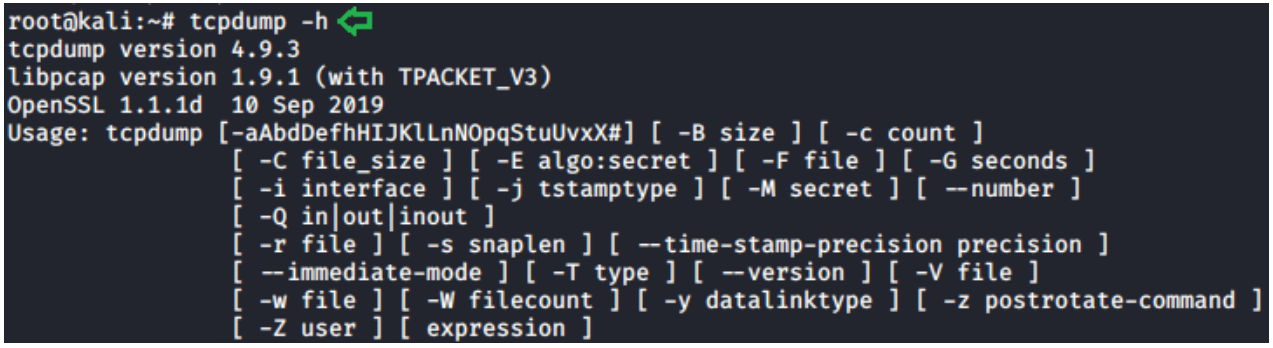
Tcpdump was originally developed in 1988 by Van Jacobson, Sally Floyd, Vern Paxson, and Steven McCanne. They worked at the Lawrence Berkeley Laboratory Network Research Group.

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

Available Options

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

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

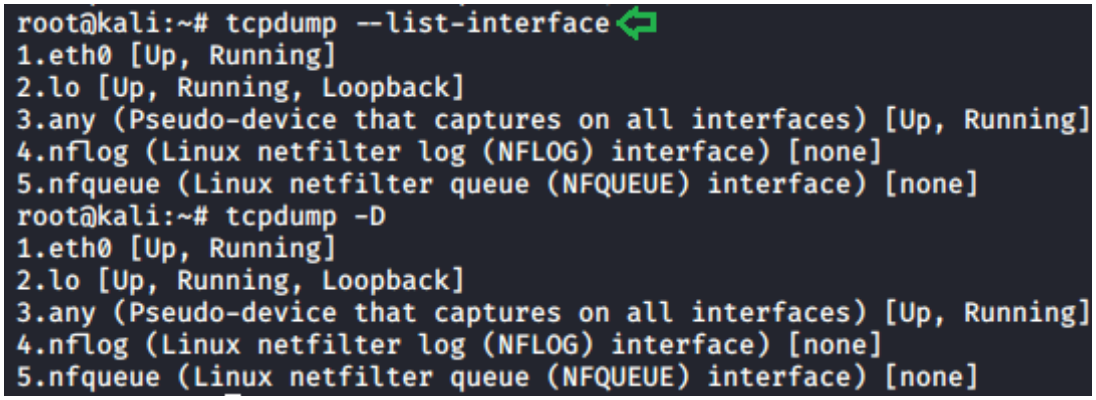


List of interfaces

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

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

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



Default Capture

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

```
tcpdump
```

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

Capturing traffic of a particular interface

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

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

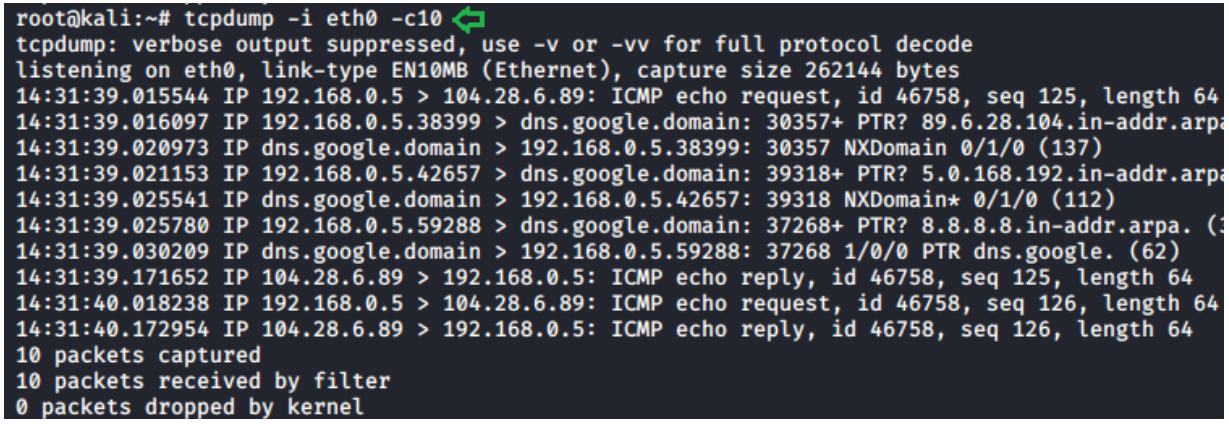
```
1 tcpdump -i eth0
```

```
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
```



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.

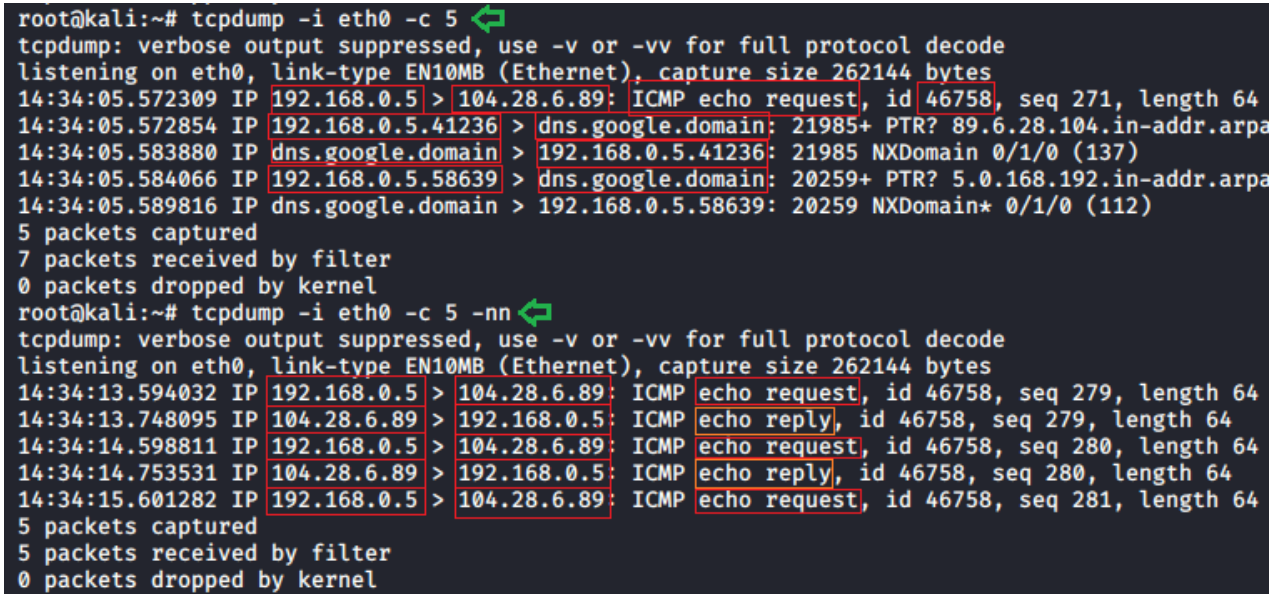
```
1 tcpdump -i eth0 -c 5 -A
```

```
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.

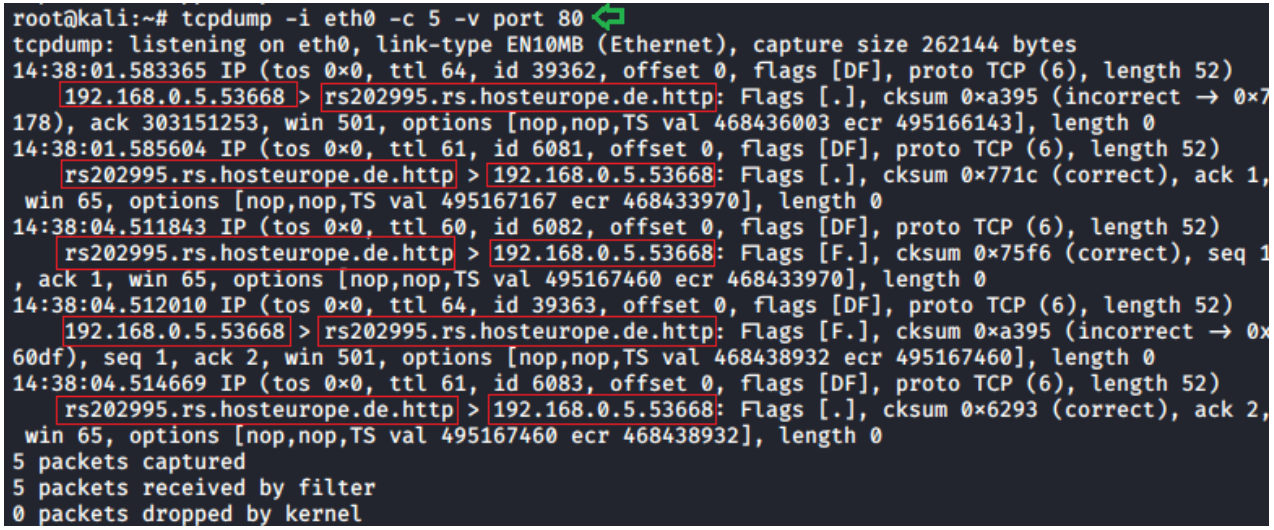
```
tcpdump -i eth0 -c 5
tcpdump -i eth0 -c 5 -nn
```



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.

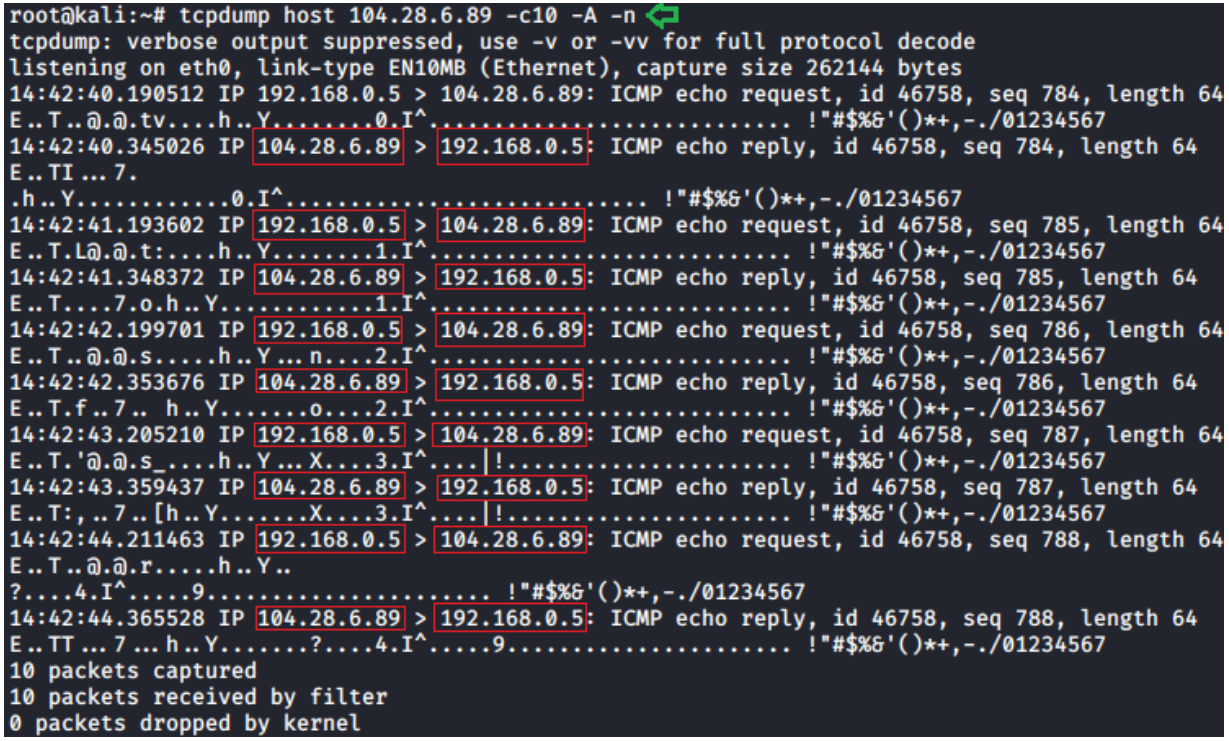
```
tcpdump -i eth0 -c 5 -v port 80
```



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.

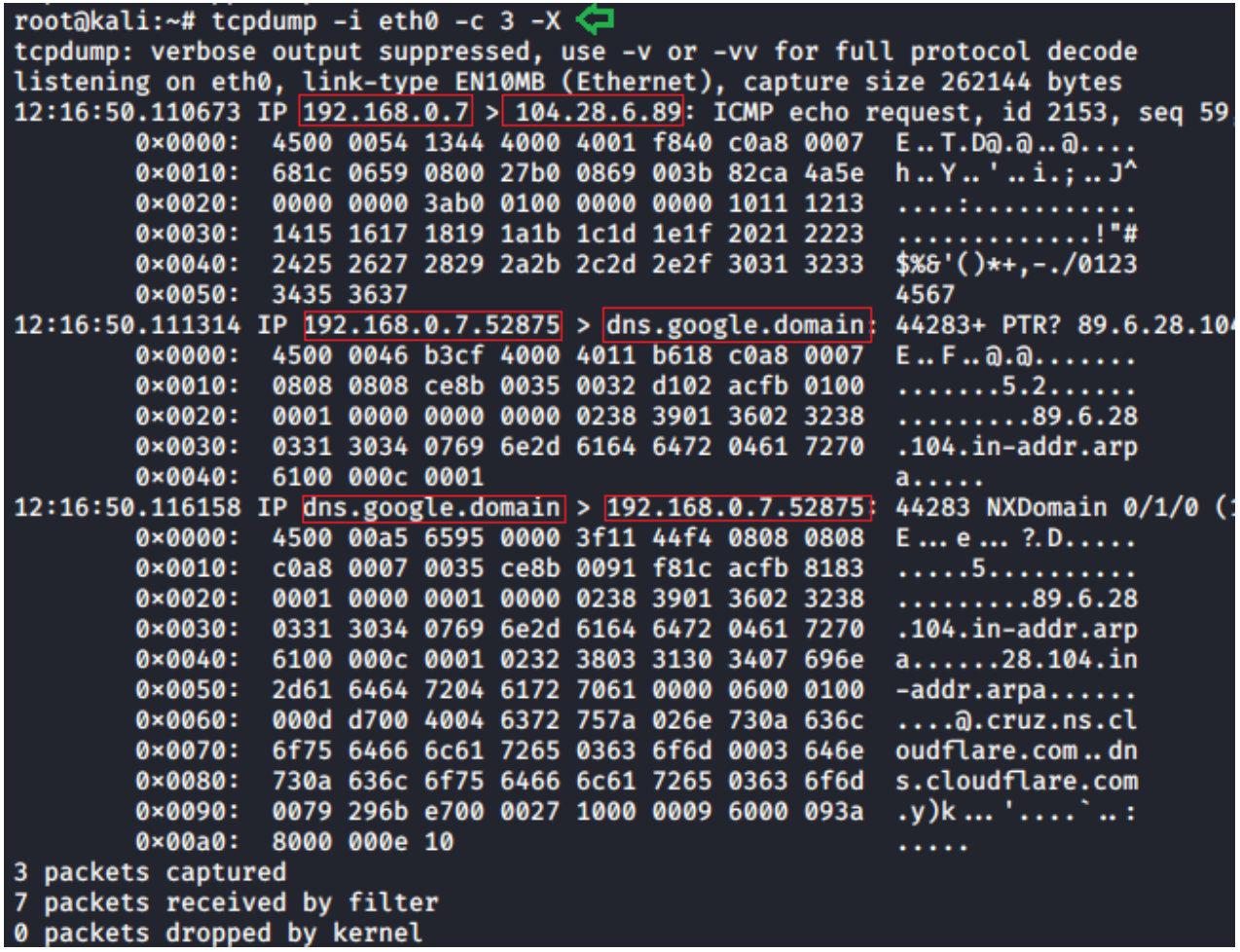
```
1 tcpdump host 104.28.6.89 -c10 -A -n
```



The header of each packet

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

```
1 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.

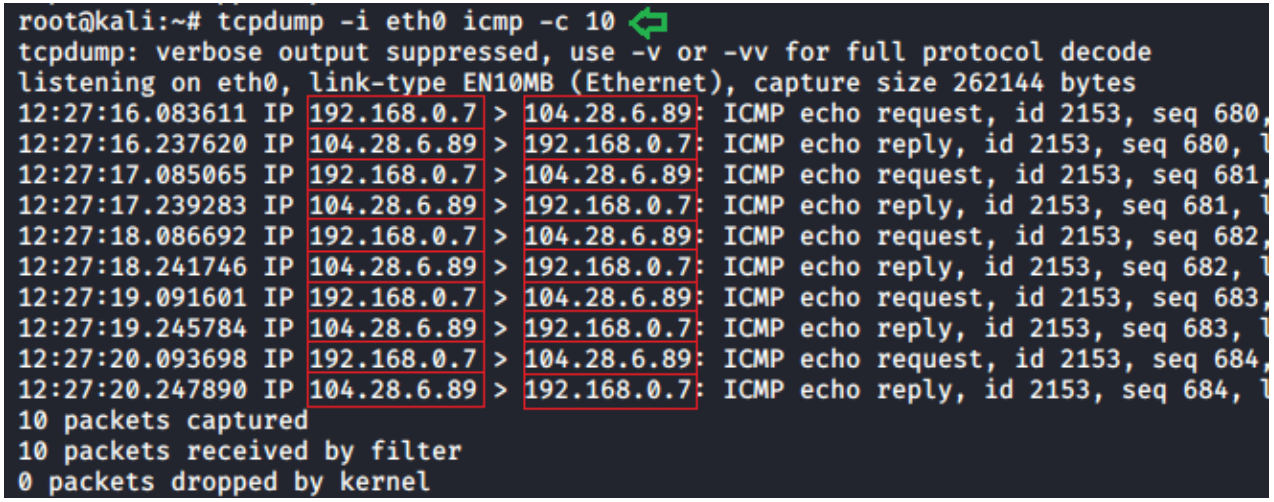
```
tcpdump -i eth0 -nnXS
```

```
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.

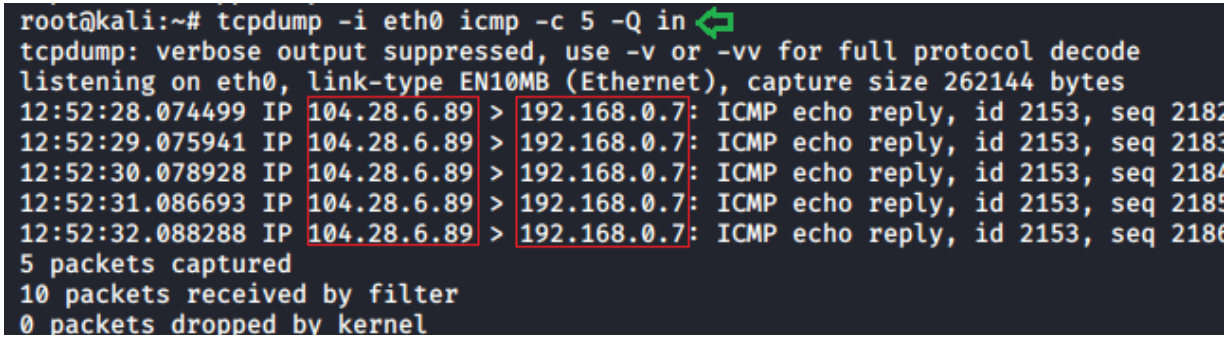
```
1 | tcpdump -i eth0 icmp -c 10
```



Packet directions

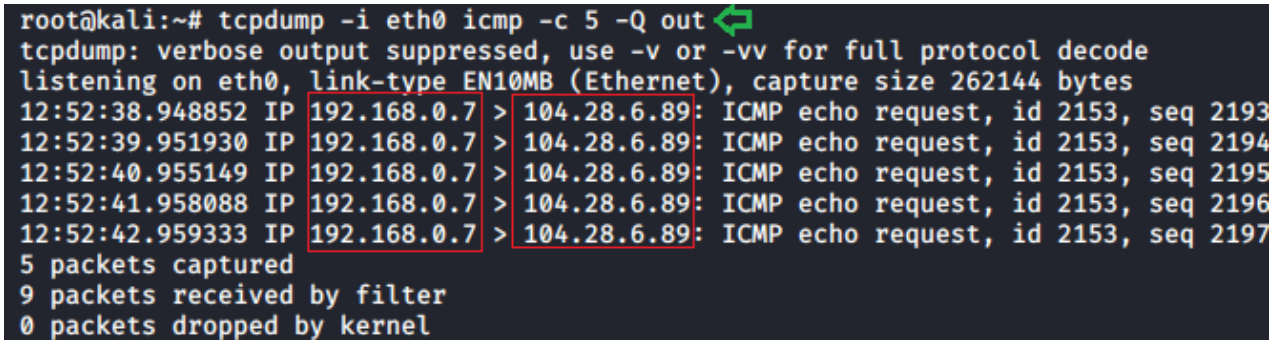
To the direction of data flow in our traffic, we can use the following parameter :

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



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
```



Live number count

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

Read and write in a file

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

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

To read this .pcap file we follow:

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

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

Snapshot length

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

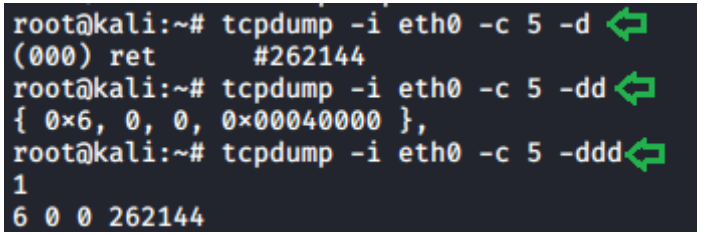
```
1 tcpdump -i eth0 icmp -s10 -c2
2 tcpdump -i eth0 icmp -s25 -c2
3 tcpdump -i eth0 icmp -s40 -c2
4 tcpdump -i eth0 icmp -s45 -c2
```

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

Dump mode

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

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



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

Author: Shubham Sharma is a Pentester and a Cybersecurity Researcher, contact LinkedIn and Twitter.

Beginners Guide to TShark (Part 3)

posted in PENETRATION TESTING on FEBRUARY 28, 2020 by RAJ CHANDEL with 1 COMMENT

This is the third instalment in the Beginners Guide to TShark Series. Please find the first and second instalments below.

- Beginners Guide to TShark (Part 1)
- Beginners Guide to TShark (Part 2)

TL; DR

In this part, we will understand the reporting functionalities and some additional tricks that we found while tinkering with TShark.

Table of Content

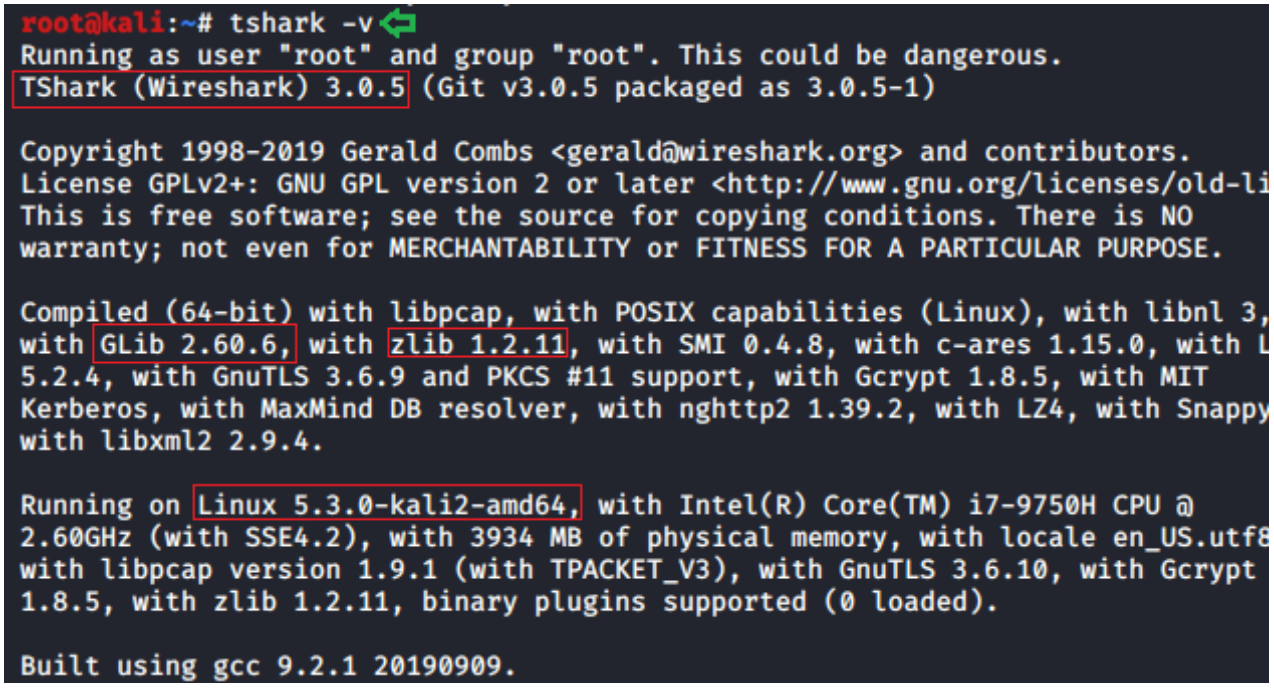
- **Version Information**
- **Reporting Options**
  - Column Formats
  - Decodes
  - Dissector Tables
  - Elastic Mapping
  - Field Count
  - Fields
  - Fundamental Types
  - Heuristic Decodes
  - Plugins
  - Protocols
  - Values
  - Preferences
  - Folders
- **PyShark**
  - Installation
  - Live Capture
  - Pretty Representation
  - Captured Length Field
  - Layers, Src and Dst Fields

- Promisc Capture

Version Information

Let's begin with the very simple command so that we can understand and correlate that all the practicals performed during this article and the previous articles are of the version depicted in the image given below. This parameter prints the Version information of the installed TShark.

```
1 tshark -v
```



Reporting Options

During any Network capture or investigation, there is a dire need of the reports so that we can share the findings with the team as well as superiors and have a

validated proof of any activity inside the network. For the same reasons, TShark has given us a beautiful option (**-G**). This option will make the TShark print a list of several types of reports that can be generated. Official Manual of TShark used the word Glossaries for describing the types of reports.

```
tshark -G help
```

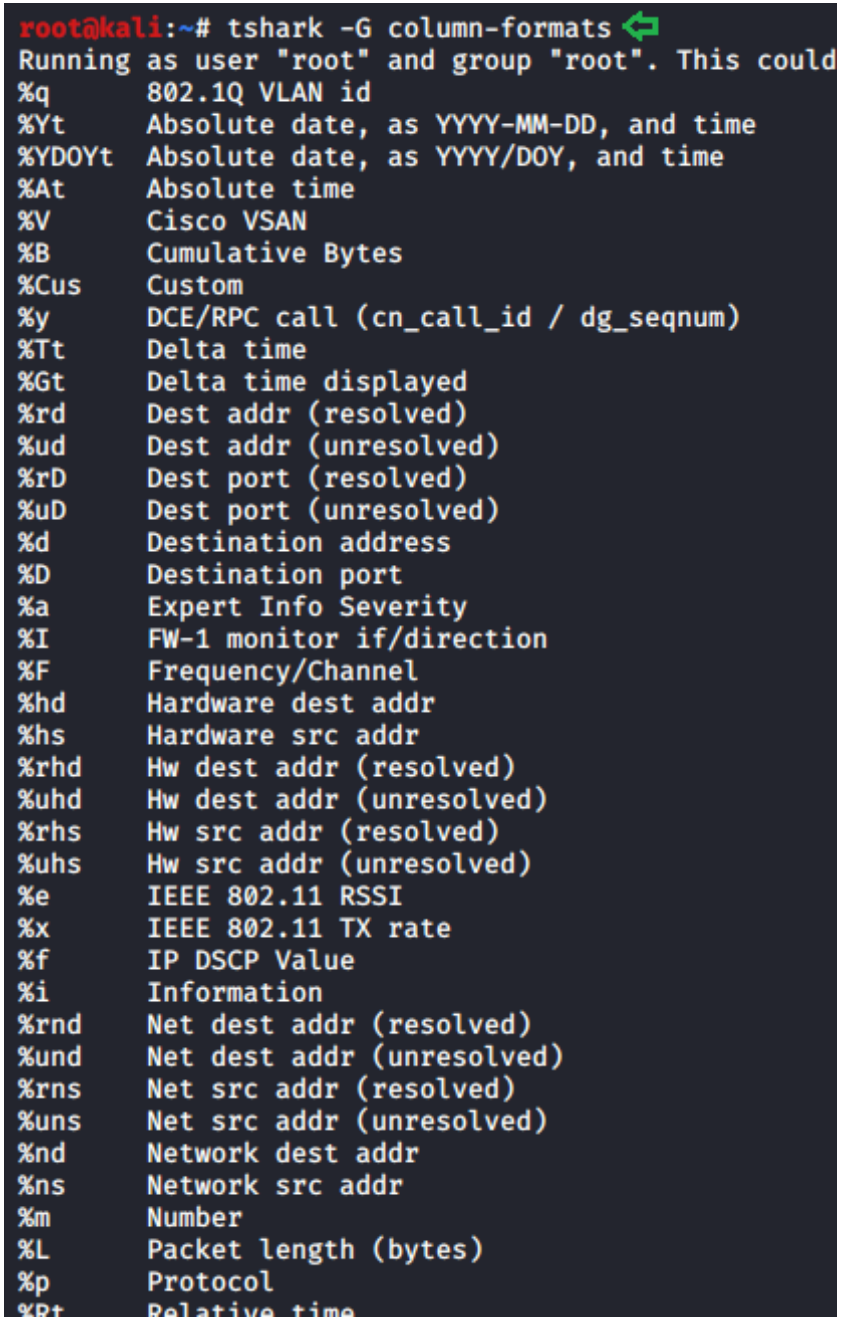
```
root@kali:~# tshark -G help
Running as user "root" and group "root". This could be dangerous.
TShark (Wireshark) 3.0.5 (Git v3.0.5 packaged as 3.0.5-1)

Usage: tshark -G [report]

Glossary table reports:
  -G column-formats        dump column format codes and exit
  -G decodes               dump "layer type"/"decode as" associations and exit
  -G dissector-tables      dump dissector table names, types, and properties
  -G elastic-mapping       dump ElasticSearch mapping file
  -G fieldcount            dump count of header fields and exit
  -G fields                dump fields glossary and exit
  -G ftypes                dump field type basic and descriptive names
  -G heuristic-decodes     dump heuristic dissector tables
  -G plugins               dump installed plugins and exit
  -G protocols             dump protocols in registration database and exit
  -G values                dump value, range, true/false strings and exit

Preference reports:
  -G currentprefs          dump current preferences and exit
  -G defaultprefs          dump default preferences and exit
  -G folders               dump about:folders
```

Column Formats

From our previous practicals, we saw that we have the Column Formats option available in the reporting section of TShark. To explore its contents, we ran the command as shown in the image given below. We see that it prints a list of

wildcards that could be used while generating a report. We have the VLAN id, Date, Time, Destination Address, Destination Port, Packet Length, Protocol, etc.

```
tshark -G column-formats
```



Decodes

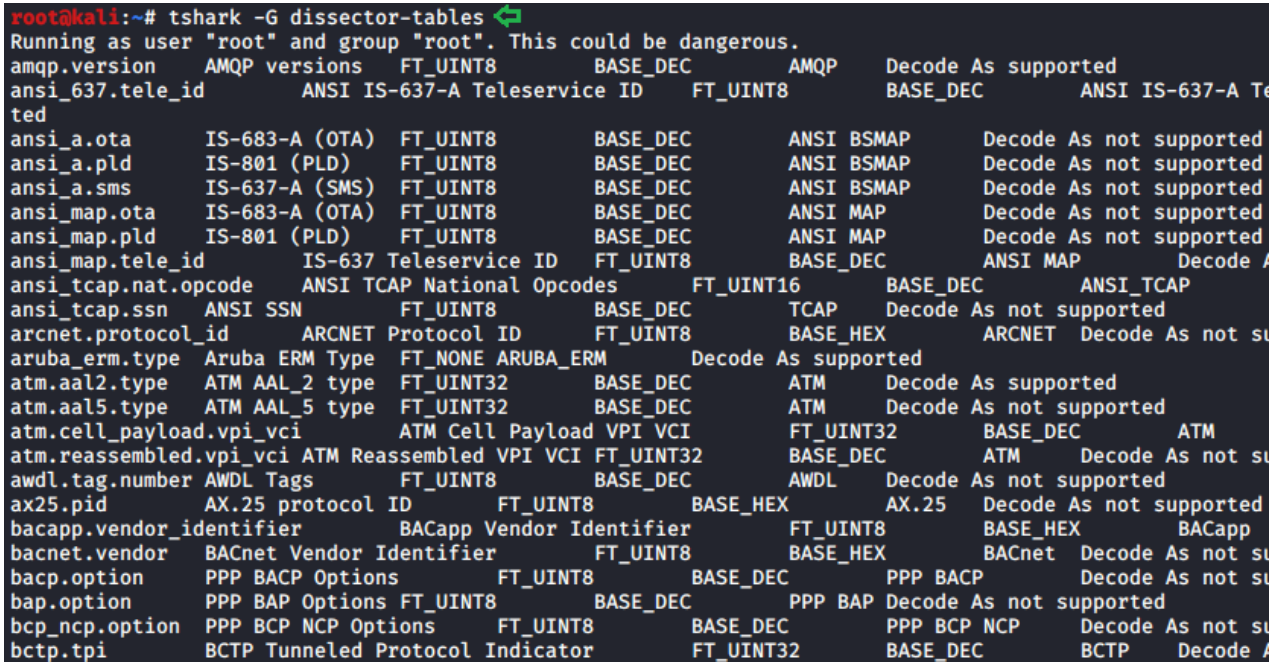
This option generates 3 Fields related to Layers as well as the protocol decoded. There is a restriction enforced for one record per line with this option. The first field that has the “s1ap.proc.sout” tells us the layer type of the network packets. Followed by that we have the value of selector in decimal format. At last, we have the decoding that was performed on the capture. We used the head command as the output was rather big to fit in the screenshot.

```
1 tshark -G decodes | head
```

```
root@kali:~# tshark -G decodes | head
Running as user "root" and group "root". This could be dangerous.
s1ap.proc.sout  17      s1ap
s1ap.proc.sout  3       s1ap
s1ap.proc.sout  6       s1ap
s1ap.proc.sout  23      s1ap
s1ap.proc.sout  9       s1ap
s1ap.proc.sout  48      s1ap
s1ap.proc.sout  43      s1ap
s1ap.proc.sout  29      s1ap
s1ap.proc.sout  4       s1ap
s1ap.proc.sout  21      s1ap
```

Dissector Tables

Most of the users reading this article are already familiar with the concept of Dissector. If not, in simple words Dissector is simply a protocol parser. The output generated by this option consists of 6 fields. Starting from the Dissector Table Name then the name is used for the dissector table in the GUI format. Next, we have the type and the base for the display and the Protocol Name. Lastly, we have the decode as a format.



```
root@kali:~# tshark -G dissector-tables
Running as user "root" and group "root". This could be dangerous.
amqp.version    AMQP versions   FT_UINT8        BASE_DEC        AMQP    Decode As supported
ansi_637.tele_id        ANSI IS-637-A Teleservice ID    FT_UINT8        BASE_DEC        ANSI IS-637-A Te
ted
ansi_a.ota      IS-683-A (OTA)  FT_UINT8        BASE_DEC        ANSI BSMAP      Decode As not supported
ansi_a.pld      IS-801 (PLD)    FT_UINT8        BASE_DEC        ANSI BSMAP      Decode As not supported
ansi_a.sms      IS-637-A (SMS)  FT_UINT8        BASE_DEC        ANSI BSMAP      Decode As not supported
ansi_map.ota    IS-683-A (OTA)  FT_UINT8        BASE_DEC        ANSI MAP        Decode As not supported
ansi_map.pld    IS-801 (PLD)    FT_UINT8        BASE_DEC        ANSI MAP        Decode As not supported
ansi_map.tele_id        IS-637 Teleservice ID   FT_UINT8        BASE_DEC        ANSI MAP        Decode A
ansi_tcap.nat.opcode    ANSI TCAP National Opcodes      FT_UINT16       BASE_DEC        ANSI_TCAP
ansi_tcap.ssn   ANSI SSN        FT_UINT8        BASE_DEC        TCAP    Decode As not supported
arcnet.protocol_id      ARCNET Protocol ID      FT_UINT8        BASE_HEX        ARCNET  Decode As not su
aruba_erm.type  Aruba ERM Type  FT_NONE ARUBA_ERM       Decode As supported
atm.aal2.type   ATM AAL_2 type  FT_UINT32       BASE_DEC        ATM     Decode As supported
atm.aal5.type   ATM AAL_5 type  FT_UINT32       BASE_DEC        ATM     Decode As not supported
atm.cell_payload.vpi_vci        ATM Cell Payload VPI VCI        FT_UINT32       BASE_DEC        ATM
atm.reassembled.vpi_vci ATM Reassembled VPI VCI FT_UINT32       BASE_DEC        ATM     Decode As not su
awdl.tag.number AWDL Tags       FT_UINT8        BASE_DEC        AWDL    Decode As not supported
ax25.pid        AX.25 protocol ID       FT_UINT8        BASE_HEX        AX.25   Decode As not supported
bacapp.vendor_identifier        BACapp Vendor Identifier        FT_UINT8        BASE_HEX        BACapp
bacnet.vendor   BACnet Vendor Identifier        FT_UINT8        BASE_HEX        BACnet  Decode As not su
bacp.option     PPP BACP Options        FT_UINT8        BASE_DEC        PPP BACP        Decode As not su
bap.option      PPP BAP Options FT_UINT8        BASE_DEC        PPP BAP Decode As not supported
bcp_ncp.option  PPP BCP NCP Options     FT_UINT8        BASE_DEC        PPP BCP NCP     Decode As not su
bctp.tpi        BCTP Tunneled Protocol Indicator        FT_UINT32       BASE_DEC        BCTP    Decode A
```

Elastic Mapping

Mapping is the outline of the documents stored in the index. Elasticsearch supports different data types for the fields in a document. The elastic-mapping option of the TShark prints out the data stored inside the ElasticSearch mapping file. Due to a large amount of data getting printed, we decided to use the head command as well.

```
1 tshark -G elastic-mapping | head
```

```
root@kali:~# tshark -G elastic-mapping | head
Running as user "root" and group "root". This could be dangerous.
{
  "template": "packets-*",
  "settings": {
    "index.mapping.total_fields.limit": 1000000
  },
  "mappings": {
    "pcap_file": {
      "dynamic": false,
      "properties": {
        "timestamp": {
```

Field Count

There are times in a network trace, where we need to get the count of the header fields travelling at any moment. In such scenarios, TShark got our back. With the fieldcount option, we can print the number of header fields with ease. As we can observe in the image given below that we have 2522 protocols and 215000 fields were pre-allocated.

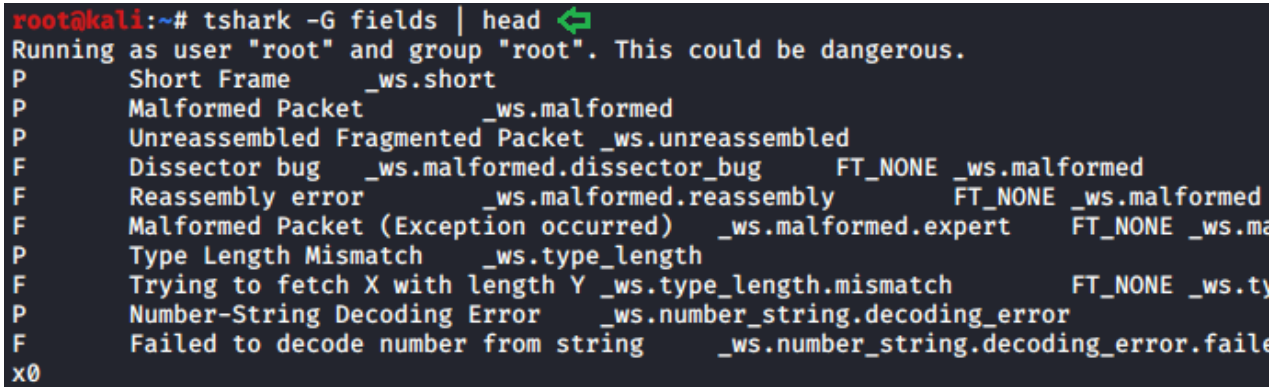
```
1 tshark -G fieldcount
```

```
root@kali:~# tshark -G fieldcount
Running as user "root" and group "root". This could be dangerous
There are 214494 header fields registered, of which:
        0 are deregistered
        2522 are protocols
        16070 have the same name as another field

215000 fields were pre-allocated.

The header field table consumes 1679 KiB of memory.
The fields themselves consume 15081 KiB of memory.
```

Fields

TShark can also get us the contents of the registration database. The output generated by this option is not as easy to interpret as the others. For some users, they can use any other parsing tool for generating a better output. Each record in the output is a protocol or a header file. This can be differentiated by the First field of the record. If the Field is P then it is a Protocol and if it is F then it's a header field. In the case of the Protocols, we have 2 more fields. One tells us about the Protocol and other fields show the abbreviation used for the said protocol. In the case of Header, the facts are a little different. We have 7 more fields. We have the Descriptive Name, Abbreviation, Type, Parent Protocol Abbreviation, Base for Display, Bitmask, Blurb Describing Field, etc.

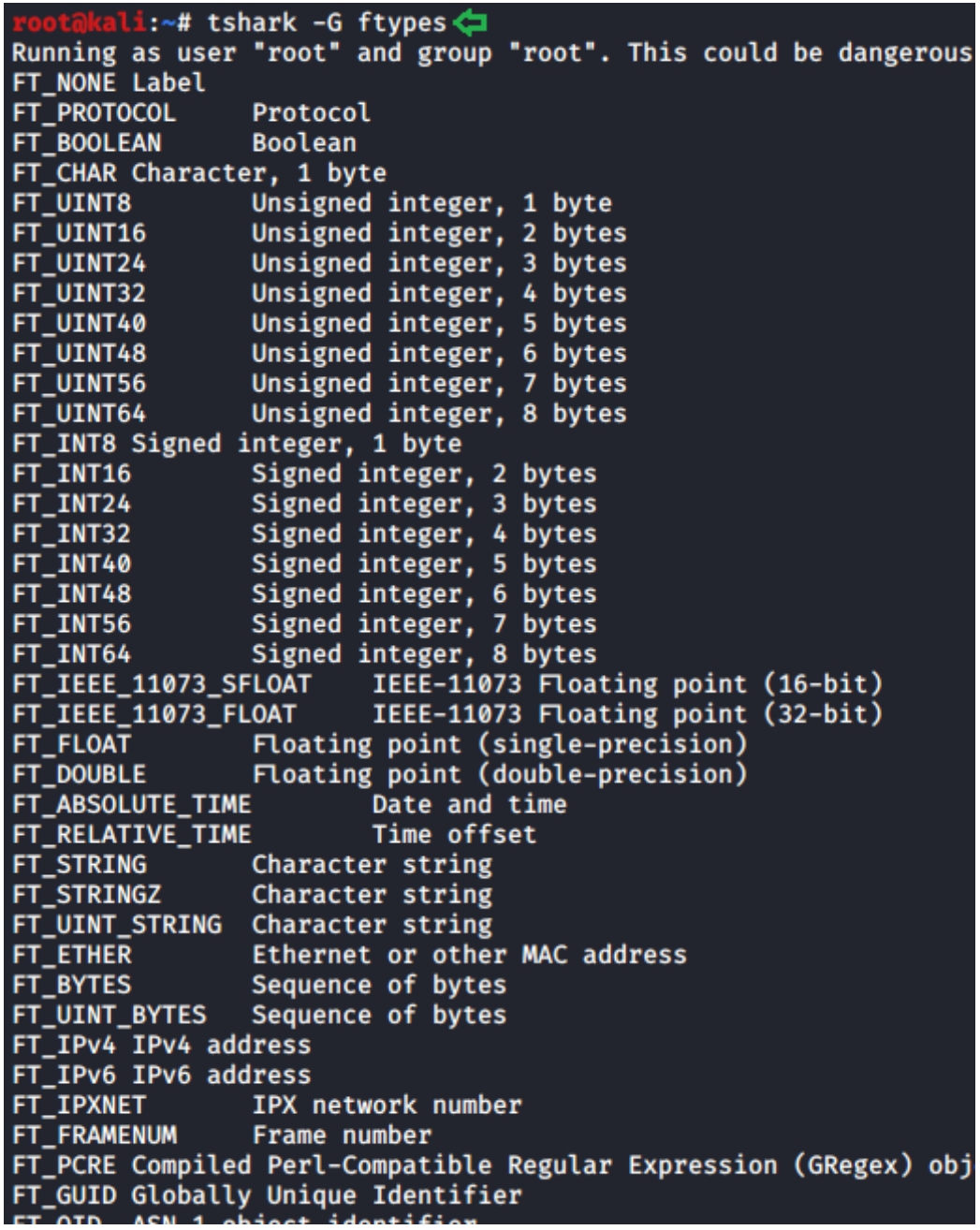
```
1 tshark -G fields | head
```



Fundamental Types

TShark also helps us generate a report centralized around the fundamental types of network protocol. This is abbreviated as ftype. This type of report consists of only 2 fields. One for the FTYPE and other for its description.

```
1 tshark -G ftypes
```



Heuristic Decodes

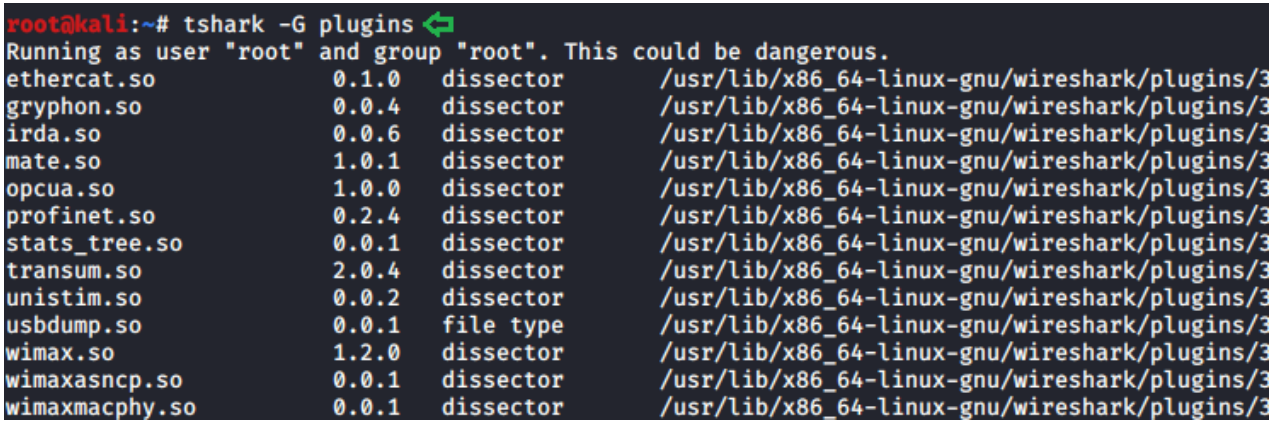
Sorting the Dissectors based on the heuristic decodes is one of the things that need to be easily and readily available. For the same reason, we have the option of heuristic decodes in TShark. This option prints all the heuristic decodes which are currently installed. It consists of 3 fields. First, one representing the underlying dissector, the second one representing the name of the heuristic decoded and the last one tells about the status of the heuristic. It will be T in case it is heuristics and F otherwise.

```
1 tshark -G heuristic-decodes
```

Plugins

Plugins are a very important kind of option that was integrated with Tshark Reporting options. As the name states it prints the name of all the plugins that are installed. The field that this report consists of is made of the Plugin Library, Plugin Version, Plugin Type and the path where the plugin is located.

```
tshark -G plugins
```



Protocols

If the users want to know the details about the protocols that are recorded in the registration database then, they can use the protocols parameter. This output is also a bit less readable so that the user can take the help of any third party tool to beautify the report. This parameter prints the data in 3 fields. We have the protocol name, short name, and the filter name.

```
tshark -G protocols | head
```

```
root@kali:~# tshark -G protocols | head
Running as user "root" and group "root". This could be dangerous.
Lua Dissection  Lua Dissection  _ws.lua
Expert Info     Expert  _ws.expert
IEC 60870-5-104-Apci    104apci 104apci
IEC 60870-5-104-Asdu    104asdu 104asdu
29West Protocol 29West  29west
Pro-MPEG Code of Practice #3 release 2 FEC Protocol     2dparityfec     2dparityfec
3Com XNS Encapsulation  3COMXNS 3comxns
3GPP2 A11       3GPP2 A11       a11
IPv6 over Low power Wireless Personal Area Networks     6LoWPAN 6lowpan
802.11 radio information        802.11 Radio    wlan_radio
```

Values

Let's talk about the values report. It consists of value strings, range strings, true/false strings. There are three types of records available here. The first field can consist of one of these three characters representing the following:

V: Value Strings

R: Range Strings

T: True/False Strings

Moreover, in the value strings, we have the field abbreviation, integer value, and the string. In the range strings, we have the same values except it holds the lower bound and upper bound values.

```
1 tshark -G values | head
```

```
root@kali:~# tshark -G values | head
Running as user "root" and group "root". This could be dangerous.
R       ieee1722.subtype        0x0     0x0     IEC 61883/IIDC Format
R       ieee1722.subtype        0x1     0x1     MMA Streams
R       ieee1722.subtype        0x2     0x2     AVTP Audio Format
R       ieee1722.subtype        0x3     0x3     Compressed Video Format
R       ieee1722.subtype        0x4     0x4     Clock Reference Format
R       ieee1722.subtype        0x5     0x5     Time Synchronous Control Format
R       ieee1722.subtype        0x6     0x6     SDI Video Format
R       ieee1722.subtype        0x7     0x7     Raw Video Format
R       ieee1722.subtype        0x8     0x6d    Reserved for future protocols
R       ieee1722.subtype        0x6e    0x6e    AES Encrypted Format Continuous
```

Preferences

In case the user requires to revise the current preferences that are configured on the system, they can use the currentprefs options to read the preference saved in the file.

```
1 tshark -G currentprefs | head
```

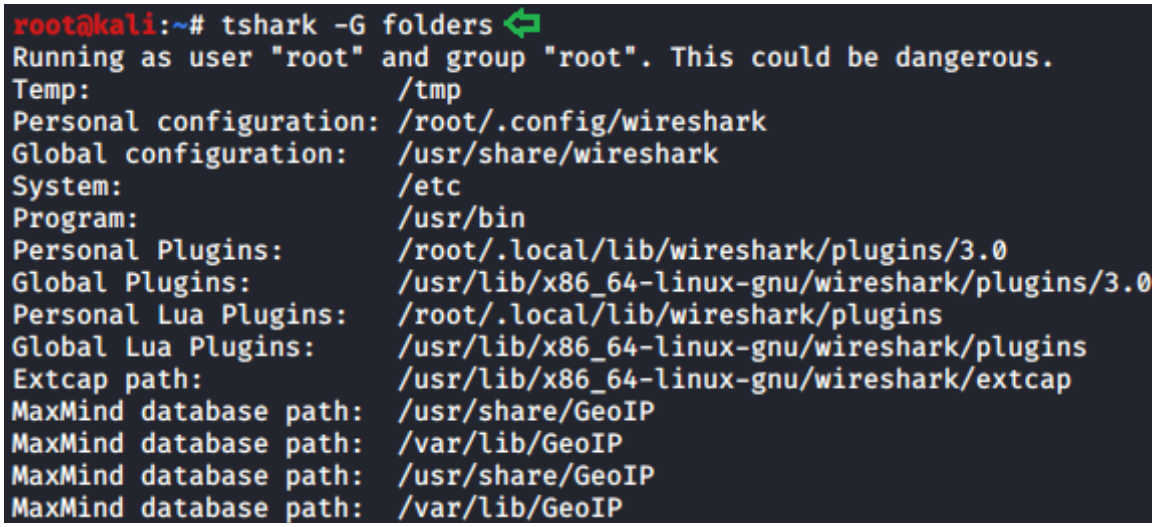
```
root@kali:~# tshark -G currentprefs | head
Running as user "root" and group "root". This could be dangerous.
# Configuration file for Wireshark 3.0.5.
#
# This file is regenerated each time preferences are saved within
# Wireshark. Making manual changes should be safe, however.
# Preferences that have been commented out have not been
# changed from their default value.

####### User Interface ########

# Open a console window (Windows only)
```

Folders

Suppose the user wants to manually change the configurations or get the program information or want to take a look at the lua configuration or some other

important files. The users need the path of those files to take a peek at them. Here the folders option comes a little handy.

```
tshark -G folders
```



Since we talked so extensively about TShark, It won't be justice if we won't talk about the tool that is heavily dependent on the data from TShark. Let's talk about PyShark.

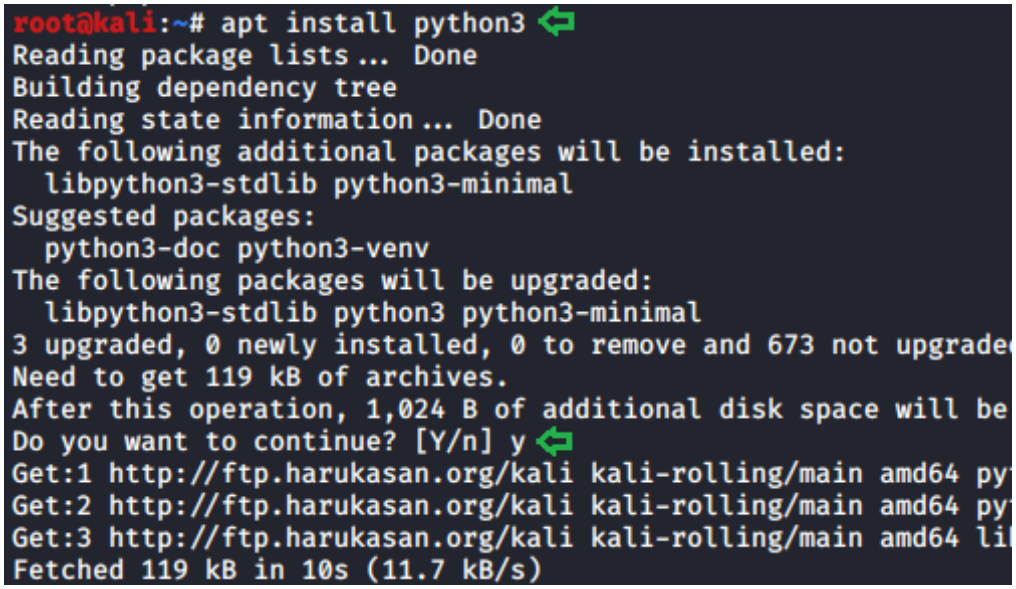
PyShark

It is essentially a wrapper that is based on Python. Its functionality is that allows the python packet parsing using the TShark dissectors. Many tools do the same job more or less but the difference is that this tool can export XMLs to use its parsing. You can read more about it from its GitHub page.

Installation

As the PyShark was developed using Python 3 and we don't Python 3 installed on our machine. We installed Python3 as shown in the image given below.

```
apt install python3
```



PyShark is available through the pip. But we don't have the pip for python 3 so we need to install it as well.

```
apt install python3-pip
```

```
root@kali:~# apt install python3-pip
Reading package lists ... Done
Building dependency tree
Reading state information ... Done
The following additional packages will be installed:
  libc-dev-bin libc6 libc6-dev libc6-i386 libcrypt-dev libc
  libpython3.7-stdlib python-pip-whl python3-dev python3-en
  python3-secretstorage python3-setuptools python3-wheel py
Suggested packages:
  glibc-doc libkf5wallet-bin gir1.2-gnomekeyring-1.0 python-
The following NEW packages will be installed:
  libcrypt-dev libcrypt1 libpython3-dev libpython3.7-dev py
  python3-keyrings.alt python3-pip python3-secretstorage py
The following packages will be upgraded:
  libc-dev-bin libc6 libc6-dev libc6-i386 libpython3.7 libp
  python3.7-minimal
10 upgraded, 15 newly installed, 0 to remove and 663 not up
Need to get 59.3 MB of archives
```

Since we have the python3 with pip we will install pyshark using pip command.

You can also install PyShark by cloning the git and running the setup.

```
1 pip3 install pyshark
```

```
root@kali:~# pip3 install pyshark
Collecting pyshark
  Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after conne
nnection.VerifiedHTTPSConnection object at 0x7fd92b76d610>: Failed to establish a new connec
ution')': /packages/b9/b0/ef87c71f7937ea8124944b2081210f9df10e47d2faa57d7c30d3e12af064/pysha
  Downloading https://files.pythonhosted.org/packages/b9/b0/ef87c71f7937ea8124944b2081210f9d
-none-any.whl
Collecting py (from pyshark)
  Downloading https://files.pythonhosted.org/packages/99/8d/21e1767c009211a62a8e3067280bfce7
ne-any.whl (83kB)
    100% |████████████████████████████████| 92kB 1.5MB/s
Requirement already satisfied: lxml in /usr/lib/python3/dist-packages (from pyshark) (4.4.1)
Installing collected packages: py, pyshark
Successfully installed py-1.8.1 pyshark-0.4.2.9
```

Live Capture

Now to get started, we need the python interpreter. To get this we write python3 and press enter. Now that we have the interpreter, the very first thing that we plan on doing is importing PyShark. Then we define network interface for the capture. Followed by that we will define the value of the timeout parameter for the capture.sniff function. At last, we will begin the capture. Here we can see that in the timeframe that we provided PyShark captured 9 packets.

```
python3
import pyshark
capture = pyshark.LiveCapture(interface='eth0')
capture.sniff(timeout=5)
capture
```

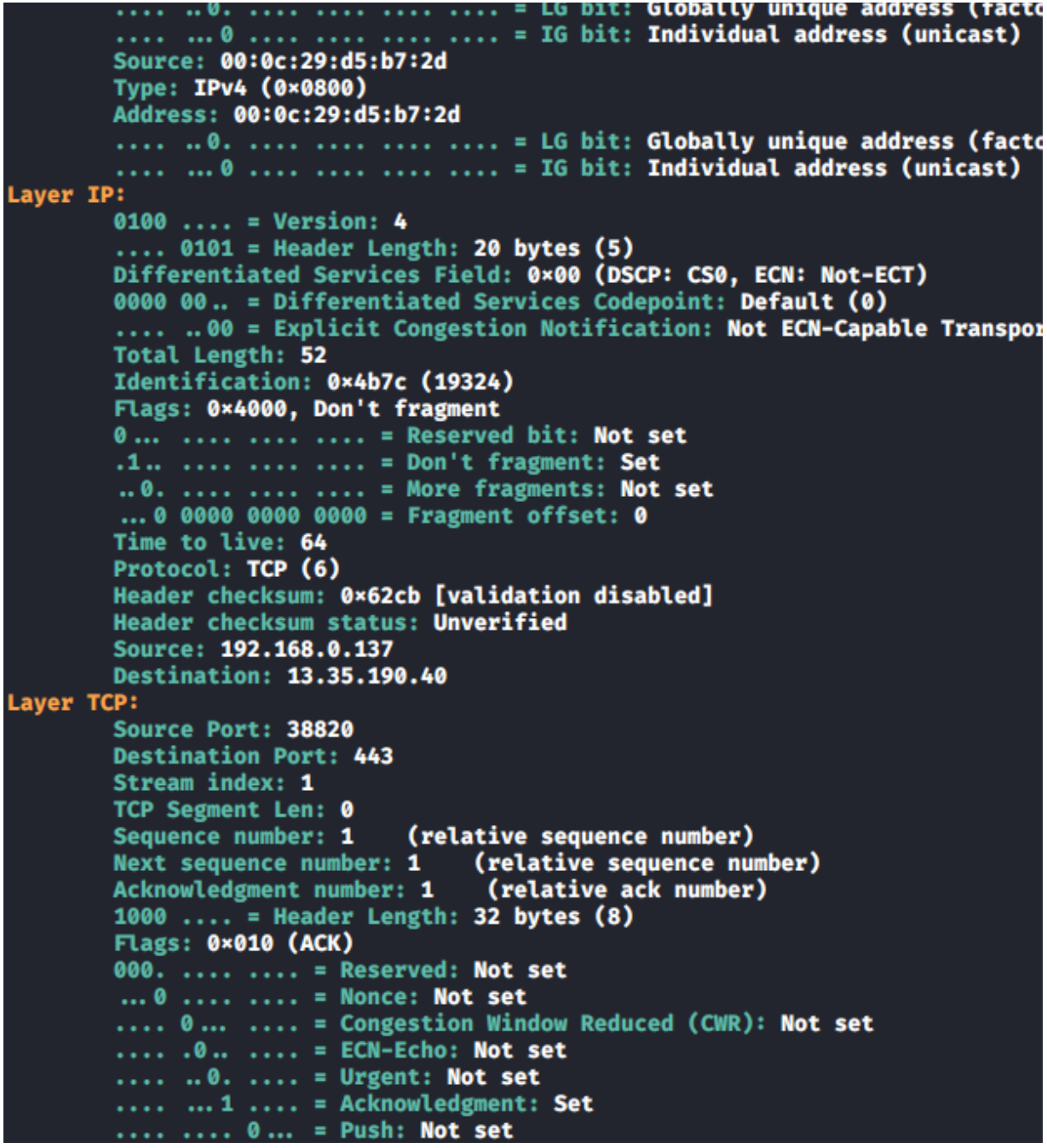
```
root@kali:~# python3
Python 3.7.6 (default, Jan 19 2020, 22:34:52)
[GCC 9.2.1 20200117] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyshark
>>> capture = pyshark.LiveCapture(interface='eth0')
>>> capture.sniff(timeout=5)
>>> capture
<LiveCapture (9 packets)>
```

Pretty Representation

There are multiple ways in which PyShark can represent data inside the captured packet. In the previous practical, we captured 9 packets. Let's take a look at the first packet that was captured with PyShark. Here we can see that we have a layer-wise analysis with the ETH Layer, IP Layer, and the TCP Layer.

```
capture[1].pretty_print()
```

```
>>> capture[1].pretty_print()
Layer ETH:
        Destination: 1c:5f:2b:59:e1:24
        Address: 1c:5f:2b:59:e1:24
```



Captured Length Field

In our capture, we saw some data that can consist of multiple attributes. These attributes need fields to get stored. To explore this field, we will be using the dir function in Python. We took the packet and then defined the variable named pkt with the value of that packet and saved it. Then using the dir function we saw explored the fields inside that particular capture. Here we can see that we have the pretty_print function which we used in the previous practical. We also have one field called captured_length to read into that we will write the name of the variable followed by the name of the field with a period (.) in between as depicted in the image below.

```
1 capture[2]
2 pkt = capture[2]
3 pkt
4 dir(pkt)
5 pkt.captured_length
```

```
>>> capture[2]
<TCP Packet>
>>> pkt = capture[2]
>>> pkt
<TCP Packet>
>>> dir(pkt)
['__bool__', '__class__', '__contains__', '__delattr__', '__dict__', '__dir__', '__doc__', '__
'__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_s
module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__s
sshook__', '__weakref__', '_packet_string', 'captured_length', 'eth', 'frame_info', 'get_multi
, 'interface_captured', 'ip', 'layers', 'length', 'number', 'pretty_print', 'show', 'sniff_tim
']
>>> pkt.captured_length
'66'
```

Layers, Src and Dst Fields

As we listed the fields in the previous step we saw that we have another field named layers. We read its contents as we did earlier to find out that we have 3

layers in this capture. Now to look into the individual layer, we need to get the fields of that individual layer. For that, we will again use the dir function. We used the dir function on the ETH layer as shown in the image given below. We observe that we have a field named src which means source, dst which means destination. We checked the value on those fields to find the physical address of the source and destination respectively.

```
pkt.layers
pkt.eth.src
pkt.eth.dst
pkt.eth.type
```

```
>>> pkt.layers
[<ETH Layer>, <IP Layer>, <TCP Layer>]
>>> dir(pkt.eth)
['DATA_LAYER', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__
e__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__
e__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__',
 '_field_prefix', '_get_all_field_lines', '_get_all_fields_with_alternates', '_g
_sanitize_field_name', 'addr', 'addr_resolved', 'dst', 'dst_resolved', 'field_na
d_value', 'ig', 'layer_name', 'lg', 'pretty_print', 'raw_mode', 'src', 'src_reso
>>> pkt.eth.src
'1c:5f:2b:59:e1:24'
>>> pkt.eth.dst
'00:0c:29:d5:b7:2d'
>>> pkt.eth.type
'0x00000800'
```

For our next step, we need the fields of the IP packet. We used the dir function on the IP layer and then we use src and dst fields here on this layer. We see that we have the IP Address as this is the IP layer. As the Ethernet layer works on the MAC Addresses they store the MAC Addresses of the Source and the Destination which changes when we come to the IP Layer.

```
dir(pkt.ip)
pkt.ip.src
```

```
3 pkt.ip.dst
4 pkt.ip.pretty_print()
```

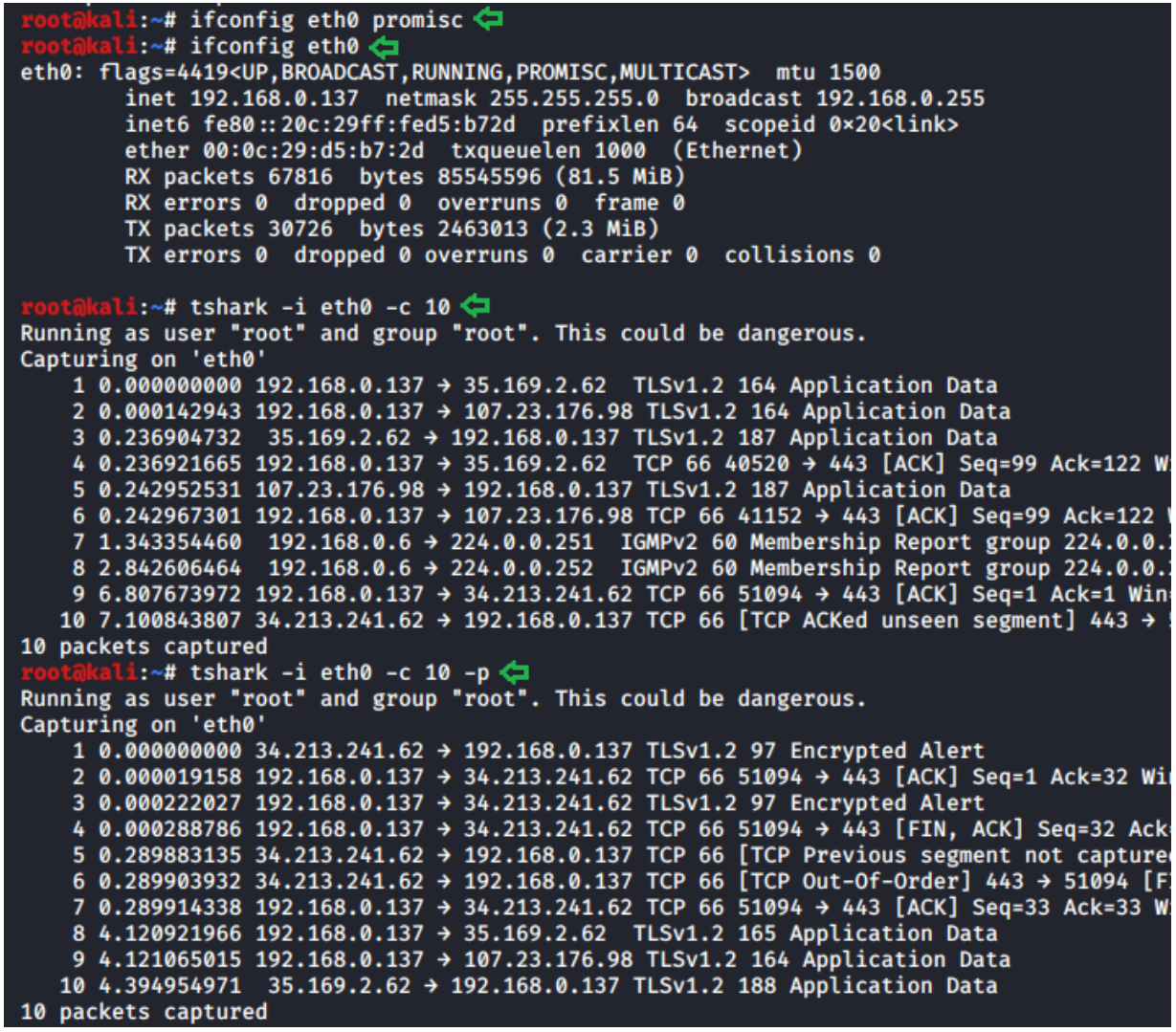
```
>>> dir(pkt.ip)
['DATA_LAYER', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq_
e__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le_
e__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__
 '_field_prefix', '_get_all_field_lines', '_get_all_fields_with_alternates', '_get_
_sanitize_field_name', 'addr', 'checksum', 'checksum_status', 'dsfield', 'dsfield_d
lags', 'flags_df', 'flags_mf', 'flags_rb', 'frag_offset', 'get', 'get_field', 'get_
 'id', 'layer_name', 'len', 'pretty_print', 'proto', 'raw_mode', 'src', 'src_host',
>>> pkt.ip.src
'13.35.190.40'
>>> pkt.ip.dst
'192.168.0.137'
>>> pkt.ip.pretty_print()
Layer IP:
        0100 .... = Version: 4
        .... 0101 = Header Length: 20 bytes (5)
        Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
        0001 00.. = Differentiated Services Codepoint: Unknown (4)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
        Total Length: 52
        Identification: 0x2e26 (11814)
        Flags: 0x4000, Don't fragment
        0... .... .... .... = Reserved bit: Not set
        .1.. .... .... .... = Don't fragment: Set
        ..0. .... .... .... = More fragments: Not set
        ...0 0000 0000 0000 = Fragment offset: 0
        Time to live: 248
        Protocol: TCP (6)
        Header checksum: 0xc810 [validation disabled]
        Header checksum status: Unverified
        Source: 13.35.190.40
        Destination: 192.168.0.137
```

Similarly, we can use the dir function and the field's value on any layer of the capture. This makes the investigation of the capture quite easier.

Promisc Capture

In previous articles we learned about the promisc mode that means that a network interface card will pass all frames received up to the operating system for processing, versus the traditional mode of operation wherein 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 we got stuck when we configured the network interface card to work on promisc mode. So while capturing traffic on TShark we can switch between the normal capture and the promisc capture using the –p parameter as shown in the image given below.

```
ifconfig eth0 promisc
ifconfig eth0
tshark -i eth0 -c 10
tshark -i eth0 -c  10 -p
```



Author: Shubham Sharma is a Pentester, Cybersecurity Researcher and Enthusiast, contact here.

Beginners Guide to TShark (Part 2)

posted in PENETRATION TESTING on FEBRUARY 19, 2020 by RAJ CHANDEL with 0 COMMENT

In the previous article, we learned about the basic functionalities of this wonderful tool called TShark. If you haven't read it until now. Click here.

TL; DR

In this part, we will the Statistical Functionalities of TShark. We will understand different ways in which we can sort our traffic capture so that we can analyse it faster and effectively.

Table of Content

- **Statistical Options**
- **Protocol Hierarchy Statistics**
- **Read Filter Analysis**
- **Endpoints Analysis**
- **Conversation Analysis**
- **Expert Mode Analysis**
- **Packet Distribution Tree**
- **Packet Length Tree**
- **Color Based Output Analysis**
- **Ring Buffer Analysis**
- **Auto-Stop**
    - Duration
    - File Size

- Data-Link Types

Statistical Options

TShark collects different types of Statistics and displays their result after finishing the reading of the captured file. To accomplish this, we will be using the “-z” parameter with TShark. Initially, to learn about all the different options inside the “-z” parameter, we will be running the TShark with the “-z” parameter followed by the help keyword. This gives us an exhaustive list of various supported formats as shown in the image given below.

```
root@kali:~# tshark -z help
Running as user "root" and group "root". This could be dangerous.
tshark: The available statistics for the "-z" option are:
     afp,srt
     ancp,tree
     ansi_a,bsmap
     ansi_a,dtap
     ansi_map
     bacapp_instanceid,tree
     bacapp_ip,tree
     bacapp_objectid,tree
     bacapp_service,tree
     camel,counter
     camel,srt
     collectd,tree
     conv,bluetooth
     conv,eth
     conv,fc
     conv,fddi
     conv,ip
     conv,ipv6
     conv,ipx
     conv,jxta
     conv,mptcp
     conv,ncp
     conv,rsvp
     conv,sctp
     conv,sll
     conv,tcp
     conv,tr
     conv,udp
     conv,usb
     conv,wlan
     dcerpc,srt
     dests,tree
     dhcp,stat
     diameter,avp
     diameter,srt
     dns,tree
     endpoints,bluetooth
```

Protocol Hierarchy Statistics

Using the TShark we can create a Protocol based Hierarchy Statistics listing the number of packets and bytes using the “io,phs” option in the “-z” parameter. In the case where no filter is given after the “io,phs” option, the statistics will be calculated for all the packets in the scope. But if a specific filter is provided than the TShark will calculate statistics for those packets that match the filter provided by the user. For our demonstration, we first captured some traffic and wrote the contents on a pcap file using the techniques that we learned in part 1 of this article series. Then we will be taking the traffic from the file, and then sort the data into a Protocol Hierarchy. Here we can observe that we have the frames count, size of packets in bytes and the Protocol used for the transmission.

```
1 tshark  -r wlan.pcap -z io,phs
```

```
===================================================================
Protocol Hierarchy Statistics
Filter:

radiotap                                 frames:66690 bytes:15014549
  wlan_radio                             frames:66690 bytes:15014549
    wlan                                 frames:66690 bytes:15014549
      wlan                               frames:6873 bytes:1923747
      data                               frames:14539 bytes:9494059
      llc                                frames:3158 bytes:1295577
        eapol                            frames:6 bytes:1162
        ipv6                             frames:16 bytes:2136
          icmpv6                         frames:16 bytes:2136
        ip                               frames:3124 bytes:1291079
          udp                            frames:143 bytes:25311
            dhcp                         frames:6 bytes:2448
            dns                          frames:126 bytes:21131
            ntp                          frames:3 bytes:444
            mdns                         frames:8 bytes:1288
          icmp                           frames:2 bytes:240
          tcp                            frames:2979 bytes:1265528
            tls                          frames:781 bytes:455386
              tcp.segments               frames:74 bytes:60600
                tls                      frames:62 bytes:53122
            http                         frames:248 bytes:123041
              data-text-lines            frames:9 bytes:6487
                tcp.segments             frames:4 bytes:2696
              image-jfif                 frames:6 bytes:4156
                tcp.segments             frames:6 bytes:4156
              image-gif                  frames:2 bytes:1352
              tcp.segments               frames:3 bytes:1402
            data                         frames:2 bytes:2924
            tcp.segments                 frames:1 bytes:1462
            _ws.malformed                frames:5 bytes:2666
        arp                              frames:12 bytes:1200
===================================================================
```

Read Filter Analysis

During the first pass analysis of the packet, the specified filter (which uses the syntax of read/display filters, rather than that of capture filters) has to be applied. Packets which are not matching the filter are not considered for future passes. This parameter makes sense with multiple passes. Note that forward-looking fields such as ‘response in frame #’ cannot be used with this filter since they will not have been calculated when this filter is applied. The “-2” parameter performs a two-pass analysis. This causes TShark to buffer output until the entire first pass is done, but allows it to fill in fields that require future knowledge, it also permits reassembly frame dependencies to be calculated correctly. Here we can see two different analysis one of them is first-pass analysis and the latter is the two-pass analysis.

```
tshark -r wlan.pcap -z io,phs,udp -q
tshark -r wlan.pcap -z io,phs -q -2 -R udp
```

```
root@kali:~# tshark -r wlan.pcap -z io,phs,udp -q
Running as user "root" and group "root". This could be dangerous.

================================================================
Protocol Hierarchy Statistics
Filter: udp

radiotap                                 frames:143 bytes:25311
  wlan_radio                             frames:143 bytes:25311
    wlan                                 frames:143 bytes:25311
      llc                                frames:143 bytes:25311
        ip                               frames:143 bytes:25311
          udp                            frames:143 bytes:25311
            dhcp                         frames:6 bytes:2448
            dns                          frames:126 bytes:21131
            ntp                          frames:3 bytes:444
            mdns                         frames:8 bytes:1288
================================================================
root@kali:~# tshark -r wlan.pcap -z io,phs -q -2 -R udp
Running as user "root" and group "root". This could be dangerous.

================================================================
Protocol Hierarchy Statistics
Filter:

radiotap                                 frames:143 bytes:25311
  wlan_radio                             frames:143 bytes:25311
    wlan                                 frames:143 bytes:25311
      llc                                frames:143 bytes:25311
        ip                               frames:143 bytes:25311
          udp                            frames:143 bytes:25311
            dhcp                         frames:6 bytes:2448
            dns                          frames:126 bytes:21131
            ntp                          frames:3 bytes:444
            mdns                         frames:8 bytes:1288
================================================================
```

Endpoints Analysis

Our next option which helps us with the statistics is the “endpoints”. It will create a table that will list all endpoints that could be seen in the capture. The type function which can be used with the endpoint option will specify the endpoint type for which we want to generate the statistics.

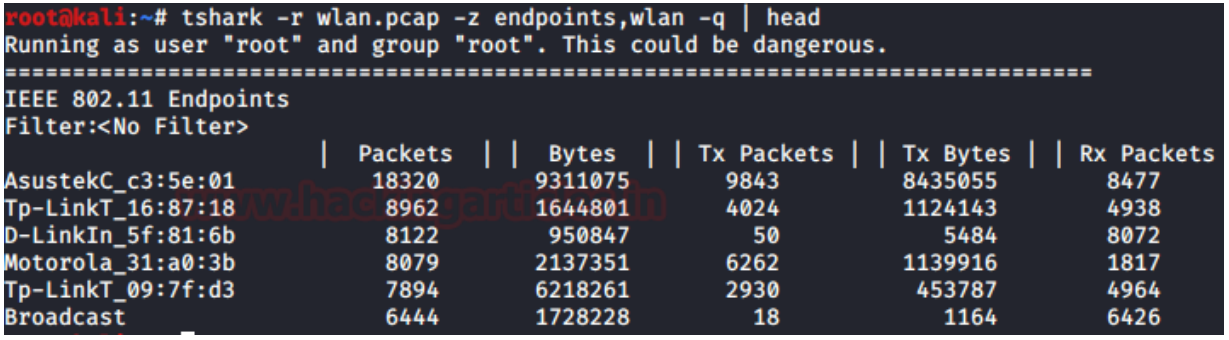
The list of Endpoints that are supported by TShark is:

Sno.	Filter	Description
1	**“bluetooth”**	Bluetooth Addresses
2	**“eth”**	Ethernet Addresses
3	**“fc”**	Fiber Channel Addresses
4	**“fddi”**	FDDI Addresses
5	**“ip”**	IPv4 Addresses
6	**“ipv6”**	IPv6 Addresses
7	**“ipx”**	IPX Addresses
8	**“jxta”**	JXTS Addresses
9	**“ncp”**	NCP Addresses
10	**“rsvp”**	RSVP Addresses
11	**“sctp”**	SCTP Addresses

12	**“tcp”**	TCP/IP socket pairs Both IPv4 and IPv6 supported
13	**“tr”**	Token Ring Addresses
14	**“usb”**	USB Addresses
15	**“udp”**	UDP/IP socket pairs Both IPv4 and IPv6 supported
16	**“wlan”**	IEEE 802.11 addresses

In case that we have specified the filter option then the statistics calculations are done for that particular specified filter. The table like the one generated in the image shown below is generated by picking up single line form each conversation and displayed against the number of packets per byte in each direction as well as the total number of packets per byte. This table is by default sorted according to the total number of frames.

```
1 tshark -r wlan.pcap -z endpoints,wlan -q | head
```



Conversation Analysis

Let's move on to the next option which is quite similar to the previous option. It helps us with the statistics is the "conversation". It will create a table that will list all conversation that could be seen in the capture. The type function which can be used with the conversation option will specify the conversation type for which we want to generate the statistics.

If we have specified the filter option then the statistics calculations are done for that particular specified filter. The table generated by picking up single line form each conversation and displayed against the number of packets per byte in each direction, the total number of packets per byte as well as the direction of the conversation travel. This table is by default sorted according to the total number of frames.

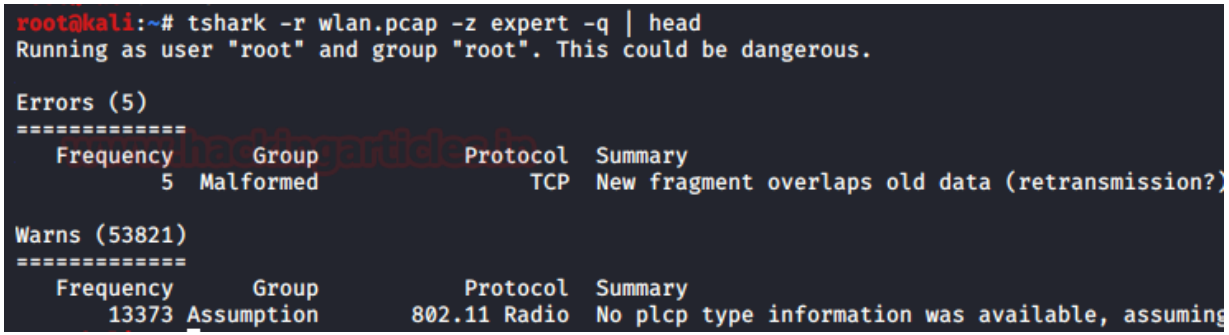
```
1 tshark -r wlan.pcap -z conv,wlan -q | head
```

```
root@kali:~# tshark -r wlan.pcap -z conv,wlan -q | head
Running as user "root" and group "root". This could be dangerous.
================================================================================
IEEE 802.11 Conversations
Filter:<No Filter>
                                               |       ←       |       →       |     Total     |
                                               | Frames  Bytes | Frames  Bytes | Frames  Bytes |
AsustekC_c3:5e:01    ↔ Tp-LinkT_09:7f:d3       |   2841 441682 |   4753 5753274 |   7594 6194956 | 15
Motorola_31:a0:3b    ↔ D-LinkIn_5f:81:6b       |      3    431 |   3455  696782 |   3458  697213 |
Motorola_31:a0:3b    ↔ Tp-LinkT_16:87:18       |   1566 937369 |   1689  358208 |   3255 1295577 | 15
AsustekC_c3:5e:01    ↔ IntelCor_96:a1:a9       |    721  91683 |   2372 2046278 |   3093 2137961 | 15
00:51:88:31:a0:3b    ↔ D-LinkIn_5f:81:6b       |      0      0 |   2898  144900 |   2898  144900 |
```

Expert Mode Analysis

The TShark Statistics Module have an Expert Mode. It collects a huge amount of data based on Expert Info and then prints this information in a specific order. All this data is grouped in the sets of severity like Errors, Warnings, etc., We can use the expert mode with a particular protocol as well. In that case, it will display all the expert items of that particular protocol.

```
tshark -r wlan.pcap -z expert -q | head
```



Packet Distribution Tree

In this option, we take the traffic form a packet and then drive it through the “http,tree” option under the “-z” parameter to count the number of the HTTP requests, their mods as well as the status code. This is a rather modular approach that is very easy to understand and analyse. Here in our case, we took the packet that we captured earlier and then drove it through the tree option that gave us the Information that a total of 126 requests were generated out of which 14 gave back the “200 OK”. It means that the rest of them either gave back an error or were redirected to another server giving back a 3XX series status code.

```
tshark -r wlan.pcap -z http,tree -q
```

```
root@kali:~# tshark -r wlan.pcap -z http,tree -q
Running as user "root" and group "root". This could be dangerous.

================================================================================
HTTP/Packet Counter:
Topic / Item                  Count         Average       Min val       Max val       Rate (ms)     Percent
--------------------------------------------------------------------------------
Total HTTP Packets            248                                                   0.0038        100%
 HTTP Request Packets         126                                                   0.0019        50.81%
  GET                         126                                                   0.0019        100.00%
 HTTP Response Packets        122                                                   0.0019        49.19%
  3xx: Redirection            105                                                   0.0016        86.07%
   304 Not Modified           101                                                   0.0015        96.19%
   302 Found                  3                                                     0.0000        2.86%
   301 Moved Permanently      1                                                     0.0000        0.95%
  2xx: Success                17                                                    0.0003        13.93%
   200 OK                     14                                                    0.0002        82.35%
   204 No Content             3                                                     0.0000        17.65%
  ???: broken                 0                                                     0.0000        0.00%
  5xx: Server Error           0                                                     0.0000        0.00%
  4xx: Client Error           0                                                     0.0000        0.00%
  1xx: Informational          0                                                     0.0000        0.00%
 Other HTTP Packets           0                                                     0.0000        0.00%
```

Packet Length Tree

As long as we are talking about the Tree option, let's explore it a bit. We have a large variety of ways in which we can use the tree option in combination with other option. To demonstrate that, we decided to use the packet length option with the tree option. This will sort the data on the basis of the size of the packets and then generate a table with it. Now, this table will not only consist of the length of the packets, but it will also have the count of the packet. The minimum value of the length in the range of the size of the packets. It will also calculate the size as well as the Percentage of the packets inside the range of packet length

```
1 tshark -r wlan.pcap -z plen,tree -q
```

```
root@kali:~# tshark -r wlan.pcap -z plen,tree -q
Running as user "root" and group "root". This could be dangerous.

================================================================================
Packet Lengths:
Topic / Item       Count         Average       Min val       Max val       Rate (ms)     Percent
--------------------------------------------------------------------------------
Packet Lengths     66690         225.14        50            1582          0.0004        100%
 0-19              0             -             -             -             0.0000        0.00%
 20-39             0             -             -             -             0.0000        0.00%
 40-79             42235         54.68         50            76            0.0003        63.33%
 80-159            9477          134.43        86            159           0.0001        14.21%
 160-319           6071          257.61        160           317           0.0000        9.10%
 320-639           2724          390.89        320           639           0.0000        4.08%
 640-1279          456           844.38        640           1278          0.0000        0.68%
 1280-2559         5727          1469.77       1280          1582          0.0000        8.59%
 2560-5119         0             -             -             -             0.0000        0.00%
 5120 and greater  0             -             -             -             0.0000        0.00%
```


Color Based Output Analysis

Note: Your terminal must support color output in order for this option to work correctly.

We can enable the coloring of packets according to standard Wireshark color filters. On Windows, colors are limited to the standard console character attribute colors. In this option, we can set up the colors according to the display filter. This helps in quickly locating a specific packet in the bunch of similar packets. It also helps in locating Handshakes in communication traffic. This can be enabled using the following command.

```
1 tshark -r color.pcap --color
```

```
root@kali:~# tshark -r color.pcap --color
Running as user "root" and group "root". This could be dangerous.
    1 0.000000000  192.168.0.6 → 224.0.0.252  IGMPv2 60 Membership Report group 2
    2 1.308991630 192.168.0.137 → 8.8.8.8      DNS 84 Standard query 0×bd04 A det
    3 1.309108098 192.168.0.137 → 8.8.8.8      DNS 84 Standard query 0×d70e AAAA
    4 1.314734876      8.8.8.8 → 192.168.0.137 DNS 248 Standard query response 0×
cd.akamai.net A 23.32.28.31 A 23.32.28.42
    5 1.317061896      8.8.8.8 → 192.168.0.137 DNS 272 Standard query response 0×
.dscd.akamai.net AAAA 2600:140f:3400::1720:1c1f AAAA 2600:140f:3400::1720:1c2a
    6 1.351099604 192.168.0.137 → 23.32.28.31  TCP 74 33274 → 80 [SYN] Seq=0 Win=
    7 1.360371655  23.32.28.31 → 192.168.0.137 TCP 74 80 → 33274 [SYN, ACK] Seq=0
    8 1.360407102 192.168.0.137 → 23.32.28.31  TCP 66 33274 → 80 [ACK] Seq=1 Ack=
    9 1.360667272 192.168.0.137 → 23.32.28.31  HTTP 354 GET /success.txt HTTP/1.1
   10 1.366231541  23.32.28.31 → 192.168.0.137 TCP 66 80 → 33274 [ACK] Seq=1 Ack=
   11 1.368532386  23.32.28.31 → 192.168.0.137 HTTP 473 HTTP/1.1 200 OK  (text/pl
   12 1.368576332 192.168.0.137 → 23.32.28.31  TCP 66 33274 → 80 [ACK] Seq=289 Ac
   13 1.714041355 192.168.0.137 → 8.8.8.8      DNS 72 Standard query 0×2172 A www
   14 1.714151234 192.168.0.137 → 8.8.8.8      DNS 72 Standard query 0×6d77 AAAA
   15 1.715114796 192.168.0.137 → 8.8.8.8      DNS 73 Standard query 0×3b2e A kal
   16 1.715179313 192.168.0.137 → 8.8.8.8      DNS 73 Standard query 0×af30 AAAA
   17 1.715291271 192.168.0.137 → 8.8.8.8      DNS 74 Standard query 0×99d0 A too
   18 1.715336702 192.168.0.137 → 8.8.8.8      DNS 74 Standard query 0×a3d2 AAAA
   19 1.726762319      8.8.8.8 → 192.168.0.137 DNS 132 Standard query response 0×
   20 1.730538887      8.8.8.8 → 192.168.0.137 DNS 133 Standard query response 0×
   21 1.780500105 192.168.0.137 → 8.8.8.8      DNS 84 Standard query 0×97f4 A sni
   22 1.780608110 192.168.0.137 → 8.8.8.8      DNS 84 Standard query 0×39f9 AAAA
   23 1.786609781      8.8.8.8 → 192.168.0.137 DNS 191 Standard query response 0×
   24 1.786635886      8.8.8.8 → 192.168.0.137 DNS 211 Standard query response 0×
   25 1.790627044 192.168.0.137 → 13.33.169.121 TCP 74 38962 → 443 [SYN] Seq=0 Wi
   26 1.833760308 13.33.169.121 → 192.168.0.137 TCP 74 443 → 38962 [SYN, ACK] Seq
   27 1.833783843 192.168.0.137 → 13.33.169.121 TCP 66 38962 → 443 [ACK] Seq=1 Ac
```

Ring Buffer Analysis

By default, the TShark to runs in the “multiple files” mode. In this mode, the TShark writes into several capture files. When the first capture file fills up to a certain capacity, the TShark switches to the next file and so on. The file names that we want to create can be stated using the -w parameter. The number of files,

creation data and creation time will be concatenated with the name provided next to -w parameter to form the complete name of the file.

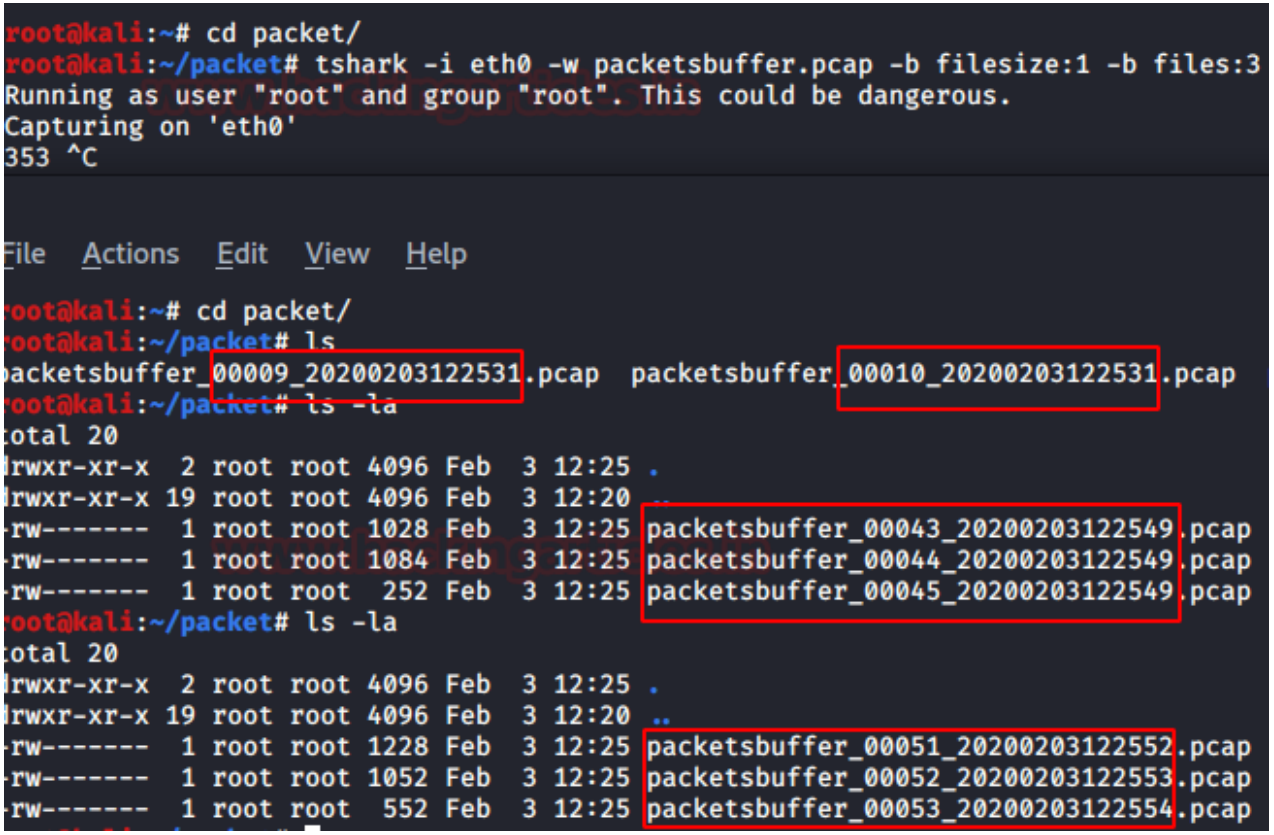
The files option will fill up new files until the number of files is specified. at that moment the TShark will discard data in the first file and start writing to that file and so on. If the files option is not set, new files filled up until one of the captures stops conditions matches or until the disk is full.

There are a lot of criteria upon which the ring buffer works but, in our demonstration, we used 2 of them. Files and the Filesize.

files: value begin again with the first file after value number of files were written (form a ring buffer). This value must be less than 100000.

filesize: value switches to the next file after it reaches a size of value kB. Note that the file size is limited to a maximum value of 2 GiB.

```
1 tshark -I eth0 -w packetsbuffer.pcap -b filesize:1 -b file:3
```



Auto-Stop

Under the huge array of the options, we have one option called auto-stop. As the name tells us that it will stop the traffic capture after the criteria are matched.

Duration

We have a couple of options, in our demonstration, we used the duration criteria. We specified the duration to 10. This value is in seconds. So, the capture tells us that in the time of 10 seconds, we captured 9 packets.

```
1 tshark -i eth0 -a duration:10
```

```
root@kali:~# tshark -i eth0 -a duration:10
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
    1 0.000000000 192.168.0.137 → 52.73.26.88  TLSv1.2 841 Application Data
    2 0.228521540  52.73.26.88 → 192.168.0.137 TLSv1.2 191 Application Data
    3 0.228547939 192.168.0.137 → 52.73.26.88  TCP 66 50990 → 443 [ACK] Seq
    4 0.228611423  52.73.26.88 → 192.168.0.137 TCP 66 443 → 50990 [ACK] Seq
    5 0.228615310 192.168.0.137 → 52.73.26.88  TCP 66 [TCP Dup ACK 3#1] 509
    6 4.977273190 192.168.0.137 → 13.249.226.169 TCP 66 34476 → 443 [ACK] S
    7 5.010827964 13.249.226.169 → 192.168.0.137 TCP 66 [TCP ACKed unseen s
    8 6.512880036 192.168.0.137 → 104.79.123.250 TCP 66 37426 → 443 [ACK] S
    9 6.623442093 104.79.123.250 → 192.168.0.137 TCP 66 [TCP ACKed unseen s
9 packets captured
```

File Size

Now another criterion for the auto-stop option is the file size. The TShark will stop writing to the specified capture file after it reaches a size provided by the user. In our demonstration, we set the filesize to 1. This value is in kB. We used the directory listing command to show that the capture was terminated as soon as the file reached the size of 1 kB.

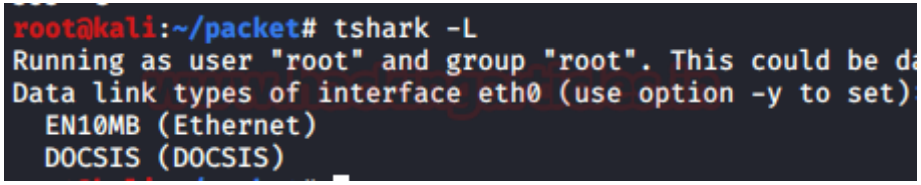
```
1 tshark -i eth0 -w 1.pcap -a filesize:1
```

```
root@kali:~# tshark -i eth0 -w 1.pcap  -a filesize:1
Running as user "root" and group "root". This could be dange
Capturing on 'eth0'
6
root@kali:~# ls -la
total 16156
drwxr-xr-x 19 root root     4096 Feb  3 12:33 .
drwxr-xr-x 18 root root     4096 Nov 25 12:38 ..
-rw-------  1 root root     1172 Feb  3 12:33 1.pcap
-rw-r--r--  1 root root     5161 Feb  3 12:24 .bash_history
```

Data-Link Types

At last, we can also modify the statistics of the captured traffic data based on the Data-Link Types. For that we will have to use an independent parameter, "-L". In our demonstration, we used the "-L" parameter to show that we have data links like EN10MB specified for the Ethernet Traffic and others.

```
1 tshark -L
```



Author: Shubham Sharma is a Pentester, Cybersecurity Researcher and Enthusiast, contact **here**.

← OLDER POSTS

NEWER POSTS →