Beginners Guide to TShark (Part 1)

February 9, 2020 By Raj Chandel

In this article, we will learn about TShark which is a well-known network protocol analyzer. It lets us capture the data packets, from the live network. It also allows us, to read or analyze the previously captured data packets of a saved file.

Table of content

Network traffic

As we know, network traffic or data traffic is the amount of data transferring across the network at some given point of time. Network data, in computer networks, is in the form of network data packets. Analyzing these network packets provides network security as it helps us to monitor traffic. As a benefit, if there is some unusual amount of data traffic in a network which is a possible sign of an attack then Tshark can help us know before it too late and the attack can be terminated as data traffic reports provide insights into preventing some good attacks.

Traffic volume is a term which comes under network traffic analyzing. Network traffic volume is the measure of the total work done. It is defined as the average data traffic intensity and time period of its network data packet study.

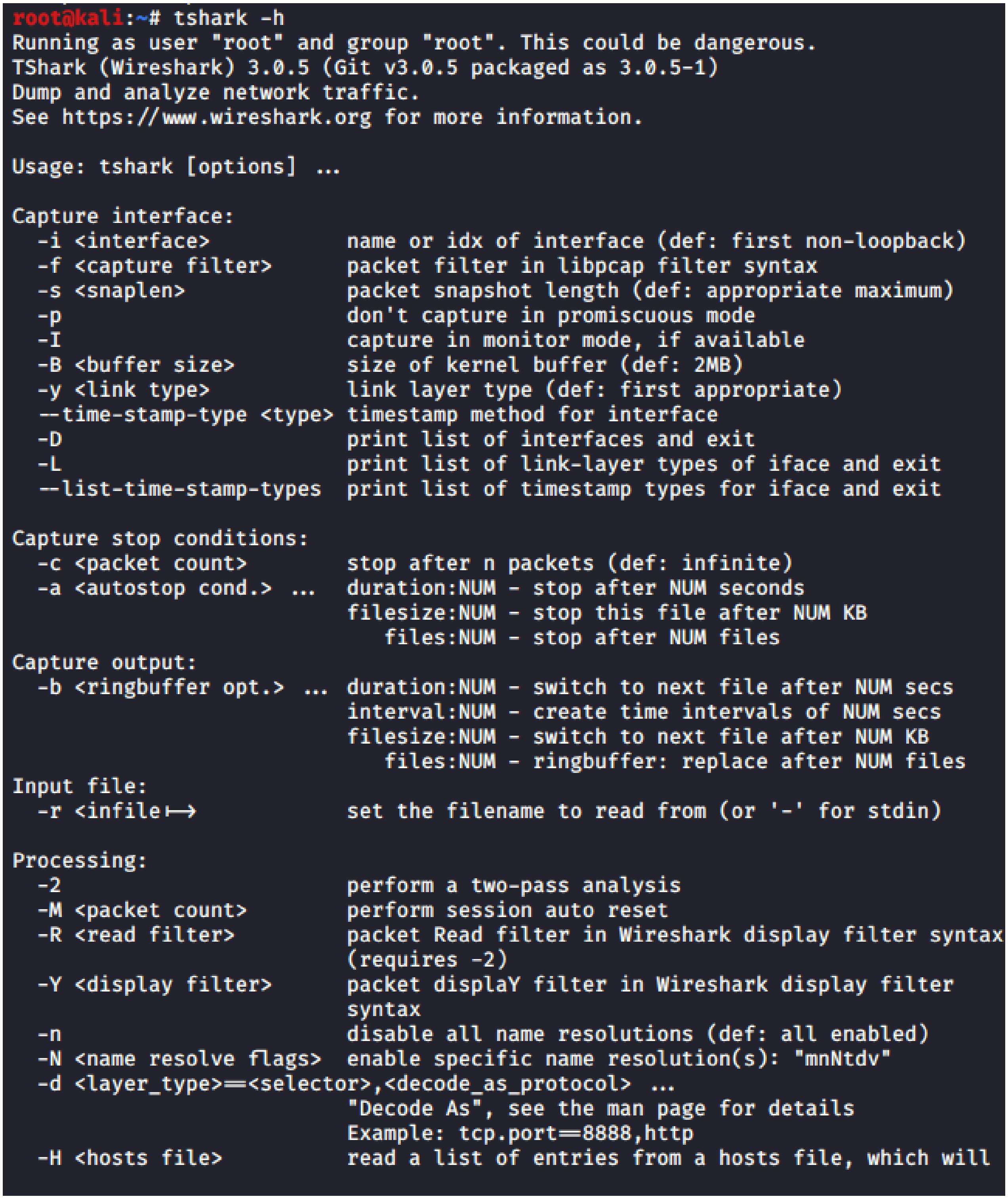
Introduction to TShark

Tshark, a well known and powerful command-line tool and is used as a network analyzer. It is developed by Wireshark. It’s working structure is quite similar to Tcpdump, but it has some powerful decoders and filters. TShark is capable of capturing the data packets information of different network layers and display them in different formats.

TShark is used to analyze real-time network traffic and it can read .pcap files to analyze the information, dig into the details of those connections, helping security professionals to identify their network problem.

TShark is a command-line based tool, which can do anything that Wireshark does. So let us start our learning process with TShark and therefore launch this tool and explore its options. To check out all the parameters, use the following command :

```
tshark -h
```

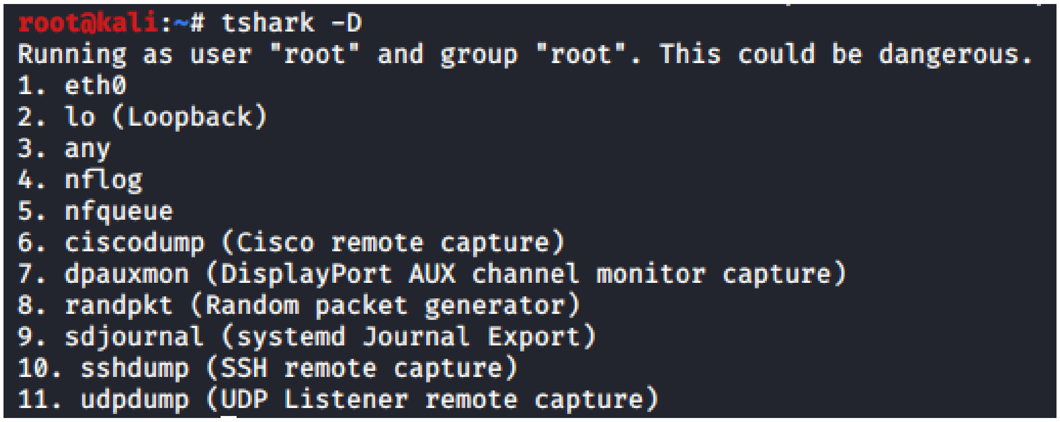


```
root@kali:~# tshark -h
Running as user "root" and group "root". This could be dangerous.
TShark (Wireshark) 3.0.5 (Git v3.0.5 packaged as 3.0.5-1)
Dump and analyze network traffic.
See https://www.wireshark.org for more information.

Usage: tshark [options] ...

Capture interface:
  -i <interface>           name or idx of interface (def: first non-loopback)
  -f <capture filter>      packet filter in libpcap filter syntax
  -s <snaplen>             packet snapshot length (def: appropriate maximum)
  -p                       don't capture in promiscuous mode
  -I                       capture in monitor mode, if available
  -B <buffer size>         size of kernel buffer (def: 2MB)
  -y <link type>           link layer type (def: first appropriate)
  --time-stamp-type <type> timestamp method for interface
  -D                       print list of interfaces and exit
  -L                       print list of link-layer types of iface and exit
  --list-time-stamp-types  print list of timestamp types for iface and exit

Capture stop conditions:
  -c <packet count>        stop after n packets (def: infinite)
  -a <autostop cond.> ...  duration:NUM - stop after NUM seconds
                           filesize:NUM - stop this file after NUM KB
                              files:NUM - stop after NUM files
Capture output:
  -b <ringbuffer opt.> ... duration:NUM - switch to next file after NUM secs
                           interval:NUM - create time intervals of NUM secs
                           filesize:NUM - switch to next file after NUM KB
                              files:NUM - ringbuffer: replace after NUM files
Input file:
  -r <infile|->            set the filename to read from (or '-' for stdin)

Processing:
  -2                       perform a two-pass analysis
  -M <packet count>        perform session auto reset
  -R <read filter>         packet Read filter in Wireshark display filter syntax
                           (requires -2)
  -Y <display filter>      packet displaY filter in Wireshark display filter
                           syntax
  -n                       disable all name resolutions (def: all enabled)
  -N <name resolve flags>  enable specific name resolution(s): "mnNtdv"
  -d <layer_type>==<selector>,<decode_as_protocol> ...
                           "Decode As", see the man page for details
                           Example: tcp.port==8888,http
  -H <hosts file>          read a list of entries from a hosts file, which will
```

List interfaces

TShark prints a list of the interfaces whose traffic it can capture. Each interface is referred to by their serial number and as you can see it is followed by a text description of the network interface. These interfaces can be specified using **-i parameter**; which is used to specify the network whose traffic we want to capture. And to check out these interfaces you can use the **parameter -D** as shown in the image below :

```
tshark -D
```

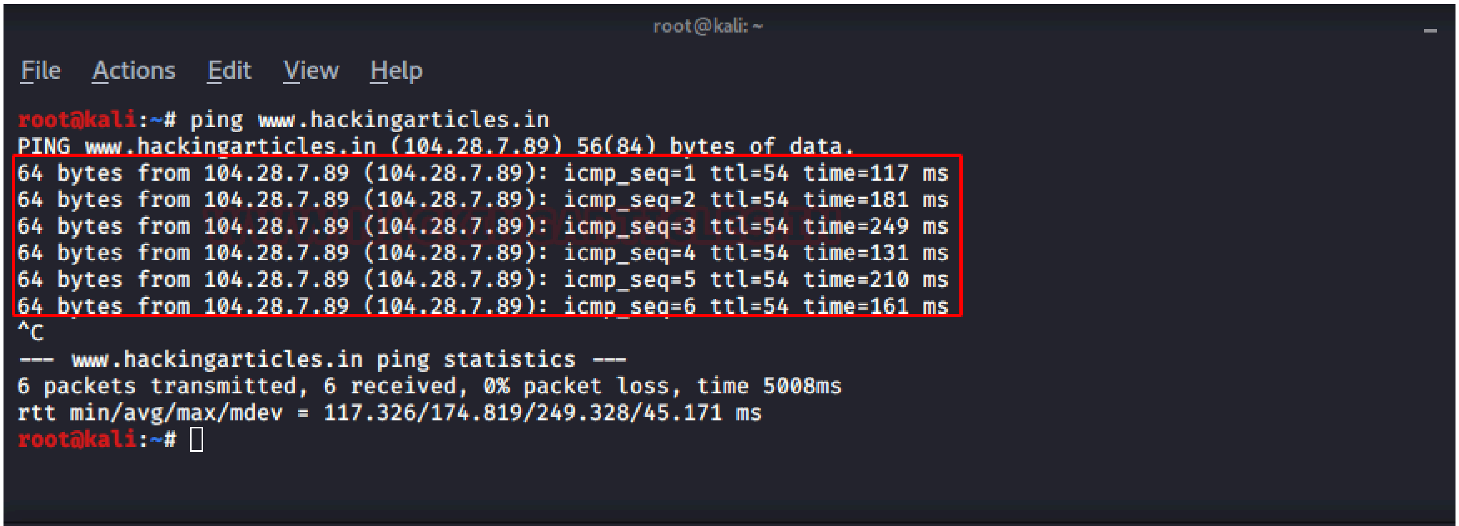


```
root@kali:~# tshark -D
Running as user "root" and group "root". This could be dangerous.
1. eth0
2. lo (Loopback)
3. any
4. nflog
5. nfqueue
6. ciscodump (Cisco remote capture)
7. dpauxmon (DisplayPort AUX channel monitor capture)
8. randpkt (Random packet generator)
9. sdjournal (systemd Journal Export)
10. sshdump (SSH remote capture)
11. udpdump (UDP Listener remote capture)
```

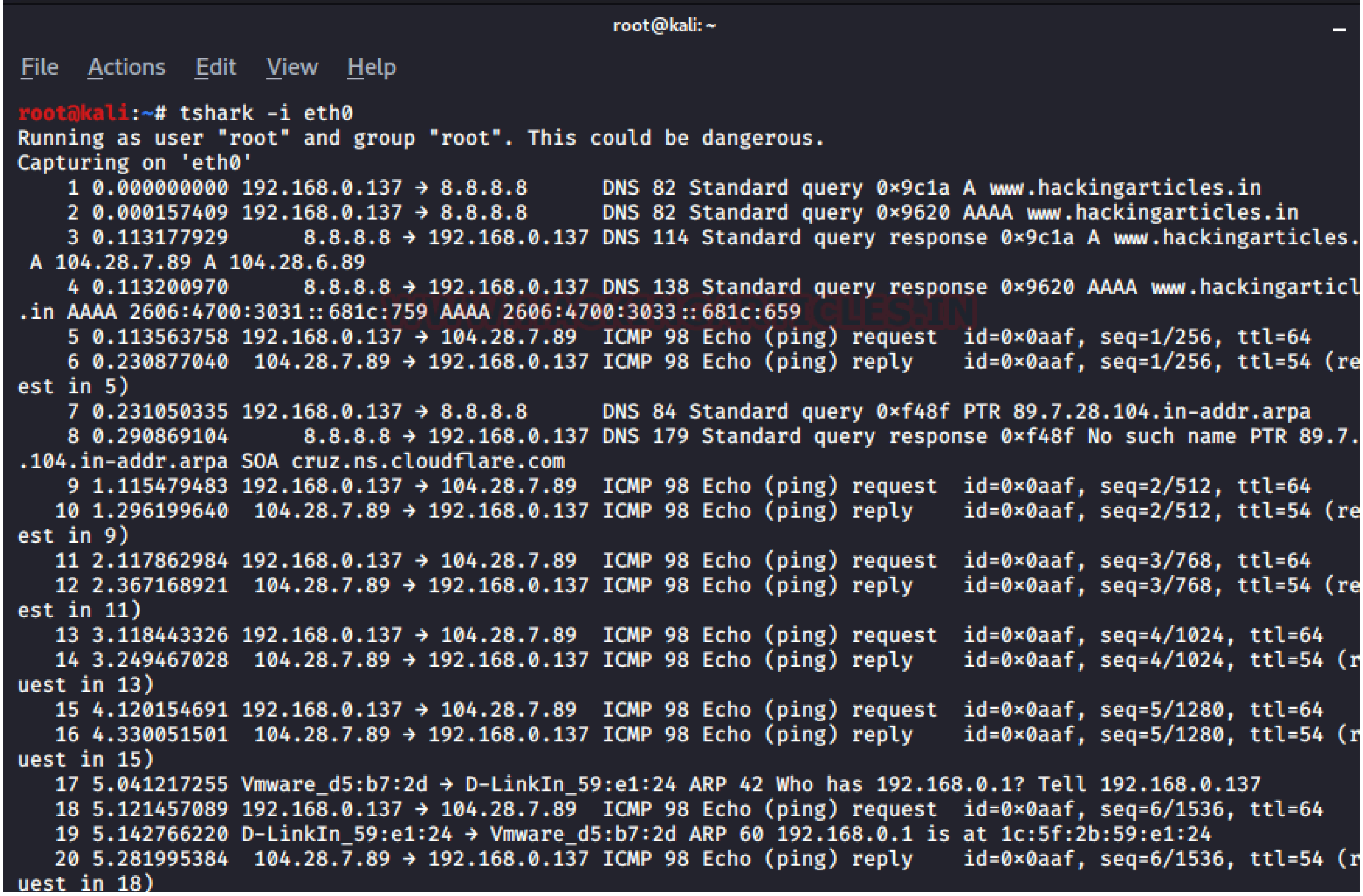
Capture traffic

Let’s now try to capture traffic, we have various choice of interface to capture traffic and therefore one can choose whichever depending on their needs and requirement. But in our scenario, the interface which we are going to use is “eth0”. In order to capture traffic, we need to initiate one too as we are testing on a controlled network and for that use ping command and then to capture traffic we have to just specify the interface name by using -i parameter as shown in the image below :

```
ping www.hackingarticles.in
tshark -i eth0
```



```
root@kali:~# ping www.hackingarticles.in
PING www.hackingarticles.in (104.28.7.89) 56(84) bytes of data.
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=1 ttl=54 time=117 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=2 ttl=54 time=181 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=3 ttl=54 time=249 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=4 ttl=54 time=131 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=5 ttl=54 time=210 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=6 ttl=54 time=161 ms
^C
--- www.hackingarticles.in ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5008ms
rtt min/avg/max/mdev = 117.326/174.819/249.328/45.171 ms
root@kali:~# 
```



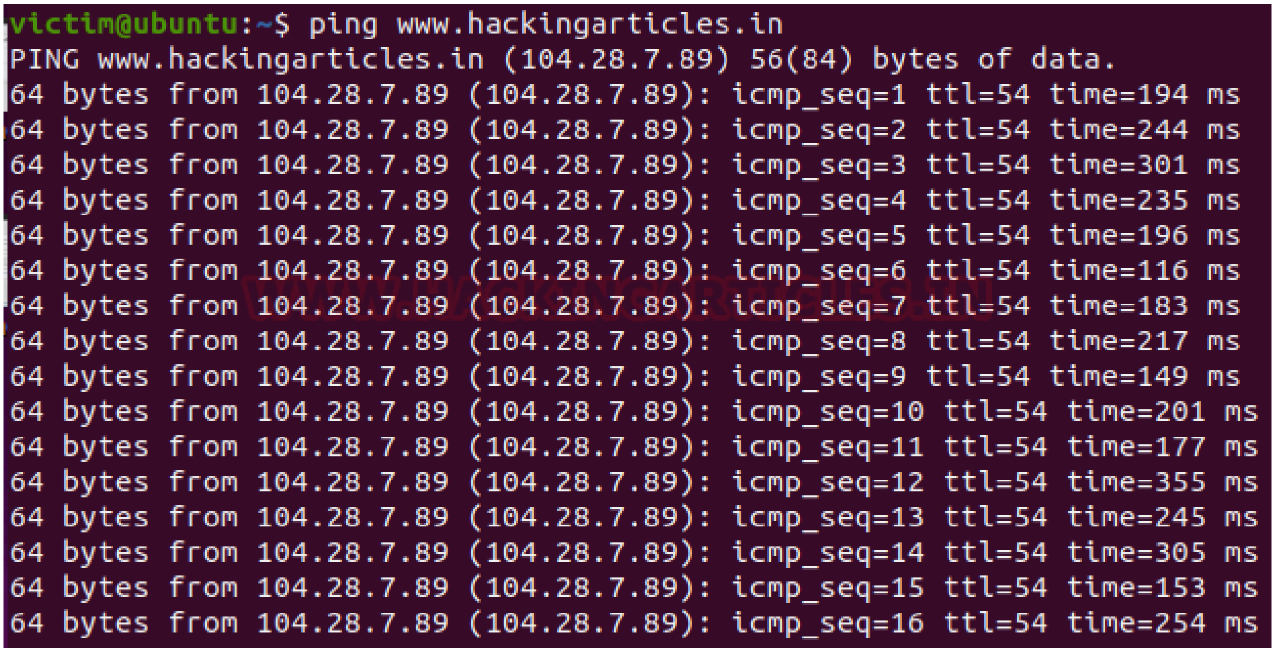
```
root@kali:~# tshark -i eth0
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
    1 0.000000000 192.168.0.137 → 8.8.8.8      DNS 82 Standard query 0x9c1a A www.hackingarticles.in
    2 0.000157409 192.168.0.137 → 8.8.8.8      DNS 82 Standard query 0x9620 AAAA www.hackingarticles.in
    3 0.113177929      8.8.8.8 → 192.168.0.137 DNS 114 Standard query response 0x9c1a A www.hackingarticles.
 A 104.28.7.89 A 104.28.6.89
    4 0.113200970      8.8.8.8 → 192.168.0.137 DNS 138 Standard query response 0x9620 AAAA www.hackingarticl
.in AAAA 2606:4700:3031::681c:759 AAAA 2606:4700:3033::681c:659
    5 0.113563758 192.168.0.137 → 104.28.7.89  ICMP 98 Echo (ping) request  id=0x0aaf, seq=1/256, ttl=64
    6 0.230877040  104.28.7.89 → 192.168.0.137 ICMP 98 Echo (ping) reply    id=0x0aaf, seq=1/256, ttl=54 (re
est in 5)
    7 0.231050335 192.168.0.137 → 8.8.8.8      DNS 84 Standard query 0xf48f PTR 89.7.28.104.in-addr.arpa
    8 0.290869104      8.8.8.8 → 192.168.0.137 DNS 179 Standard query response 0xf48f No such name PTR 89.7.
.104.in-addr.arpa SOA cruz.ns.cloudflare.com
    9 1.115479483 192.168.0.137 → 104.28.7.89  ICMP 98 Echo (ping) request  id=0x0aaf, seq=2/512, ttl=64
   10 1.296199640  104.28.7.89 → 192.168.0.137 ICMP 98 Echo (ping) reply    id=0x0aaf, seq=2/512, ttl=54 (re
est in 9)
   11 2.117862984 192.168.0.137 → 104.28.7.89  ICMP 98 Echo (ping) request  id=0x0aaf, seq=3/768, ttl=64
   12 2.367168921  104.28.7.89 → 192.168.0.137 ICMP 98 Echo (ping) reply    id=0x0aaf, seq=3/768, ttl=54 (re
est in 11)
   13 3.118443326 192.168.0.137 → 104.28.7.89  ICMP 98 Echo (ping) request  id=0x0aaf, seq=4/1024, ttl=64
   14 3.249467028  104.28.7.89 → 192.168.0.137 ICMP 98 Echo (ping) reply    id=0x0aaf, seq=4/1024, ttl=54 (r
uest in 13)
   15 4.120154691 192.168.0.137 → 104.28.7.89  ICMP 98 Echo (ping) request  id=0x0aaf, seq=5/1280, ttl=64
   16 4.330051501  104.28.7.89 → 192.168.0.137 ICMP 98 Echo (ping) reply    id=0x0aaf, seq=5/1280, ttl=54 (r
uest in 15)
   17 5.041217255 Vmware_d5:b7:2d → D-LinkIn_59:e1:24 ARP 42 Who has 192.168.0.1? Tell 192.168.0.137
   18 5.121457089 192.168.0.137 → 104.28.7.89  ICMP 98 Echo (ping) request  id=0x0aaf, seq=6/1536, ttl=64
   19 5.142766220 D-LinkIn_59:e1:24 → Vmware_d5:b7:2d ARP 60 192.168.0.1 is at 1c:5f:2b:59:e1:24
   20 5.281995384  104.28.7.89 → 192.168.0.137 ICMP 98 Echo (ping) reply    id=0x0aaf, seq=6/1536, ttl=54 (r
uest in 18)
```

As we can clearly see it is performing its three-way handshake, then starts the process of ICMP request and reply.

Promiscuous mode

In the networking, promiscuous mode is used as an interface controller that causes tshark to pass all the traffic it receives to the CPU rather than passing the frames to the promiscuous mode is normally used for packet sniffing that can take place on a router or on a computer connected to a wired network or a part of LAN.

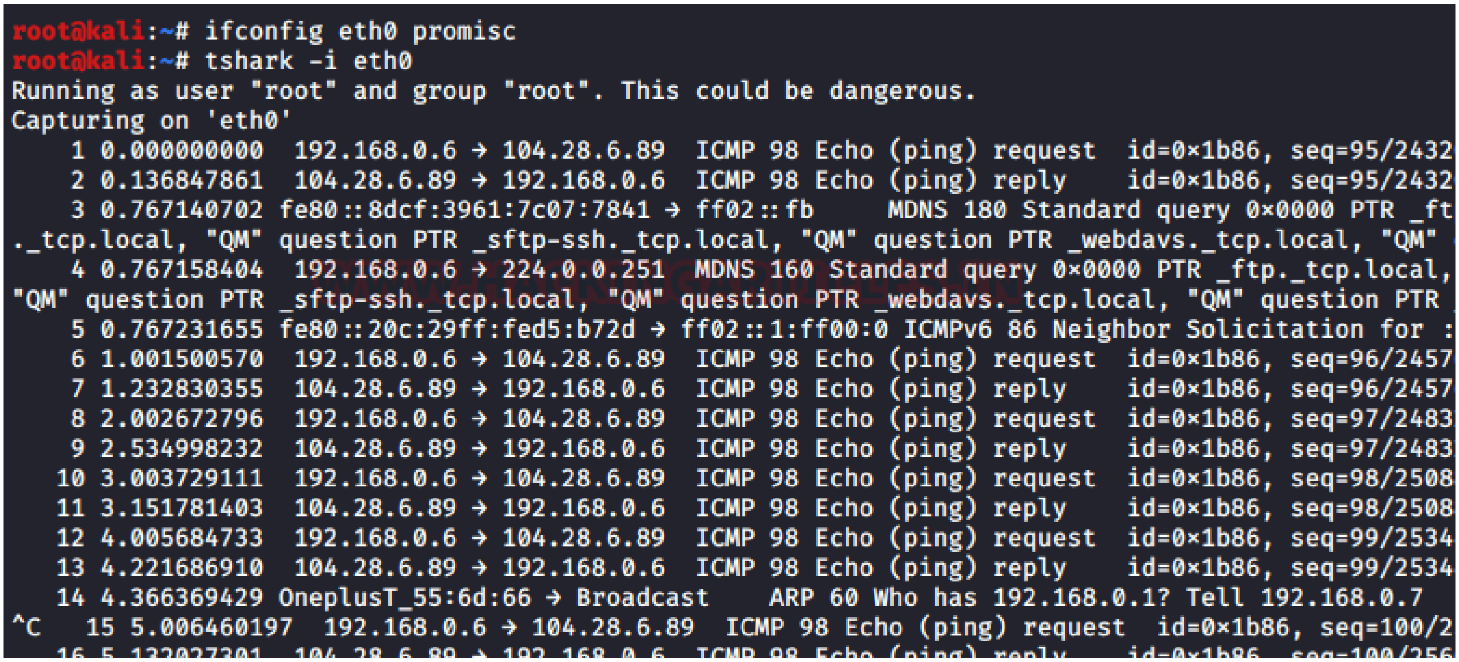
When using this mode, we will need to configure it with the help of ifconfig so that it let us capture the data packets of the whole network. Therefore, we will start by pinging a website and try to capture its data packets.



```
victim@ubuntu:~$ ping www.hackingarticles.in
PING www.hackingarticles.in (104.28.7.89) 56(84) bytes of data.
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=1 ttl=54 time=194 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=2 ttl=54 time=244 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=3 ttl=54 time=301 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=4 ttl=54 time=235 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=5 ttl=54 time=196 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=6 ttl=54 time=116 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=7 ttl=54 time=183 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=8 ttl=54 time=217 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=9 ttl=54 time=149 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=10 ttl=54 time=201 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=11 ttl=54 time=177 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=12 ttl=54 time=355 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=13 ttl=54 time=245 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=14 ttl=54 time=305 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=15 ttl=54 time=153 ms
64 bytes from 104.28.7.89 (104.28.7.89): icmp_seq=16 ttl=54 time=254 ms
```

Now, configure the promiscuous mode by following these commands and try to capture the packets :

```
ifconfig eth0 promisc
tshark -i eth0
```

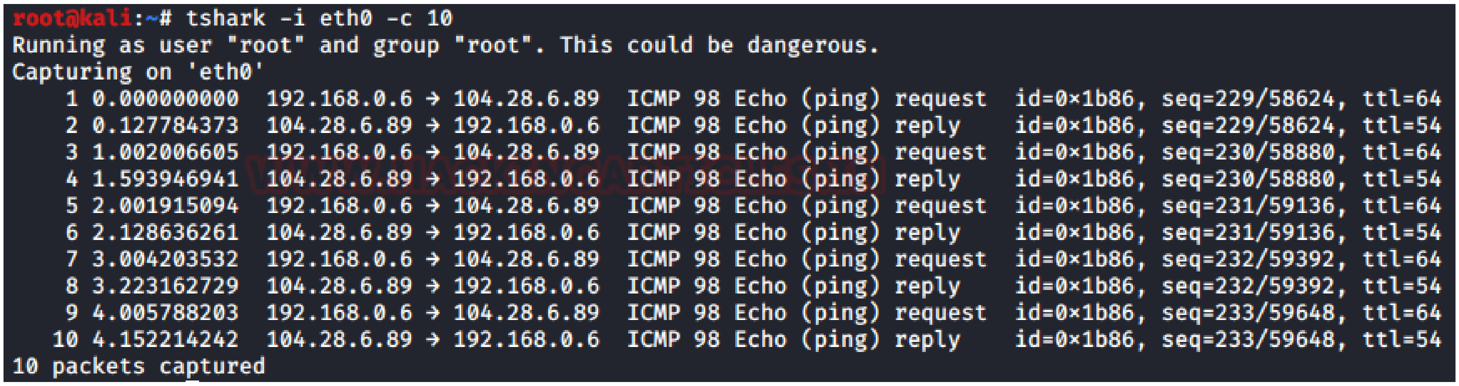


```
root@kali:~# ifconfig eth0 promisc
root@kali:~# tshark -i eth0
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
    1 0.000000000  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0×1b86, seq=95/2432
    2 0.136847861  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0×1b86, seq=95/2432
    3 0.767140702 fe80::8dcf:3961:7c07:7841 → ff02::fb     MDNS 180 Standard query 0×0000 PTR _ft
._tcp.local, "QM" question PTR _sftp-ssh._tcp.local, "QM" question PTR _webdavs._tcp.local, "QM"
    4 0.767158404  192.168.0.6 → 224.0.0.251  MDNS 160 Standard query 0×0000 PTR _ftp._tcp.local,
"QM" question PTR _sftp-ssh._tcp.local, "QM" question PTR _webdavs._tcp.local, "QM" question PTR
    5 0.767231655 fe80::20c:29ff:fed5:b72d → ff02::1:ff00:0 ICMPv6 86 Neighbor Solicitation for :
    6 1.001500570  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0×1b86, seq=96/2457
    7 1.232830355  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0×1b86, seq=96/2457
    8 2.002672796  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0×1b86, seq=97/2483
    9 2.534998232  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0×1b86, seq=97/2483
   10 3.003729111  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0×1b86, seq=98/2508
   11 3.151781403  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0×1b86, seq=98/2508
   12 4.005684733  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0×1b86, seq=99/2534
   13 4.221686910  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0×1b86, seq=99/2534
   14 4.366369429 OneplusT_55:6d:66 → Broadcast    ARP 60 Who has 192.168.0.1? Tell 192.168.0.7
^C   15 5.006460197  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0×1b86, seq=100/2
   16 5.132027301  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0×1b86, seq=100/256
```

Packet count

Tshark has amazing features with which we can work more efficiently and we can access these features using various parameters. One such parameter is '-c', it lets us capture the exact amount of data that we require and it will display only those. This option helps us to refine the outcome of captured traffic.

```
tshark -i eth0 -c 10
```



```
root@kali:~# tshark -i eth0 -c 10
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
    1 0.000000000  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=229/58624, ttl=64
    2 0.127784373  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=229/58624, ttl=54
    3 1.002006605  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=230/58880, ttl=64
    4 1.593946941  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=230/58880, ttl=54
    5 2.001915094  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=231/59136, ttl=64
    6 2.128636261  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=231/59136, ttl=54
    7 3.004203532  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=232/59392, ttl=64
    8 3.223162729  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=232/59392, ttl=54
    9 4.005788203  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=233/59648, ttl=64
   10 4.152214242  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=233/59648, ttl=54
10 packets captured
```

As we can clearly see in the image above that it stops after the 10 counts.

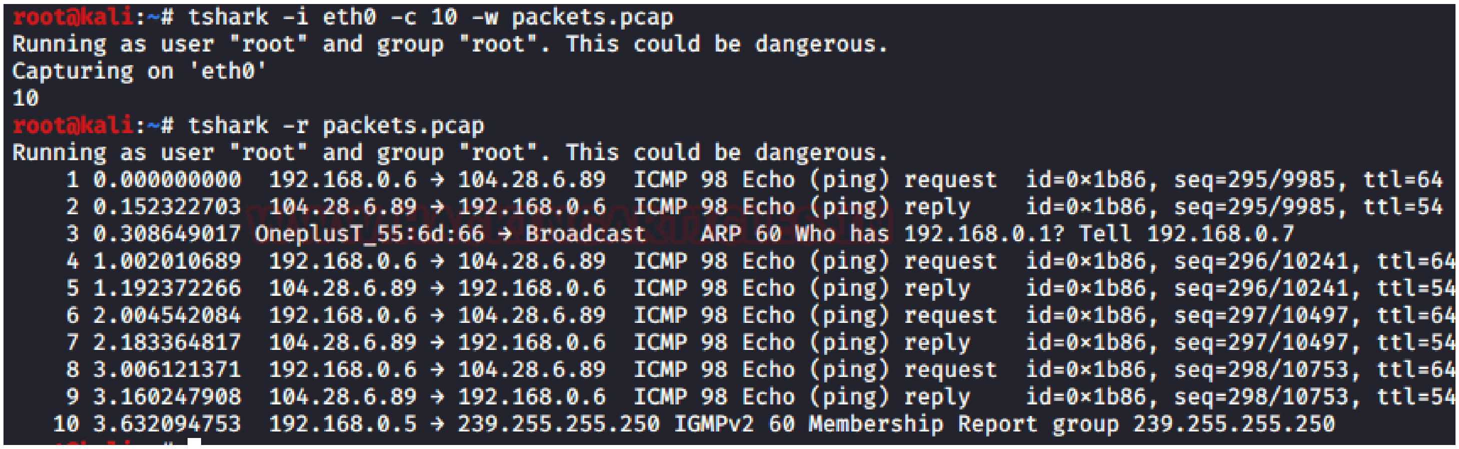
Read and Write in a file

In Tshark we can write and read into .pcap file. Write option (-w) allows us to write raw packet data output to a standard .pcap file whereas read option (-r) help us to read that raw output data packets in our desired manner. To write the packets into a .pcap file use the following command :

```
tshark -i eth0 -c 10 -w packets.pcap
```

And to read the said .pcap file use the following command :

```
tshark -r packets.pcap
```

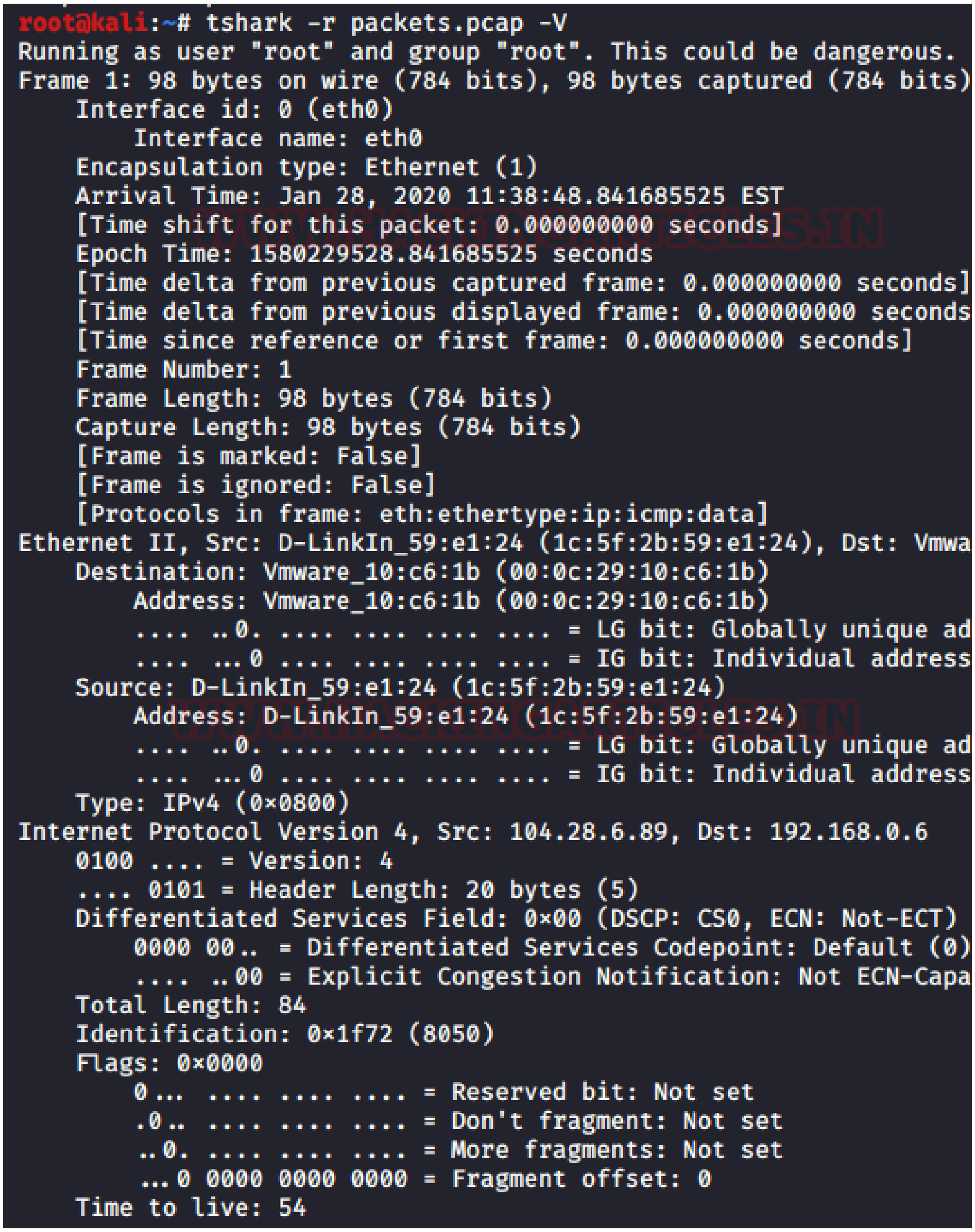


```
root@kali:~# tshark -i eth0 -c 10 -w packets.pcap
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
10
root@kali:~# tshark -r packets.pcap
Running as user "root" and group "root". This could be dangerous.
    1 0.000000000  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=295/9985, ttl=64
    2 0.152322703  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=295/9985, ttl=54
    3 0.308649017 OneplusT_55:6d:66 → Broadcast     ARP 60 Who has 192.168.0.1? Tell 192.168.0.7
    4 1.002010689  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=296/10241, ttl=64
    5 1.192372266  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=296/10241, ttl=54
    6 2.004542084  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=297/10497, ttl=64
    7 2.183364817  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=297/10497, ttl=54
    8 3.006121371  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=298/10753, ttl=64
    9 3.160247908  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=298/10753, ttl=54
   10 3.632094753  192.168.0.5 → 239.255.255.250 IGMPv2 60 Membership Report group 239.255.255.250
```

Verbose mode

The verbose mode provides us with additional details of a packet in traffic. Using the verbose mode, we can see the information that each packet contains and for this option we can use the **parameter -V.**

```
tshark -r packets.pcap -V
```

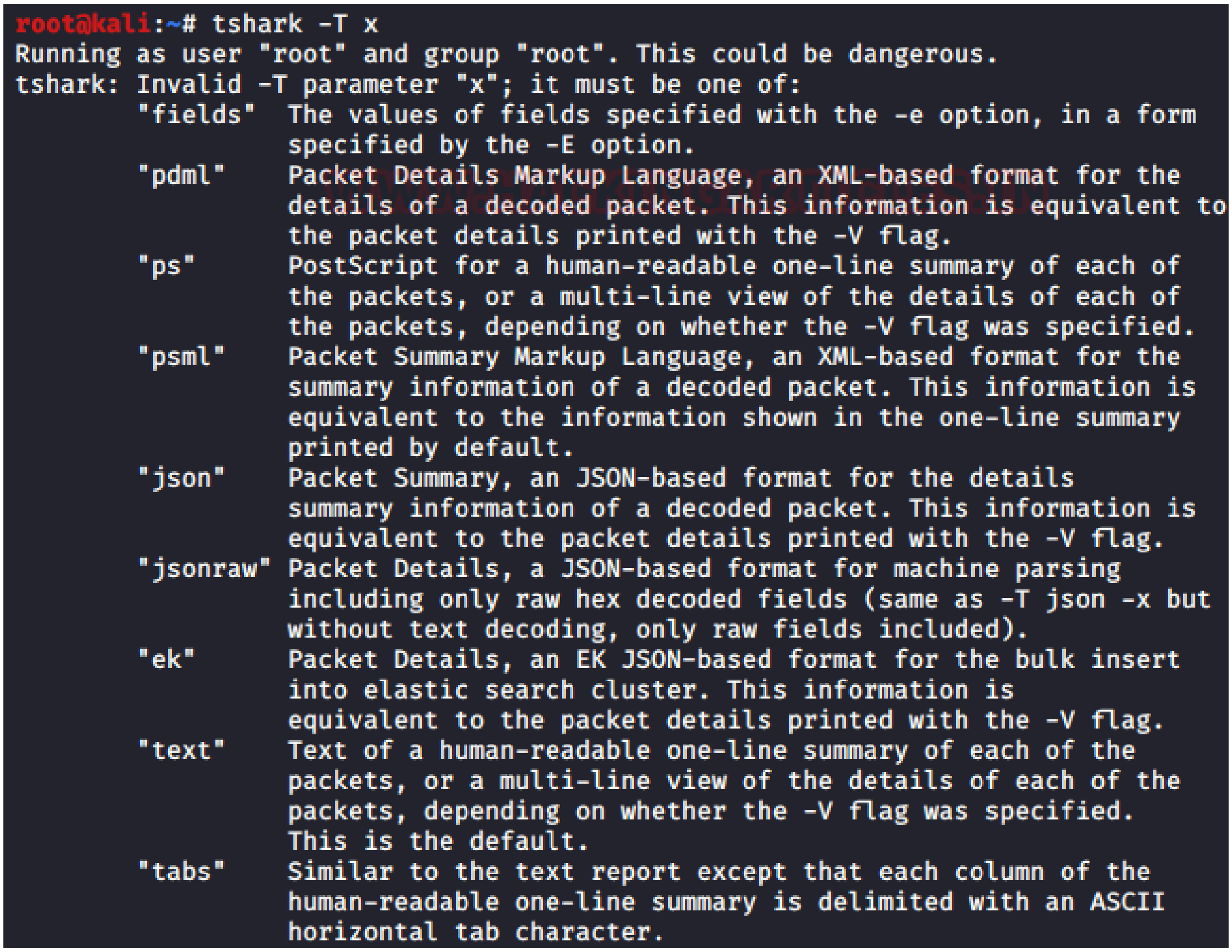


```
root@kali:~# tshark -r packets.pcap -V
Running as user "root" and group "root". This could be dangerous.
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
    Interface id: 0 (eth0)
        Interface name: eth0
    Encapsulation type: Ethernet (1)
    Arrival Time: Jan 28, 2020 11:38:48.841685525 EST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1580229528.841685525 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
Ethernet II, Src: D-LinkIn_59:e1:24 (1c:5f:2b:59:e1:24), Dst: Vmwa
    Destination: Vmware_10:c6:1b (00:0c:29:10:c6:1b)
        Address: Vmware_10:c6:1b (00:0c:29:10:c6:1b)
        .... ..0. .... .... .... .... = LG bit: Globally unique ad
        .... ...0 .... .... .... .... = IG bit: Individual address
    Source: D-LinkIn_59:e1:24 (1c:5f:2b:59:e1:24)
        Address: D-LinkIn_59:e1:24 (1c:5f:2b:59:e1:24)
        .... ..0. .... .... .... .... = LG bit: Globally unique ad
        .... ...0 .... .... .... .... = IG bit: Individual address
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 104.28.6.89, Dst: 192.168.0.6
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capa
    Total Length: 84
    Identification: 0x1f72 (8050)
    Flags: 0x0000
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..0. .... .... .... = More fragments: Not set
        ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 54
```

Output formats

For our convenience, in tshark, we have -T option that lets us save decoded packets in various output formats. It can set the format of the output in the way that it becomes easy to understand. To see all the available options type the following command :

```
tshark -T x
```

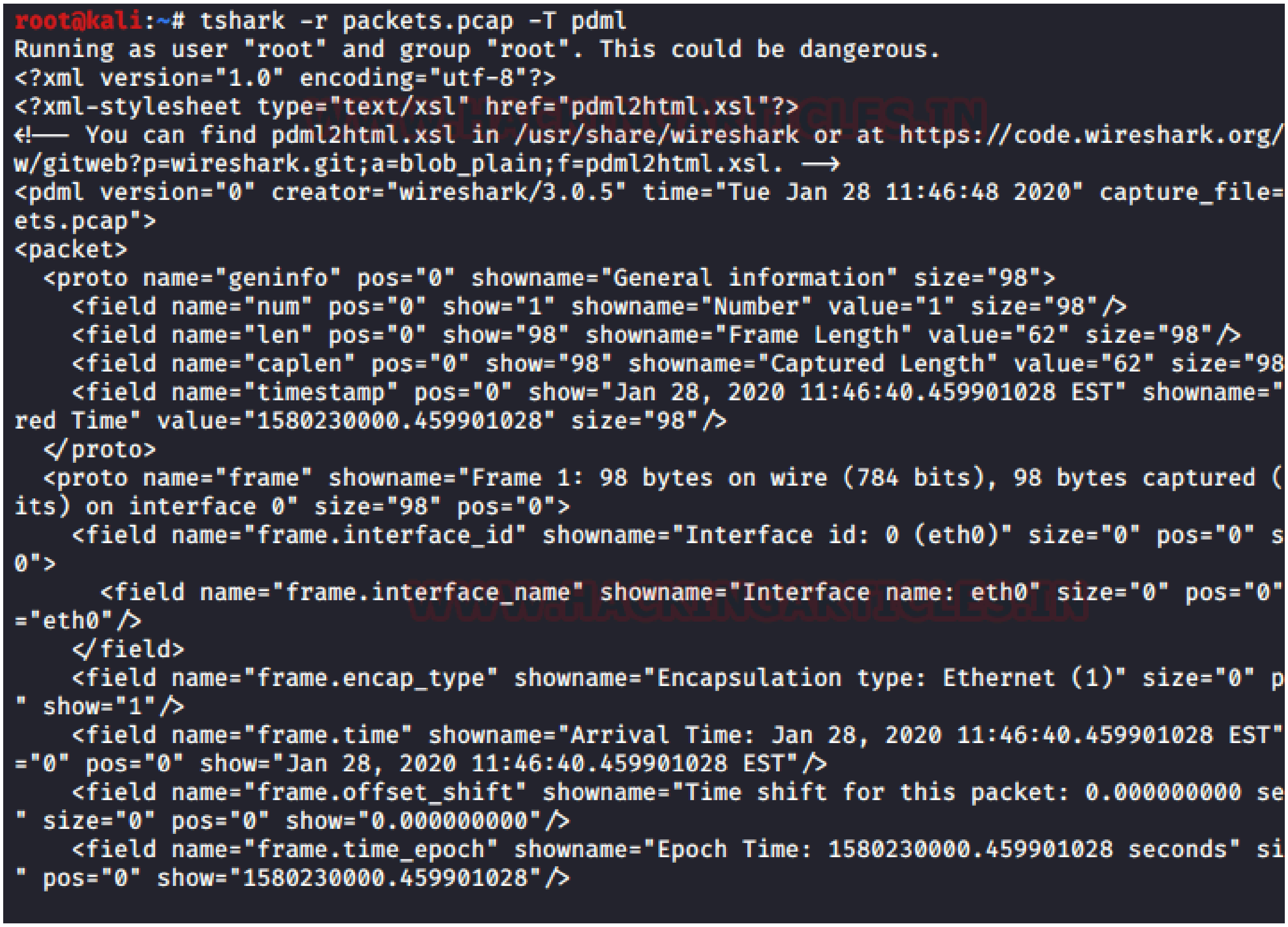


```
root@kali:~# tshark -T x
Running as user "root" and group "root". This could be dangerous.
tshark: Invalid -T parameter "x"; it must be one of:
        "fields"  The values of fields specified with the -e option, in a form
                  specified by the -E option.
        "pdml"    Packet Details Markup Language, an XML-based format for the
                  details of a decoded packet. This information is equivalent to
                  the packet details printed with the -V flag.
        "ps"      PostScript for a human-readable one-line summary of each of
                  the packets, or a multi-line view of the details of each of
                  the packets, depending on whether the -V flag was specified.
        "psml"    Packet Summary Markup Language, an XML-based format for the
                  summary information of a decoded packet. This information is
                  equivalent to the information shown in the one-line summary
                  printed by default.
        "json"    Packet Summary, an JSON-based format for the details
                  summary information of a decoded packet. This information is
                  equivalent to the packet details printed with the -V flag.
        "jsonraw" Packet Details, a JSON-based format for machine parsing
                  including only raw hex decoded fields (same as -T json -x but
                  without text decoding, only raw fields included).
        "ek"      Packet Details, an EK JSON-based format for the bulk insert
                  into elastic search cluster. This information is
                  equivalent to the packet details printed with the -V flag.
        "text"    Text of a human-readable one-line summary of each of the
                  packets, or a multi-line view of the details of each of the
                  packets, depending on whether the -V flag was specified.
                  This is the default.
        "tabs"    Similar to the text report except that each column of the
                  human-readable one-line summary is delimited with an ASCII
                  horizontal tab character.
```

PDML

PDML stands for **Packet Details Mark-Up Language** which is an XML based. This information is quite equivalent to the verbose mode which we used earlier. And to have output in this format type the following command :

```
tshark -r packets.pcap -T pdml
```

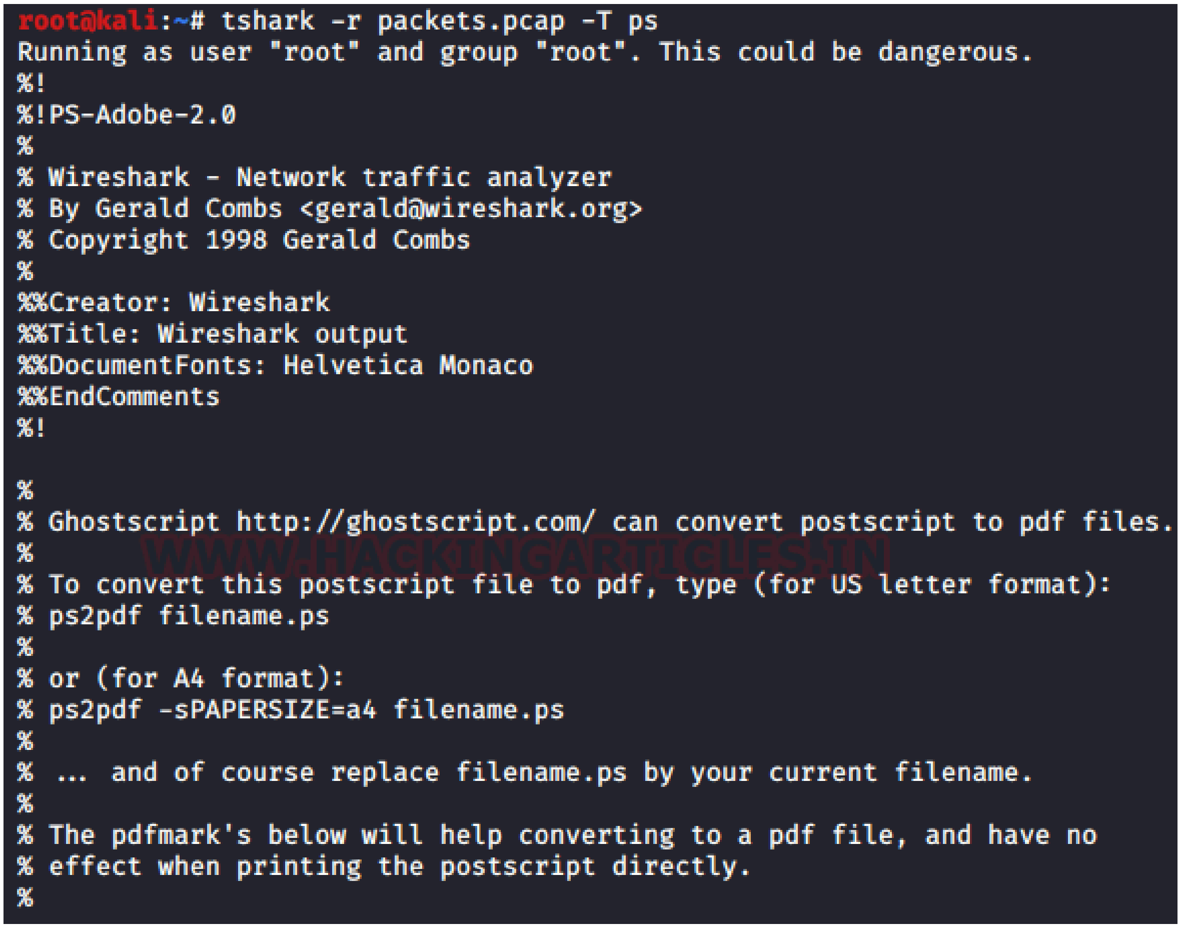


```
root@kali:~# tshark -r packets.pcap -T pdml
Running as user "root" and group "root". This could be dangerous.
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="pdml2html.xsl"?>
<!-- You can find pdml2html.xsl in /usr/share/wireshark or at https://code.wireshark.org/
w/gitweb?p=wireshark.git;a=blob_plain;f=pdml2html.xsl. -->
<pdml version="0" creator="wireshark/3.0.5" time="Tue Jan 28 11:46:48 2020" capture_file=
ets.pcap">
<packet>
  <proto name="geninfo" pos="0" showname="General information" size="98">
    <field name="num" pos="0" show="1" showname="Number" value="1" size="98"/>
    <field name="len" pos="0" show="98" showname="Frame Length" value="62" size="98"/>
    <field name="caplen" pos="0" show="98" showname="Captured Length" value="62" size="98
    <field name="timestamp" pos="0" show="Jan 28, 2020 11:46:40.459901028 EST" showname="
red Time" value="1580230000.459901028" size="98"/>
  </proto>
  <proto name="frame" showname="Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (
its) on interface 0" size="98" pos="0">
    <field name="frame.interface_id" showname="Interface id: 0 (eth0)" size="0" pos="0" s
0">
      <field name="frame.interface_name" showname="Interface name: eth0" size="0" pos="0"
="eth0"/>
    </field>
    <field name="frame.encap_type" showname="Encapsulation type: Ethernet (1)" size="0" p
" show="1"/>
    <field name="frame.time" showname="Arrival Time: Jan 28, 2020 11:46:40.459901028 EST"
="0" pos="0" show="Jan 28, 2020 11:46:40.459901028 EST"/>
    <field name="frame.offset_shift" showname="Time shift for this packet: 0.000000000 se
" size="0" pos="0" show="0.000000000"/>
    <field name="frame.time_epoch" showname="Epoch Time: 1580230000.459901028 seconds" si
" pos="0" show="1580230000.459901028"/>
```

PS

PS stands for **PostScript**. This output is in a form of oneliner summary of each data packets or multi-line detail view of each data packets depending upon each data packet specification. These one-liners are very quick to understand as well as reliable. For this, use the following command :

```
tshark -r packets.pcap -T ps
```

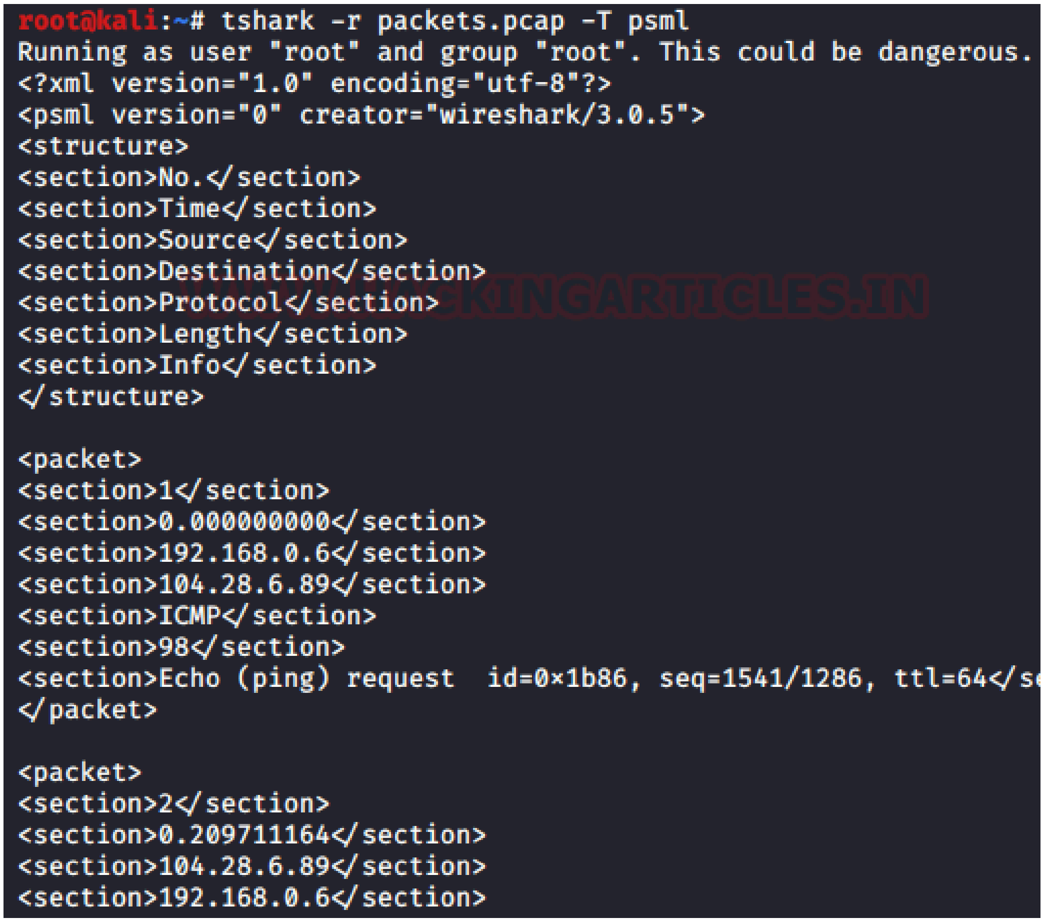


```
root@kali:~# tshark -r packets.pcap -T ps
Running as user "root" and group "root". This could be dangerous.
%!
%!PS-Adobe-2.0
%
% Wireshark - Network traffic analyzer
% By Gerald Combs <gerald@wireshark.org>
% Copyright 1998 Gerald Combs
%
%%Creator: Wireshark
%%Title: Wireshark output
%%DocumentFonts: Helvetica Monaco
%%EndComments
%!

%
% Ghostscript http://ghostscript.com/ can convert postscript to pdf files.
%
% To convert this postscript file to pdf, type (for US letter format):
% ps2pdf filename.ps
%
% or (for A4 format):
% ps2pdf -sPAPERSIZE=a4 filename.ps
%
%  ... and of course replace filename.ps by your current filename.
%
% The pdfmark's below will help converting to a pdf file, and have no
% effect when printing the postscript directly.
%
```

PSML

PSML stands for **Packet Summary Mark-Up Language**. It is also an XML based format like PDML which summarises the detailed information of the packets. And for this format type :

```
tshark -r packets.pcap -T psml
```

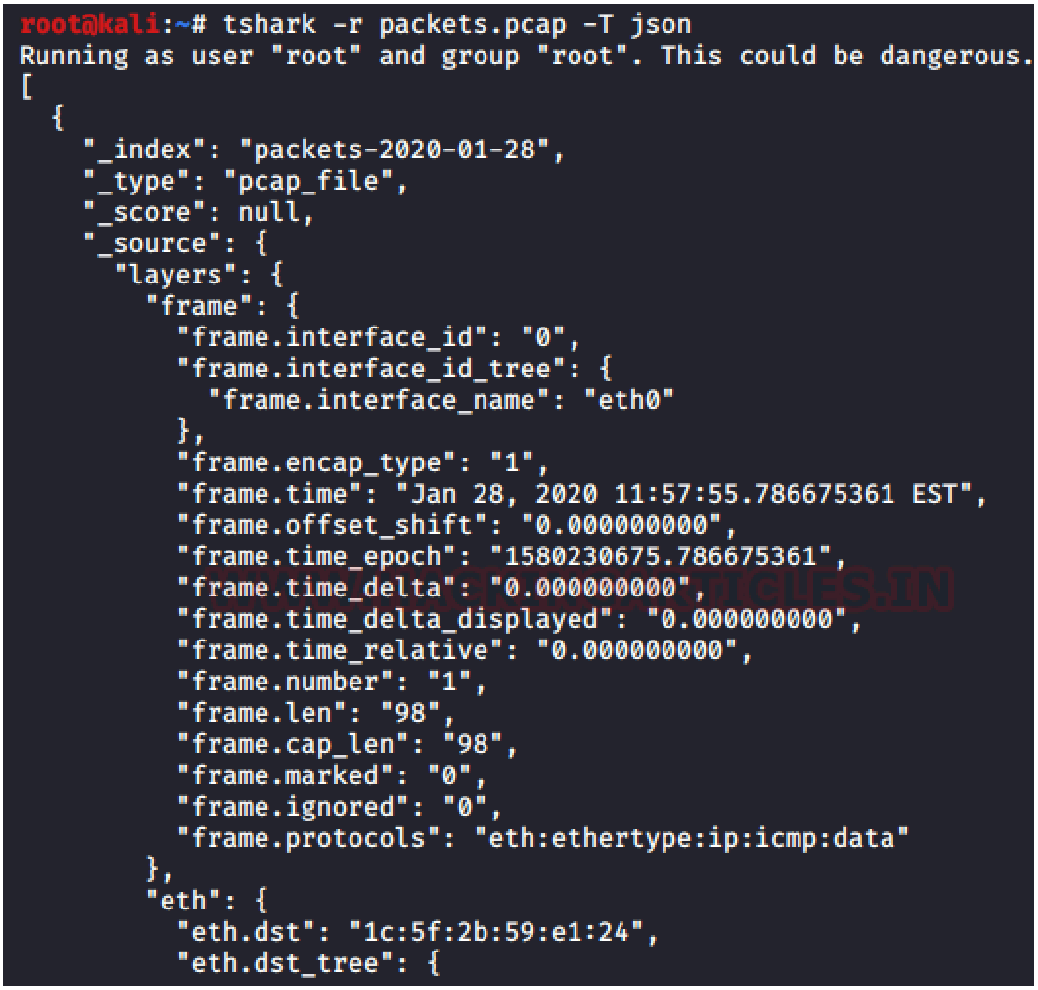


```
root@kali:~# tshark -r packets.pcap -T psml
Running as user "root" and group "root". This could be dangerous.
<?xml version="1.0" encoding="utf-8"?>
<psml version="0" creator="wireshark/3.0.5">
<structure>
<section>No.</section>
<section>Time</section>
<section>Source</section>
<section>Destination</section>
<section>Protocol</section>
<section>Length</section>
<section>Info</section>
</structure>

<packet>
<section>1</section>
<section>0.000000000</section>
<section>192.168.0.6</section>
<section>104.28.6.89</section>
<section>ICMP</section>
<section>98</section>
<section>Echo (ping) request  id=0×1b86, seq=1541/1286, ttl=64</se
</packet>

<packet>
<section>2</section>
<section>0.209711164</section>
<section>104.28.6.89</section>
<section>192.168.0.6</section>
```

JSON

JSON stands for Java-Script Object Notation. It is an open standard file format that displays text in a readable form. The information in this format is fully documented and referred at wolfram. To see that packets in this format, type :

```
tshark -r packets.pcap -T json
```

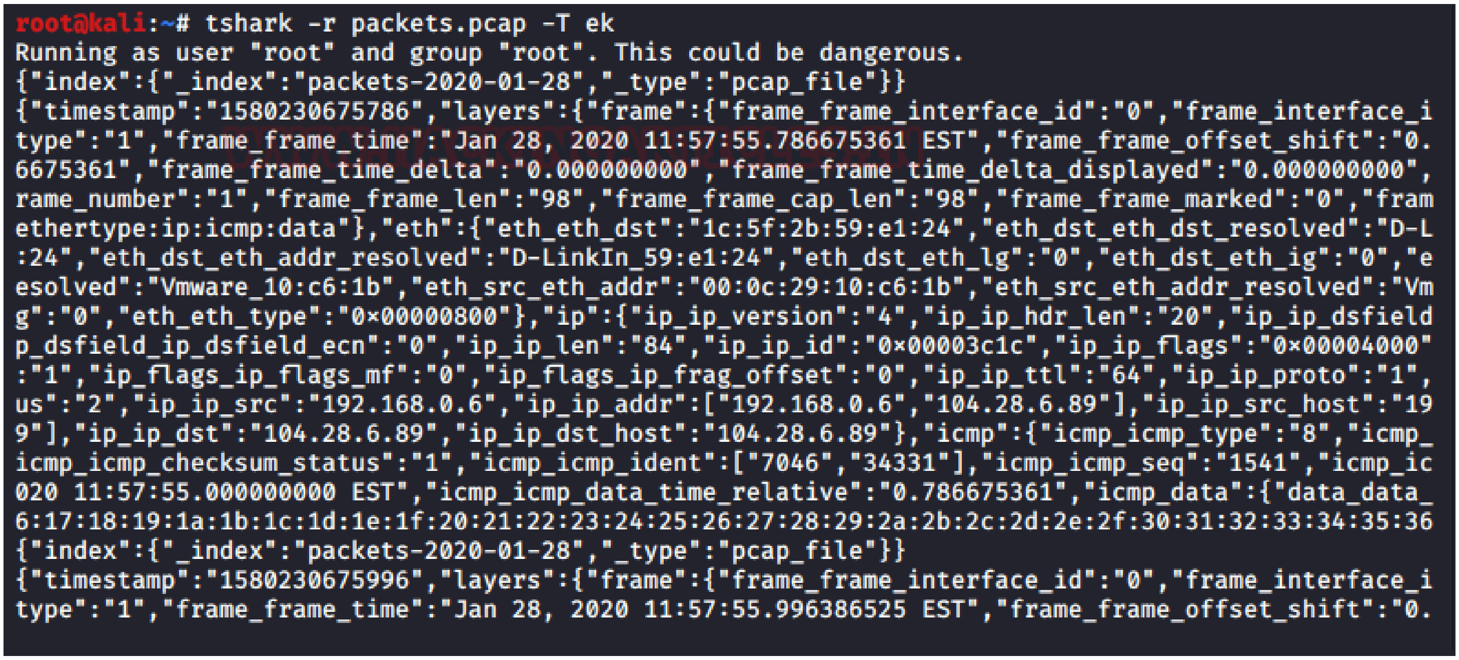


```
root@kali:~# tshark -r packets.pcap -T json
Running as user "root" and group "root". This could be dangerous.
[
  {
    "_index": "packets-2020-01-28",
    "_type": "pcap_file",
    "_score": null,
    "_source": {
      "layers": {
        "frame": {
          "frame.interface_id": "0",
          "frame.interface_id_tree": {
            "frame.interface_name": "eth0"
          },
          "frame.encap_type": "1",
          "frame.time": "Jan 28, 2020 11:57:55.786675361 EST",
          "frame.offset_shift": "0.000000000",
          "frame.time_epoch": "1580230675.786675361",
          "frame.time_delta": "0.000000000",
          "frame.time_delta_displayed": "0.000000000",
          "frame.time_relative": "0.000000000",
          "frame.number": "1",
          "frame.len": "98",
          "frame.cap_len": "98",
          "frame.marked": "0",
          "frame.ignored": "0",
          "frame.protocols": "eth:ethertype:ip:icmp:data"
        },
        "eth": {
          "eth.dst": "1c:5f:2b:59:e1:24",
          "eth.dst_tree": {
```

EK

It is newline delimited JSON format function for bulk import into the elastic search option. And for this format use the following command :

```
tshark -r packets.pcap -T ek
```

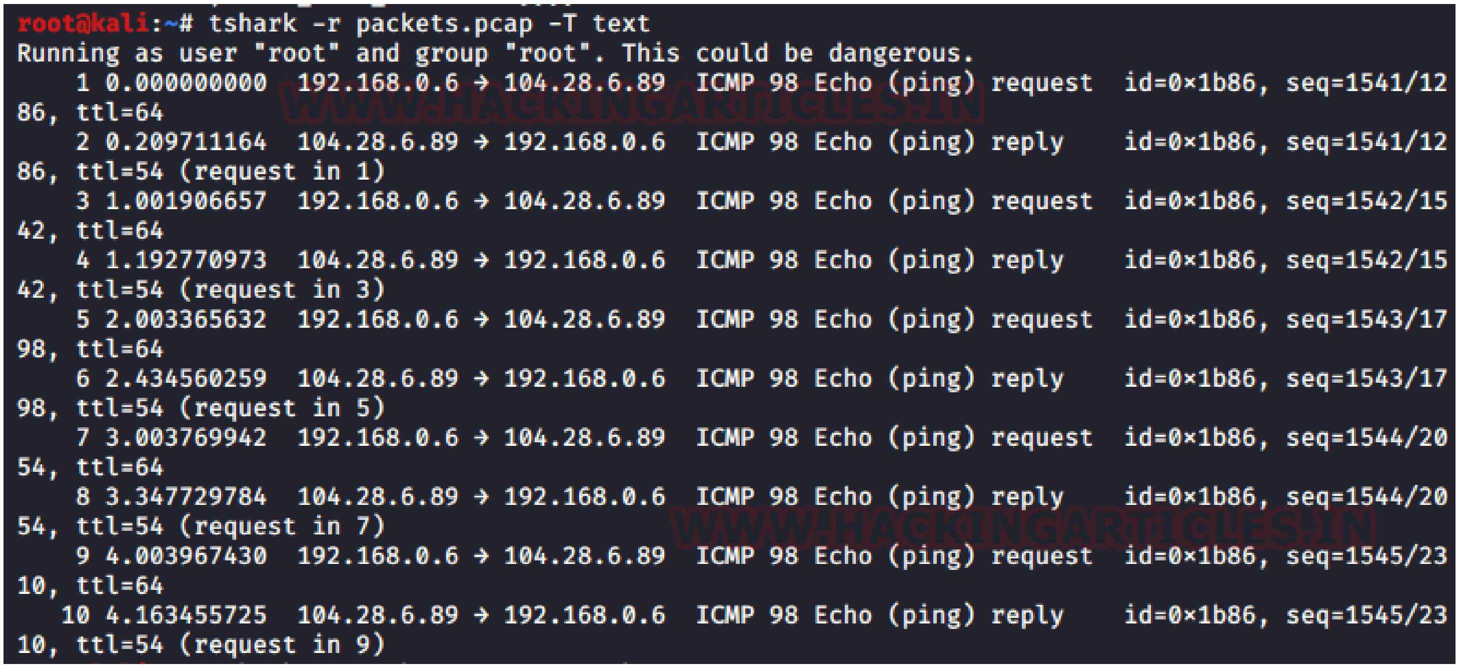


```
root@kali:~# tshark -r packets.pcap -T ek
Running as user "root" and group "root". This could be dangerous.
{"index":{"_index":"packets-2020-01-28","_type":"pcap_file"}}
{"timestamp":"1580230675786","layers":{"frame":{"frame_frame_interface_id":"0","frame_interface_i
type":"1","frame_frame_time":"Jan 28, 2020 11:57:55.786675361 EST","frame_frame_offset_shift":"0.
6675361","frame_frame_time_delta":"0.000000000","frame_frame_time_delta_displayed":"0.000000000",
rame_number":"1","frame_frame_len":"98","frame_frame_cap_len":"98","frame_frame_marked":"0","fram
ethertype:ip:icmp:data"},"eth":{"eth_eth_dst":"1c:5f:2b:59:e1:24","eth_dst_eth_dst_resolved":"D-L
:24","eth_dst_eth_addr_resolved":"D-LinkIn_59:e1:24","eth_dst_eth_lg":"0","eth_dst_eth_ig":"0","e
esolved":"Vmware_10:c6:1b","eth_src_eth_addr":"00:0c:29:10:c6:1b","eth_src_eth_addr_resolved":"Vm
g":"0","eth_eth_type":"0x00000800"},"ip":{"ip_ip_version":"4","ip_ip_hdr_len":"20","ip_ip_dsfield
p_dsfield_ip_dsfield_ecn":"0","ip_ip_len":"84","ip_ip_id":"0x00003c1c","ip_ip_flags":"0x00004000"
:"1","ip_flags_ip_flags_mf":"0","ip_flags_ip_frag_offset":"0","ip_ip_ttl":"64","ip_ip_proto":"1",
us":"2","ip_ip_src":"192.168.0.6","ip_ip_addr":["192.168.0.6","104.28.6.89"],"ip_ip_src_host":"19
9"],"ip_ip_dst":"104.28.6.89","ip_ip_dst_host":"104.28.6.89"},"icmp":{"icmp_icmp_type":"8","icmp_
icmp_icmp_checksum_status":"1","icmp_icmp_ident":["7046","34331"],"icmp_icmp_seq":"1541","icmp_ic
020 11:57:55.000000000 EST","icmp_icmp_data_time_relative":"0.786675361","icmp_data":{"data_data_
6:17:18:19:1a:1b:1c:1d:1e:1f:20:21:22:23:24:25:26:27:28:29:2a:2b:2c:2d:2e:2f:30:31:32:33:34:35:36
{"index":{"_index":"packets-2020-01-28","_type":"pcap_file"}}
{"timestamp":"1580230675996","layers":{"frame":{"frame_frame_interface_id":"0","frame_interface_i
type":"1","frame_frame_time":"Jan 28, 2020 11:57:55.996386525 EST","frame_frame_offset_shift":"0.
```

Text

Text is a human-readable one lines summary of each of the packets. This is the simplest of the formats. And for this, use the following command :

```
tshark -r packets.pcap -T text
```

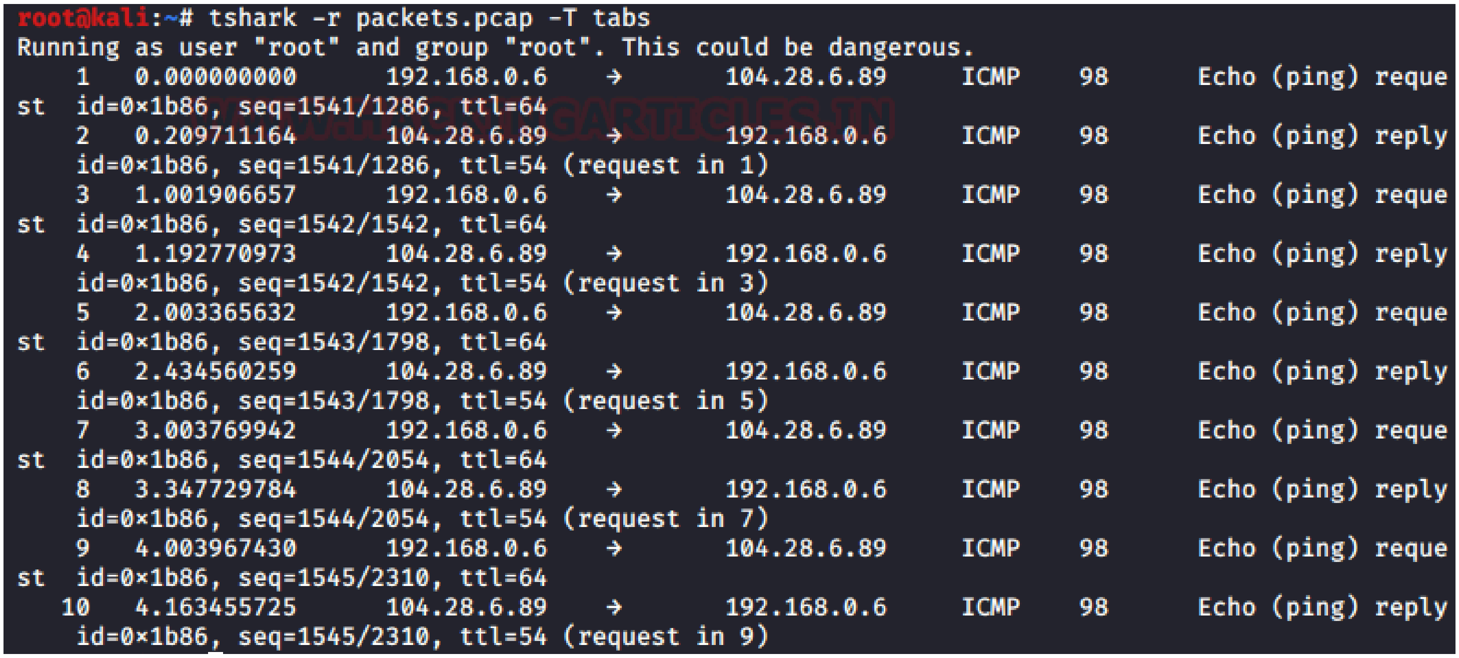


```
root@kali:~# tshark -r packets.pcap -T text
Running as user "root" and group "root". This could be dangerous.
    1 0.000000000  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=1541/12
86, ttl=64
    2 0.209711164  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=1541/12
86, ttl=54 (request in 1)
    3 1.001906657  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=1542/15
42, ttl=64
    4 1.192770973  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=1542/15
42, ttl=54 (request in 3)
    5 2.003365632  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=1543/17
98, ttl=64
    6 2.434560259  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=1543/17
98, ttl=54 (request in 5)
    7 3.003769942  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=1544/20
54, ttl=64
    8 3.347729784  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=1544/20
54, ttl=54 (request in 7)
    9 4.003967430  192.168.0.6 → 104.28.6.89  ICMP 98 Echo (ping) request  id=0x1b86, seq=1545/23
10, ttl=64
   10 4.163455725  104.28.6.89 → 192.168.0.6  ICMP 98 Echo (ping) reply    id=0x1b86, seq=1545/23
10, ttl=54 (request in 9)
```

Tabs

This option is quite similar to the text except, it includes an ASCII horizontal tab (oxo9) character as the delimiter between each column. To try this, type :

```
tshark -r packets.pcap -T tabs
```

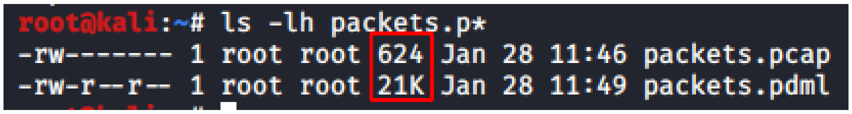


```
root@kali:~# tshark -r packets.pcap -T tabs
Running as user "root" and group "root". This could be dangerous.
    1   0.000000000      192.168.0.6    →       104.28.6.89     ICMP    98      Echo (ping) reque
st  id=0×1b86, seq=1541/1286, ttl=64
    2   0.209711164      104.28.6.89    →       192.168.0.6     ICMP    98      Echo (ping) reply
    id=0×1b86, seq=1541/1286, ttl=54 (request in 1)
    3   1.001906657      192.168.0.6    →       104.28.6.89     ICMP    98      Echo (ping) reque
st  id=0×1b86, seq=1542/1542, ttl=64
    4   1.192770973      104.28.6.89    →       192.168.0.6     ICMP    98      Echo (ping) reply
    id=0×1b86, seq=1542/1542, ttl=54 (request in 3)
    5   2.003365632      192.168.0.6    →       104.28.6.89     ICMP    98      Echo (ping) reque
st  id=0×1b86, seq=1543/1798, ttl=64
    6   2.434560259      104.28.6.89    →       192.168.0.6     ICMP    98      Echo (ping) reply
    id=0×1b86, seq=1543/1798, ttl=54 (request in 5)
    7   3.003769942      192.168.0.6    →       104.28.6.89     ICMP    98      Echo (ping) reque
st  id=0×1b86, seq=1544/2054, ttl=64
    8   3.347729784      104.28.6.89    →       192.168.0.6     ICMP    98      Echo (ping) reply
    id=0×1b86, seq=1544/2054, ttl=54 (request in 7)
    9   4.003967430      192.168.0.6    →       104.28.6.89     ICMP    98      Echo (ping) reque
st  id=0×1b86, seq=1545/2310, ttl=64
   10   4.163455725      104.28.6.89    →       192.168.0.6     ICMP    98      Echo (ping) reply
    id=0×1b86, seq=1545/2310, ttl=54 (request in 9)
```

Difference between decoded packets and encoded packets

When we try to write the live data packets in a .pcap format file; we compress all that data packets in smaller segments. To better understand these data packets we need to decode them which leads to a difference in the size of the file and to check the size of any given file at the given moment use the following command :

```
ls -lh packets.p*
```



```
root@kali:~# ls -lh packets.p*
-rw------- 1 root root 624 Jan 28 11:46 packets.pcap
-rw-r--r-- 1 root root 21K Jan 28 11:49 packets.pdml
```

Like we discussed there is a huge difference in these files, that’s why we use decoding techniques to extract this information.

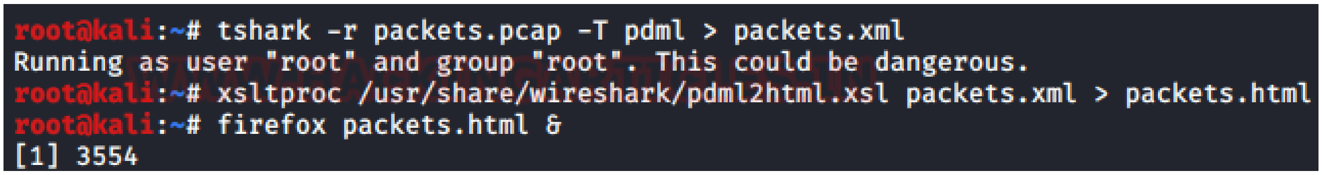
Converting PDML file HTML page

The only difference between the Wireshark and tshark is that Wireshark is a GUI based tool and tshark is a command-line based tool. But with the help of some external source, we can also view our data packets in HTML. So to achieve that first, we need to save our data packets in PDML format and then convert it into an XML file using the following command :

```
tshark -r packets.pcap -T pdml > packets.xml
```

The XML file will be saved at location **/usr/share/wireshark/pdml2html.xsl.** So, we are going to use **xsltproc tool** to execute this file it which will help us to create our HTML page. Creating the HTML page will format all the unnecessary information and only let us view the usable data. To create the HTML use following command

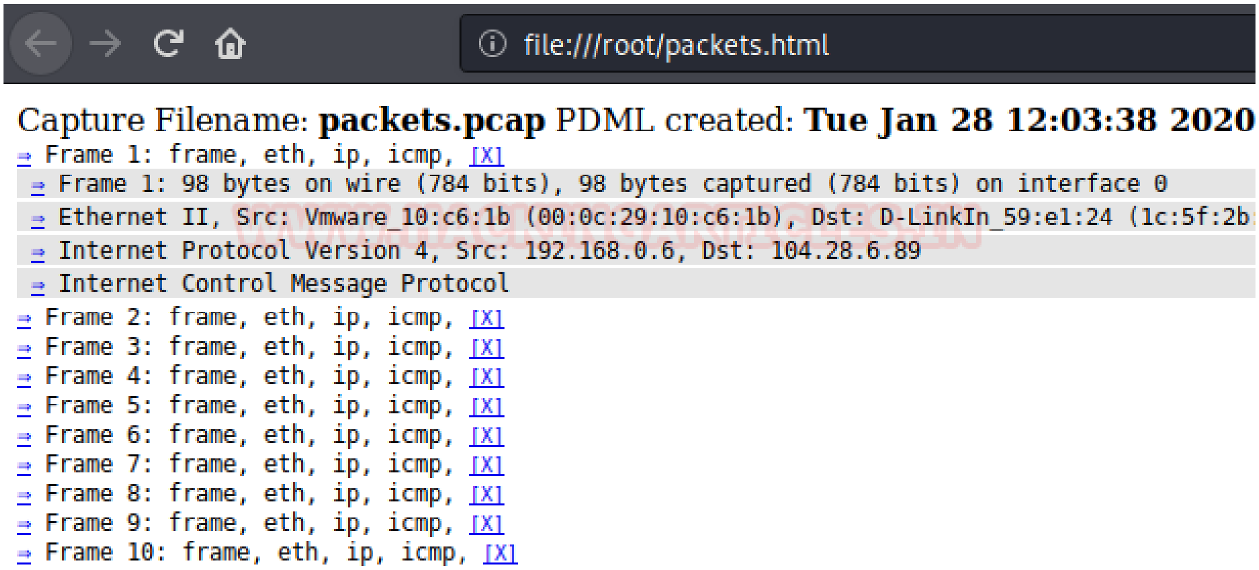
```
xsltproc /usr/share/wireshark/pdml2html.xsl packets.xml > packets.html
```



```
root@kali:~# tshark -r packets.pcap -T pdml > packets.xml
Running as user "root" and group "root". This could be dangerous.
root@kali:~# xsltproc /usr/share/wireshark/pdml2html.xsl packets.xml > packets.html
root@kali:~# firefox packets.html &
[1] 3554
```

To open the HTML page in the browser, refer to the above image and use the following command :

```
firefox packets.html &
```



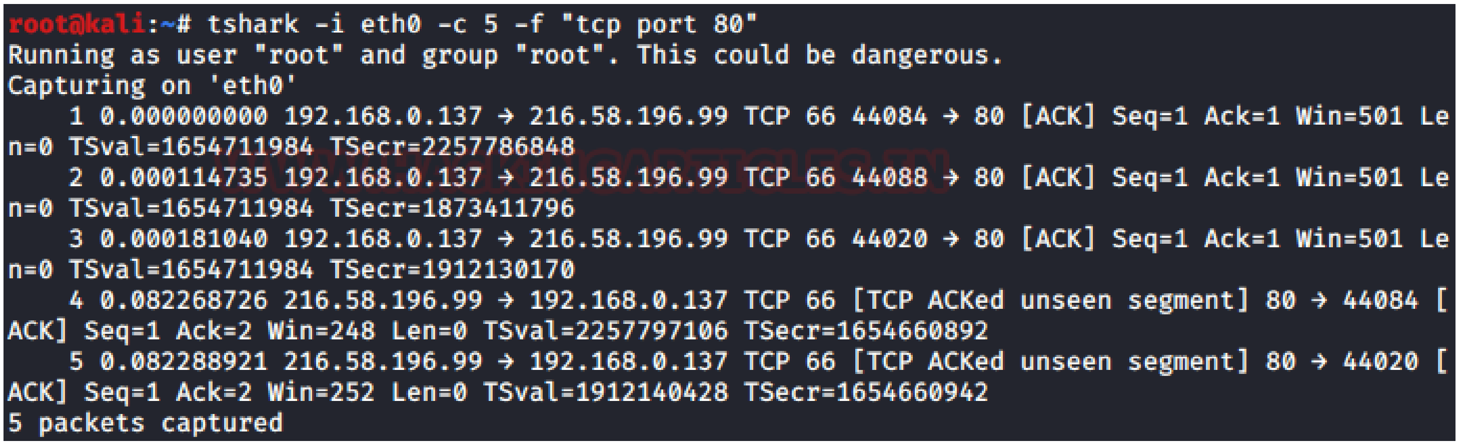
Capture Filename: **packets.pcap** PDML created: **Tue Jan 28 12:03:38 2020**

```
⇒ Frame 1: frame, eth, ip, icmp, [X]
  ⇒ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
  ⇒ Ethernet II, Src: Vmware_10:c6:1b (00:0c:29:10:c6:1b), Dst: D-LinkIn_59:e1:24 (1c:5f:2b
  ⇒ Internet Protocol Version 4, Src: 192.168.0.6, Dst: 104.28.6.89
  ⇒ Internet Control Message Protocol
⇒ Frame 2: frame, eth, ip, icmp, [X]
⇒ Frame 3: frame, eth, ip, icmp, [X]
⇒ Frame 4: frame, eth, ip, icmp, [X]
⇒ Frame 5: frame, eth, ip, icmp, [X]
⇒ Frame 6: frame, eth, ip, icmp, [X]
⇒ Frame 7: frame, eth, ip, icmp, [X]
⇒ Frame 8: frame, eth, ip, icmp, [X]
⇒ Frame 9: frame, eth, ip, icmp, [X]
⇒ Frame 10: frame, eth, ip, icmp, [X]
```

Capturing packets of a particular port

A lot of times we use Wireshark on a dedicated port. And by using the -f option we can capture data packets of a particular port. It helps us to better analyze the data packets of the network. We are using this feature to capture TCP port 80 and the command for this is :

```
tshark -i eth0 -c 5 -f "tcp port 80"
```



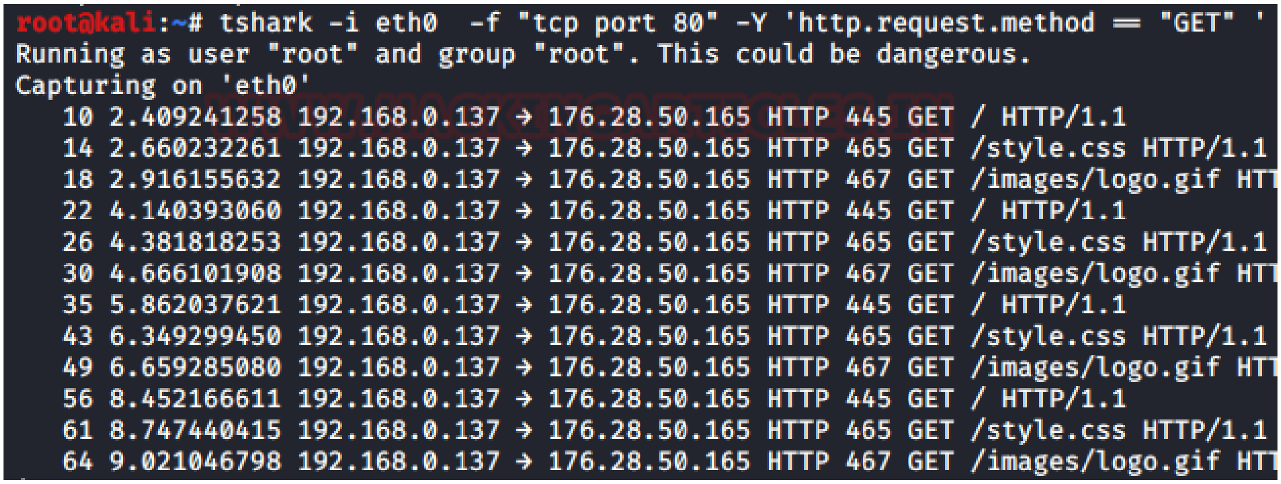
```
root@kali:~# tshark -i eth0 -c 5 -f "tcp port 80"
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
    1 0.000000000 192.168.0.137 → 216.58.196.99 TCP 66 44084 → 80 [ACK] Seq=1 Ack=1 Win=501 Le
n=0 TSval=1654711984 TSecr=2257786848
    2 0.000114735 192.168.0.137 → 216.58.196.99 TCP 66 44088 → 80 [ACK] Seq=1 Ack=1 Win=501 Le
n=0 TSval=1654711984 TSecr=1873411796
    3 0.000181040 192.168.0.137 → 216.58.196.99 TCP 66 44020 → 80 [ACK] Seq=1 Ack=1 Win=501 Le
n=0 TSval=1654711984 TSecr=1912130170
    4 0.082268726 216.58.196.99 → 192.168.0.137 TCP 66 [TCP ACKed unseen segment] 80 → 44084 [
ACK] Seq=1 Ack=2 Win=248 Len=0 TSval=2257797106 TSecr=1654660892
    5 0.082288921 216.58.196.99 → 192.168.0.137 TCP 66 [TCP ACKed unseen segment] 80 → 44020 [
ACK] Seq=1 Ack=2 Win=252 Len=0 TSval=1912140428 TSecr=1654660942
5 packets captured
```

Display filter

Display filter was introduced by Wireshark. It helps us to filter the captured data packets or live data packets. With the help of this filter, we can request for any kind of filter that we want to capture in the live environment.

In our scenario, we apply the GET request filter to capture only GET request from the traffic and for, use the following command :

```
tshark -i eth0 -c 5 -f "tcp port 80" -Y 'http.request.method == "GET" '
```



```
root@kali:~# tshark -i eth0  -f "tcp port 80" -Y 'http.request.method == "GET" '
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
   10 2.409241258 192.168.0.137 → 176.28.50.165 HTTP 445 GET / HTTP/1.1
   14 2.660232261 192.168.0.137 → 176.28.50.165 HTTP 465 GET /style.css HTTP/1.1
   18 2.916155632 192.168.0.137 → 176.28.50.165 HTTP 467 GET /images/logo.gif HTT
   22 4.140393060 192.168.0.137 → 176.28.50.165 HTTP 445 GET / HTTP/1.1
   26 4.381818253 192.168.0.137 → 176.28.50.165 HTTP 465 GET /style.css HTTP/1.1
   30 4.666101908 192.168.0.137 → 176.28.50.165 HTTP 467 GET /images/logo.gif HTT
   35 5.862037621 192.168.0.137 → 176.28.50.165 HTTP 445 GET / HTTP/1.1
   43 6.349299450 192.168.0.137 → 176.28.50.165 HTTP 465 GET /style.css HTTP/1.1
   49 6.659285080 192.168.0.137 → 176.28.50.165 HTTP 467 GET /images/logo.gif HTT
   56 8.452166611 192.168.0.137 → 176.28.50.165 HTTP 445 GET / HTTP/1.1
   61 8.747440415 192.168.0.137 → 176.28.50.165 HTTP 465 GET /style.css HTTP/1.1
   64 9.021046798 192.168.0.137 → 176.28.50.165 HTTP 467 GET /images/logo.gif HTT
```

Conclusion

This article focuses on the basic commands and functionality of tshark as it is the first article in the series. So get yourself familiar with the features of it as and stay tuned for the advance features of tshark in our next article.