

# COMPREHENSIVE GUIDE ON **TSHARK**



# Table of Contents

---

<b>Abstract.....</b>	<b>4</b>
<b>Network Traffic.....</b>	<b>5</b>
<b>Introduction to TShark.....</b>	<b>5</b>
<b>List Interfaces.....</b>	<b>7</b>
<b>Capture Traffic.....</b>	<b>7</b>
<b>Promiscuous mode.....</b>	<b>8</b>
<b>Packet Count.....</b>	<b>9</b>
<b>Read and Write a File.....</b>	<b>10</b>
<b>Verbose Mode.....</b>	<b>11</b>
<b>Output Formats.....</b>	<b>12</b>
<b>PDML .....</b>	<b>12</b>
<b>PS .....</b>	<b>13</b>
<b>PSML .....</b>	<b>13</b>
<b>JSON .....</b>	<b>14</b>
<b>EK .....</b>	<b>15</b>
<b>Text .....</b>	<b>15</b>
<b>Tabs .....</b>	<b>16</b>
<b>Difference between decoded and encoded packets.....</b>	<b>16</b>
<b>Converting PDML file to HTML Page.....</b>	<b>17</b>
<b>Capturing Packets of a Particular Port.....</b>	<b>18</b>
<b>Display Filter.....</b>	<b>18</b>
<b>Statistical Options.....</b>	<b>19</b>
<b>Protocol Hierarchy Statistics.....</b>	<b>20</b>
<b>Read Filter Analysis.....</b>	<b>21</b>
<b>Endpoint Analysis.....</b>	<b>22</b>
<b>Conversation Analysis.....</b>	<b>23</b>
<b>Expert Mode Analysis.....</b>	<b>23</b>
<b>Packet Distribution Tree.....</b>	<b>24</b>
<b>Packet Length Tree.....</b>	<b>25</b>
<b>Colour Based Output Analysis.....</b>	<b>25</b>
<b>Ring Buffer Analysis.....</b>	<b>26</b>

<b>Autostop</b> .....	<b>27</b>
<b>Duration</b> .....	<b>27</b>
<b>File size</b> .....	<b>27</b>
<b>Data-link Types</b> .....	<b>28</b>
<b>Version Information</b> .....	<b>28</b>
<b>Reporting Options</b> .....	<b>29</b>
<b>Column Formats</b> .....	<b>29</b>
<b>Decodes</b> .....	<b>30</b>
<b>Dissector Tables</b> .....	<b>31</b>
<b>Elastic Mapping</b> .....	<b>32</b>
<b>Field Count</b> .....	<b>32</b>
<b>Fields</b> .....	<b>33</b>
<b>Fundamental Types</b> .....	<b>33</b>
<b>Heuristic Decodes</b> .....	<b>34</b>
<b>Plugins</b> .....	<b>35</b>
<b>Protocols</b> .....	<b>36</b>
<b>Values</b> .....	<b>36</b>
<b>Preferences</b> .....	<b>37</b>
<b>Folders</b> .....	<b>37</b>
<b>PyShark</b> .....	<b>38</b>
<b>Installation</b> .....	<b>38</b>
<b>Live Capture</b> .....	<b>39</b>
<b>Pretty Representation</b> .....	<b>39</b>
<b>Captured Length Field</b> .....	<b>41</b>
<b>Layers, Src and Dst Fields</b> .....	<b>41</b>
<b>Promisc Capture</b> .....	<b>42</b>
<b>References</b> .....	<b>43</b>
<b>About Us</b> .....	<b>44</b>

# Abstract

---

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

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

We will understand the reporting functionalities and some additional tricks that we found while tinkering with TShark.

# 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. Analysing 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 analysing. 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 analyser. It is developed by Wireshark. Its 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 analyse real-time network traffic and it can read .pcap files to analyse 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
```

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

```

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=0x0aaaf, seq=1/256, ttl=64
  6 0.230877040 104.28.7.89 > 192.168.0.137 ICMP 98 Echo (ping) reply id=0x0aaaf, 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=0x0aaaf, seq=2/512, ttl=64
 10 1.296199640 104.28.7.89 > 192.168.0.137 ICMP 98 Echo (ping) reply id=0x0aaaf, 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=0x0aaaf, seq=3/768, ttl=64
 12 2.367168921 104.28.7.89 > 192.168.0.137 ICMP 98 Echo (ping) reply id=0x0aaaf, 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=0x0aaaf, seq=4/1024, ttl=64
 14 3.249467028 104.28.7.89 > 192.168.0.137 ICMP 98 Echo (ping) reply id=0x0aaaf, 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=0x0aaaf, seq=5/1280, ttl=64
 16 4.330051501 104.28.7.89 > 192.168.0.137 ICMP 98 Echo (ping) reply id=0x0aaaf, 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=0x0aaaf, 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=0x0aaaf, seq=6/1536, ttl=54 (r
uest in 18)

```

## 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=0x1b86, seq=95/2432
  2 0.136847861 104.28.6.89 → 192.168.0.6 ICMP 98 Echo (ping) reply id=0x1b86, seq=95/2432
  3 0.767140702 fe80::8dcf:3961:7c07:7841 → ff02::fb MDNS 180 Standard query 0x0000 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 0x0000 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=0x1b86, seq=96/2457
  7 1.232830355 104.28.6.89 → 192.168.0.6 ICMP 98 Echo (ping) reply id=0x1b86, seq=96/2457
  8 2.002672796 192.168.0.6 → 104.28.6.89 ICMP 98 Echo (ping) request id=0x1b86, seq=97/2483
  9 2.534998232 104.28.6.89 → 192.168.0.6 ICMP 98 Echo (ping) reply id=0x1b86, seq=97/2483
 10 3.003729111 192.168.0.6 → 104.28.6.89 ICMP 98 Echo (ping) request id=0x1b86, seq=98/2508
 11 3.151781403 104.28.6.89 → 192.168.0.6 ICMP 98 Echo (ping) reply id=0x1b86, seq=98/2508
 12 4.005684733 192.168.0.6 → 104.28.6.89 ICMP 98 Echo (ping) request id=0x1b86, seq=99/2534
 13 4.221686910 104.28.6.89 → 192.168.0.6 ICMP 98 Echo (ping) reply id=0x1b86, seq=99/2534
 14 4.366369429 OnePlusT_55:6d:66 → Broadcast ARP 60 Who has 192.168.0.1? Tell 192.168.0.7
 15 5.006460197 192.168.0.6 → 104.28.6.89 ICMP 98 Echo (ping) request id=0x1b86, seq=100/255
 16 5.122027201 104.28.6.89 → 192.168.0.6 ICMP 98 Echo (ping) reply id=0x1b86 seq=100/255
```

## 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 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"/>
```

## PS

---

PS stands for **PostScript**. This output is in a form of one-liner 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
%!
```

## 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>
```

## 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.type": "0x0800"
        }
      }
    }
  }
]
```

## 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_interface_id":0,"frame_interface_i
type":1,"frame_frame_time":Jan 28, 2020 11:57:55.786675361 EST,"frame_offset_shift":0.
6675361,"frame_frame_time_delta":0.000000000,"frame_frame_time_delta_displayed":0.000000000,
"frame_number":1,"frame_frame_len":98,"frame_frame_cap_len":98,"frame_frame_marked":0,"fram
etheretype:ip:icmp:data"}, "eth": {"eth_dst": "1c:5f:2b:59:e1:24", "eth_dst_ether_resolved": "D-L
:24", "eth_dst_ether_addr_resolved": "D-LinkIn_59:e1:24", "eth_dst_ether_lq": 0, "eth_dst_ether_ir": 0, "e
resolved": "Vmware_10:c6:1b", "eth_src": "00:0c:29:10:c6:1b", "eth_src_ether_addr_resolved": "Vm
g": 0, "eth_ether_type": "0x00000800", "ip": {"ip_ip_version": 4, "ip_ip_hdr_len": 20, "ip_ip_d
dsfield_ip_dsfield_ecn": 0, "ip_ip_len": 84, "ip_ip_id": 0x0003c1c, "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, "i
us": 2, "ip_ip_src": "192.168.0.6", "ip_ip_addr": ["192.168.0.6", "104.28.6.89"], "ip_ip_src_ho
st": 19
9}, "ip_ip_dst": "104.28.6.89", "ip_ip_dst_host": "104.28.6.89"}, "icmp": {"icmp_icmp_type": 8, "ic
mp_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_da
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
f"index": "packets-2020-01-28", "type": "pcap_file"}}}
```

## 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=0xb86, seq=1541/12
 86, ttl=64
 2 0.209711164 104.28.6.89 > 192.168.0.6 ICMP 98 Echo (ping) reply      id=0xb86, 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=0xb86, seq=1542/15
 42, ttl=64
 4 1.92770973 104.28.6.89 > 192.168.0.6 ICMP 98 Echo (ping) reply      id=0xb86, 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=0xb86, seq=1543/17
 98, ttl=64
 6 2.434560259 104.28.6.89 > 192.168.0.6 ICMP 98 Echo (ping) reply      id=0xb86, 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=0xb86, seq=1544/20
 54, ttl=64
 8 3.347729784 104.28.6.89 > 192.168.0.6 ICMP 98 Echo (ping) reply      id=0xb86, 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=0xb86, seq=1545/23
10, ttl=64
10 4.163455725 104.28.6.89 > 192.168.0.6 ICMP 98 Echo (ping) reply      id=0xb86, 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 (ox09) 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) request
st id=0x1b86, seq=1541/1286, ttl=64
  2  0.209711164    104.28.6.89  →      192.168.0.6    ICMP   98    Echo (ping) reply
id=0x1b86, seq=1541/1286, ttl=54 (request in 1)
  3  1.001906657    192.168.0.6  →      104.28.6.89    ICMP   98    Echo (ping) request
st id=0x1b86, seq=1542/1542, ttl=64
  4  1.192770973    104.28.6.89  →      192.168.0.6    ICMP   98    Echo (ping) reply
id=0x1b86, seq=1542/1542, ttl=54 (request in 3)
  5  2.003365632    192.168.0.6  →      104.28.6.89    ICMP   98    Echo (ping) request
st id=0x1b86, seq=1543/1798, ttl=64
  6  2.434560259    104.28.6.89  →      192.168.0.6    ICMP   98    Echo (ping) reply
id=0x1b86, seq=1543/1798, ttl=54 (request in 5)
  7  3.003769942    192.168.0.6  →      104.28.6.89    ICMP   98    Echo (ping) request
st id=0x1b86, seq=1544/2054, ttl=64
  8  3.347729784    104.28.6.89  →      192.168.0.6    ICMP   98    Echo (ping) reply
id=0x1b86, seq=1544/2054, ttl=54 (request in 7)
  9  4.003967430    192.168.0.6  →      104.28.6.89    ICMP   98    Echo (ping) request
st id=0x1b86, seq=1545/2310, ttl=64
 10  4.163455725   104.28.6.89  →      192.168.0.6    ICMP   98    Echo (ping) reply
id=0x1b86, seq=1545/2310, ttl=54 (request in 9)
```

## Difference between decoded 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 to 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:

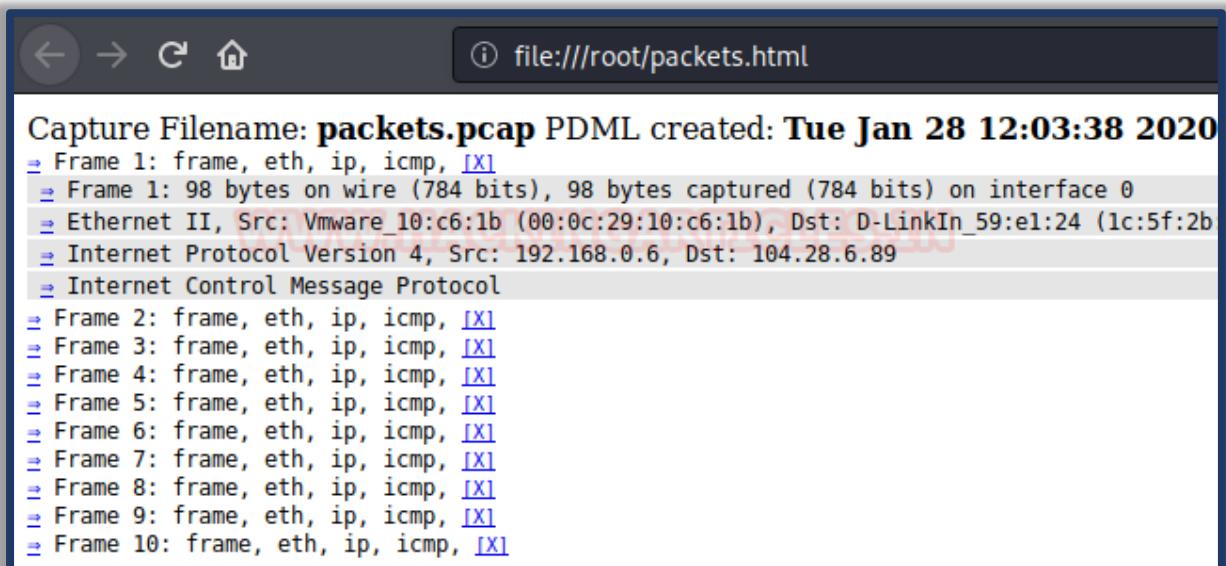
```
tshark -r packets.pcap -T pdml > packets.xml
```

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 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.

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

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



# 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 Len=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 Len=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 Len=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 HTTP/1.1
  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 HTTP/1.1
  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 HTTP/1.1
```

# 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.

```
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
=====
www.hackingarticles.in
```

# 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  
          ...
```

# Endpoint 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:

Sr.no.	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.

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

```

root@kali:~# tshark -r wlan.pcap -z endpoints,wlan -q | head
Running as user "root" and group "root". This could be dangerous.
=====
IEEE 802.11 Endpoints
Filter:<No Filter>
          | Packets | | Bytes | | Tx Packets | | Tx Bytes | | Rx Packets
AsustekC_c3:5e:01    18320   9311075    9843   8435055    8477
Tp-LinkT_16:87:18    8962    1644801    4024   1124143    4938
D-LinkIn_5f:81:6b    8122    950847     50    5484        8072
Motorola_31:a0:3b    8079    2137351    6262   1139916    1817
Tp-LinkT_09:7f:d3    7894    6218261    2930   453787    4964
Broadcast           -       6444    1728228     18    1164        6426

```

## 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.

```
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
```

```

root@kali:~# tshark -r wlan.pcap -z expert -q | head
Running as user "root" and group "root". This could be dangerous.

Errors (5)
=====
  Frequency Group          Protocol Summary
      5       Malformed      TCP    New fragment overlaps old data (retransmission?)

Warns (53821)
=====
  Frequency Group          Protocol Summary
  13373   Assumption     802.11 Radio  No plcp type information was available, assuming

```

## Packet Distribution Tree

---

In this option, we take the traffic from 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      0.0019      0.0019      100%
HTTP Request Packets  126           0.0019      0.0019      0.0019      50.81%
    GET                  126           0.0019      0.0019      0.0019      100.00%
HTTP Response Packets 122           0.0019      0.0019      0.0019      49.19%
    3xx: Redirection     105           0.0016      0.0016      0.0016      86.07%
        304 Not Modified   101           0.0015      0.0015      0.0015      96.19%
        302 Found          3            0.0000      0.0000      0.0000      2.86%
        301 Moved Permanently 1            0.0000      0.0000      0.0000      0.95%
    2xx: Success          17           0.0003      0.0002      0.0002      13.93%
        200 OK              14           0.0002      0.0002      0.0002      82.35%
        204 No Content       3            0.0000      0.0000      0.0000      17.65%
    ????: broken          0            0.0000      0.0000      0.0000      0.00%
    5xx: Server Error      0            0.0000      0.0000      0.0000      0.00%
    4xx: Client Error      0            0.0000      0.0000      0.0000      0.00%
    1xx: Informational     0            0.0000      0.0000      0.0000      0.00%
Other HTTP Packets      0            0.0000      0.0000      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

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

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%

# Colour Based Output Analysis

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.

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

```
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 0xb04 A det
3 1.309108098 192.168.0.137 → 8.8.8.8 DNS 84 Standard query 0xd70e AAAA
4 1.314734876 8.8.8.8 → 192.168.0.137 DNS 248 Standard query response 0x
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 0x
.dsccd.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 0x2172 A www
14 1.714151234 192.168.0.137 → 8.8.8.8 DNS 72 Standard query 0x6d77 AAAA
15 1.715114796 192.168.0.137 → 8.8.8.8 DNS 73 Standard query 0x3b2e A kal
16 1.715179313 192.168.0.137 → 8.8.8.8 DNS 73 Standard query 0xaf30 AAAA
17 1.715291271 192.168.0.137 → 8.8.8.8 DNS 74 Standard query 0x99d0 A too
18 1.715336702 192.168.0.137 → 8.8.8.8 DNS 74 Standard query 0xa3d2 AAAA
19 1.726762319 8.8.8.8 → 192.168.0.137 DNS 132 Standard query response 0x
20 1.730538887 8.8.8.8 → 192.168.0.137 DNS 133 Standard query response 0x
21 1.780500105 192.168.0.137 → 8.8.8.8 DNS 84 Standard query 0x97f4 A sni
22 1.780608110 192.168.0.137 → 8.8.8.8 DNS 84 Standard query 0x39f9 AAAA
23 1.786609781 8.8.8.8 → 192.168.0.137 DNS 191 Standard query response 0x
24 1.786635886 8.8.8.8 → 192.168.0.137 DNS 211 Standard query response 0x
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=1
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 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 date 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.

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

```
root@kali:~# cd packet/  
root@kali:~/packet# tshark -i eth0 -w packetsbuffer.pcap -b filesize:1 -b files:3  
Running as user "root" and group "root". This could be dangerous.  
Capturing on 'eth0'  
353 ^C  
  
File Actions Edit View Help  
  
root@kali:~# cd packet/  
root@kali:~/packet# ls  
packetsbuffer_00009_20200203122531.pcap  packetsbuffer_00010_20200203122531.pcap  
root@kali:~/packet# ls -la  
total 20  
lrwxr-xr-x  2 root root 4096 Feb  3 12:25 .  
lrwxr-xr-x 19 root root 4096 Feb  3 12:20 ..  
rw-----  1 root root 1028 Feb  3 12:25 packetsbuffer_00043_20200203122549.pcap  
rw-----  1 root root 1084 Feb  3 12:25 packetsbuffer_00044_20200203122549.pcap  
rw-----  1 root root  252 Feb  3 12:25 packetsbuffer_00045_20200203122549.pcap  
root@kali:~/packet# ls -la  
total 20  
lrwxr-xr-x  2 root root 4096 Feb  3 12:25 .  
lrwxr-xr-x 19 root root 4096 Feb  3 12:20 ..  
rw-----  1 root root 1228 Feb  3 12:25 packetsbuffer_00051_20200203122552.pcap  
rw-----  1 root root 1052 Feb  3 12:25 packetsbuffer_00052_20200203122553.pcap  
rw-----  1 root root  552 Feb  3 12:25 packetsbuffer_00053_20200203122554.pcap
```

## Autostop

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.

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

### 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.

```
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 dangerous.
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.

```
tshark -L
```

```
root@kali:~/packet# tshark -L
Running as user "root" and group "root". This could be dangerous.
Data link types of interface eth0 (use option -y to set)
  EN10MB (Ethernet)
  DOCSIS (DOCSIS)
```

## 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.

```
tshark -v
```

```

root@kali:~# tshark -v ↵
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)

Copyright 1998-2019 Gerald Combs <gerald@wireshark.org> and contributors.
License GPLv2+: GNU GPL version 2 or later <http://www.gnu.org/licenses/old-li
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiled (64-bit) with libpcap, with POSIX capabilities (Linux), with libnl 3,
with GLib 2.60.6, with zlib 1.2.11, with SMI 0.4.8, with c-ares 1.15.0, with L
5.2.4, with GnuTLS 3.6.9 and PKCS #11 support, with Gcrypt 1.8.5, with MIT
Kerberos, with MaxMind DB resolver, with nghttp2 1.39.2, with LZ4, with Snappy
with libxml2 2.9.4.

Running on Linux 5.3.0-kali2-amd64, with Intel(R) Core(TM) i7-9750H CPU @
2.60GHz (with SSE4.2), with 3934 MB of physical memory, with locale en_US.utf8
with libpcap version 1.9.1 (with TPACKET_V3), with GnuTLS 3.6.10, with Gcrypt
1.8.5, with zlib 1.2.11, binary plugins supported (0 loaded).

Built using gcc 9.2.1 20190909.

```

## 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

```
root@kali:~# tshark -G column-formats
Running as user "root" and group "root". This could
%q      802.1Q VLAN id
%Yt    Absolute date, as YYYY-MM-DD, and time
%YDOYt  Absolute date, as YYYY/DOY, and time
%At    Absolute time
%V      Cisco VSAN
%B      Cumulative Bytes
%Cus   Custom
%y      DCE/RPC call (cn_call_id / dg_seqnum)
%Tt    Delta time
%Gt    Delta time displayed
%rd    Dest addr (resolved)
%ud    Dest addr (unresolved)
%rD    Dest port (resolved)
%uD    Dest port (unresolved)
%d     Destination address
%D     Destination port
%a     Expert Info Severity
%I     FW-1 monitor if/direction
%F     Frequency/Channel
%hd   Hardware dest addr
%hs   Hardware src addr
%rhd  Hw dest addr (resolved)
%uhd  Hw dest addr (unresolved)
%rhs  Hw src addr (resolved)
%uhs  Hw src addr (unresolved)
%e    IEEE 802.11 RSSI
%x    IEEE 802.11 TX rate
%f    IP DSCP Value
%i    Information
%rnd  Net dest addr (resolved)
%und  Net dest addr (unresolved)
%rns  Net src addr (resolved)
%uns  Net src addr (unresolved)
%nd   Network dest addr
%ns   Network src addr
%m    Number
%L    Packet length (bytes)
%p    Protocol
%P+  Relative time
```

## 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 “slap.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.

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.

tshark -G dissector-tables

```
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 Teleservice
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 As not supported
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 supported
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 supported
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 supported
bcp.option      PPP BACP Options FT_UINT8        BASE_DEC      PPP BACP      Decode As not supported
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 supported
bctp.tpi        BCTP Tunneled Protocol Indicator FT_UINT32     BASE_DEC      BCTP      Decode As not supported
```

## 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.

```
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.

```
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.

**tshark -G fields | head**

```
root@kali:~# tshark -G fields | head ↵
Running as user "root" and group "root". This could be dangerous.
P     Short Frame      _ws.short
P     Malformed Packet    _ws.malformed
P     Unreassembled Fragmented Packet _ws.unreassembled
F     Dissector bug      _ws.malformed.dissector_bug      FT_NONE _ws.malformed
F     Reassembly error    _ws.malformed.reassembly      FT_NONE _ws.malformed
F     Malformed Packet (Exception occurred) _ws.malformed.expert      FT_NONE _ws.ma
P     Type Length Mismatch _ws.type_length
F     Trying to fetch X with length Y _ws.type_length.mismatch      FT_NONE _ws.ty
P     Number-String Decoding Error _ws.number_string.decoding_error
F     Failed to decode number from string      _ws.number_string.decoding_error.failed
x0
```

## 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.

**tshark -G ftypes**

```
root@kali:~# tshark -G ftypes
Running as user "root" and group "root". This could be dangerous
FT_NONE Label
FT_PROTOCOL Protocol
FT_BOOLEAN Boolean
FT_CHAR Character, 1 byte
FT_UINT8 Unsigned integer, 1 byte
FT_UINT16 Unsigned integer, 2 bytes
FT_UINT24 Unsigned integer, 3 bytes
FT_UINT32 Unsigned integer, 4 bytes
FT_UINT40 Unsigned integer, 5 bytes
FT_UINT48 Unsigned integer, 6 bytes
FT_UINT56 Unsigned integer, 7 bytes
FT_UINT64 Unsigned integer, 8 bytes
FT_INT8 Signed integer, 1 byte
FT_INT16 Signed integer, 2 bytes
FT_INT24 Signed integer, 3 bytes
FT_INT32 Signed integer, 4 bytes
FT_INT40 Signed integer, 5 bytes
FT_INT48 Signed integer, 6 bytes
FT_INT56 Signed integer, 7 bytes
FT_INT64 Signed integer, 8 bytes
FT_IEEE_11073_SFLOAT IEEE-11073 Floating point (16-bit)
FT_IEEE_11073_FLOAT IEEE-11073 Floating point (32-bit)
FT_FLOAT Floating point (single-precision)
FT_DOUBLE Floating point (double-precision)
FT_ABSOLUTE_TIME Date and time
FT_RELATIVE_TIME Time offset
FT_STRING Character string
FT_STRINGZ Character string
FT_UINT_STRING Character string
FT_ETHER Ethernet or other MAC address
FT_BYTES Sequence of bytes
FT_UINT_BYTES Sequence of bytes
FT_IPv4 IPv4 address
FT_IPv6 IPv6 address
FT_IPXNET IPX network number
FT_FRAMENUM Frame number
FT_PCRE Compiled Perl-Compatible Regular Expression (GRegex) obj
FT_GUID Globally Unique Identifier
FT_OID ASN.1 object identifier
```

## 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 talks about the status of the heuristic. It will be T in case it is heuristics and F otherwise.

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

```
root@kali:~# tshark -G heuristic-decodes
Running as user "root" and group "root".
rtsp      rtp      F
sctp      sip      T
sctp      nbap     T
sctp      jxta     T
udp       xml      F
udp       wol      T
udp       wg       T
udp       waveagent T
udp       wassp     F
udp       udt      T
udp       teredo   F
udp       stun     T
udp       srt      T
udp       sprt     T
udp       skype    F
udp       sip      T
udp       rtps     T
udp       rtp      F
udp       rtcp     T
udp       rpcap    T
udp       nnc      T
```

## 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**

```
root@kali:~# tshark -G plugins ↵
Running as user "root" and group "root". This could be dangerous.
ethercat.so        0.1.0  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
gryphon.so         0.0.4  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
irda.so            0.0.6  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
mate.so             1.0.1  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
opcua.so           1.0.0  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
profinet.so        0.2.4  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
stats_tree.so      0.0.1  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
transum.so          2.0.4  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
unistim.so          0.0.2  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
usbdump.so          0.0.1  file type   /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
wimax.so            1.2.0  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
wimaxasncp.so      0.0.1  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
wimaxmacphy.so     0.0.1  dissector    /usr/lib/x86_64-linux-gnu/wireshark/plugins/3
```

## 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.

```
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.

```
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
```

```
root@kali:~# tshark -G folders ↵
Running as user "root" and group "root". This could be dangerous.
Temp:          /tmp
Personal configuration: /root/.config/wireshark
Global configuration:  /usr/share/wireshark
System:         /etc
Program:        /usr/bin
Personal Plugins: /root/.local/lib/wireshark/plugins/3.0
Global Plugins:   /usr/lib/x86_64-linux-gnu/wireshark/plugins/3.0
Personal Lua Plugins: /root/.local/lib/wireshark/plugins
Global Lua Plugins:  /usr/lib/x86_64-linux-gnu/wireshark/plugins
Extcap path:    /usr/lib/x86_64-linux-gnu/wireshark/extcap
MaxMind database path: /usr/share/GeoIP
MaxMind database path: /var/lib/GeoIP
MaxMind database path: /usr/share/GeoIP
MaxMind database path: /var/lib/GeoIP
```

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 have Python 3 installed on our machine. We installed Python3 as shown in the image given below.

```
apt install python3
```

```
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 libcrypt1-dev libpython3.7-dev libpython3.7-minimal libpython3.7-stdlib python-pip-whl python3-dev python3-environ python3-secretstorage python3-setuptools python3-wheel python3-xcb-dev
Suggested packages:
  glibc-doc libkf5wallet-bin gir1.2-gnomekeyring-1.0 python3-gnomekeyring
The following NEW packages will be installed:
  libcrypt-dev libcrypt1 libpython3-dev libpython3.7-dev python3-keyrings.alt python3-pip python3-secretstorage python3-xcb-dev
The following packages will be upgraded:
  libc-dev-bin libc6 libc6-dev libc6-i386 libpython3.7 libpython3.7-minimal
10 upgraded, 15 newly installed, 0 to remove and 663 not upgraded.
Need to get 502 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.

```
pip3 install pyshark
```

```
root@kali:~# pip3 install pyshark ↵
Collecting pyshark
  Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection.VerifiedHTTPSConnection object at 0x7fd92b76d610>: Failed to establish a new connection'': /packages/b9/b0/ef87c71f7937ea8124944b2081210f9df10e47d2faa57d7c30d3e12af064/pyshark-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
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory-assigned)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: 00:0c:29:d5:b7:2d
    Type: IPv4 (0x0800)
    Address: 00:0c:29:d5:b7:2d
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory-assigned)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
Layer IP:
    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-Capable Transport
    Total Length: 52
    Identification: 0x4b7c (19324)
    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: 64
    Protocol: TCP (6)
    Header checksum: 0x62cb [validation disabled]
    Header checksum status: Unverified
    Source: 192.168.0.137
    Destination: 13.35.190.40
Layer TCP:
    Source Port: 38820
    Destination Port: 443
    Stream index: 1
    TCP Segment Len: 0
    Sequence number: 1 (relative sequence number)
    Next sequence number: 1 (relative sequence number)
    Acknowledgment number: 1 (relative ack number)
    1000 .... = Header Length: 32 bytes (8)
    Flags: 0x010 (ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... =Nonce: Not set
    .... 0... .... = Congestion Window Reduced (CWR): Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
```

## 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.

```
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'
```

## Layers, Src and Dst Fields

---

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

```

>>> dir(pkt.ip) ↵
['DATA_LAYER', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
 '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__',
 '__reduce_ex__', '__repr__', '__setattr__', '__setstate__', '__sizeof__', '__str__',
 '__field_prefix__', '_get_all_field_lines', '_get_all_fields_with_alternates',
 '_get_sanitize_field_name', 'addr', 'checksum', 'checksum_status', 'dsfield',
 'dsfield_dlags', '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

```

```

root@kali:~# ifconfig eth0 promisc ↵
root@kali:~# ifconfig eth0 ↵
eth0: flags=4419<UP,BROADCAST,RUNNING,PROMISC,MULTICAST> mtu 1500
        inet 192.168.0.137 netmask 255.255.255.0 broadcast 192.168.0.255
        inet6 fe80::20c:29ff:fed5:b72d prefixlen 64 scopeid 0x20<link>
          ether 00:0c:29:d5:b7:2d txqueuelen 1000 (Ethernet)
            RX packets 67816 bytes 85545596 (81.5 MiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 30726 bytes 2463013 (2.3 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~# tshark -i eth0 -c 10 ↵
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
  1 0.000000000 192.168.0.137 → 35.169.2.62 TLSv1.2 164 Application Data
  2 0.000142943 192.168.0.137 → 107.23.176.98 TLSv1.2 164 Application Data
  3 0.236904732 35.169.2.62 → 192.168.0.137 TLSv1.2 187 Application Data
  4 0.236921665 192.168.0.137 → 35.169.2.62 TCP 66 40520 → 443 [ACK] Seq=99 Ack=122 Win=1
  5 0.242952531 107.23.176.98 → 192.168.0.137 TLSv1.2 187 Application Data
  6 0.242967301 192.168.0.137 → 107.23.176.98 TCP 66 41152 → 443 [ACK] Seq=99 Ack=122 Win=1
  7 1.343354460 192.168.0.6 → 224.0.0.251 IGMPv2 60 Membership Report group 224.0.0.1
  8 2.842606464 192.168.0.6 → 224.0.0.252 IGMPv2 60 Membership Report group 224.0.0.1
  9 6.807673972 192.168.0.137 → 34.213.241.62 TCP 66 51094 → 443 [ACK] Seq=1 Ack=1 Win=1
 10 7.100843807 34.213.241.62 → 192.168.0.137 TCP 66 [TCP ACKed unseen segment] 443 → 164
10 packets captured
root@kali:~# tshark -i eth0 -c 10 -p ↵
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
  1 0.000000000 34.213.241.62 → 192.168.0.137 TLSv1.2 97 Encrypted Alert
  2 0.000019158 192.168.0.137 → 34.213.241.62 TCP 66 51094 → 443 [ACK] Seq=1 Ack=32 Win=1
  3 0.000222027 192.168.0.137 → 34.213.241.62 TLSv1.2 97 Encrypted Alert
  4 0.000288786 192.168.0.137 → 34.213.241.62 TCP 66 51094 → 443 [FIN, ACK] Seq=32 Ack=33 Win=1
  5 0.289883135 34.213.241.62 → 192.168.0.137 TCP 66 [TCP Previous segment not captured]
  6 0.289903932 34.213.241.62 → 192.168.0.137 TCP 66 [TCP Out-Of-Order] 443 → 51094 [FIN, ACK]
  7 0.289914338 192.168.0.137 → 34.213.241.62 TCP 66 51094 → 443 [ACK] Seq=33 Ack=33 Win=1
  8 4.120921966 192.168.0.137 → 35.169.2.62 TLSv1.2 165 Application Data
  9 4.121065015 192.168.0.137 → 107.23.176.98 TLSv1.2 164 Application Data
 10 4.394954971 35.169.2.62 → 192.168.0.137 TLSv1.2 188 Application Data
10 packets captured

```

## References

- <https://www.hackingarticles.in/beginners-guide-to-tshark-part-1/>
- <https://www.hackingarticles.in/beginners-guide-to-tshark-part-2/>
- <https://www.hackingarticles.in/beginners-guide-to-tshark-part-3/>

# JOIN OUR TRAINING PROGRAMS

**CLICK HERE**

## BEGINNER

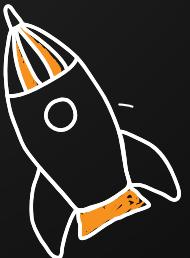
Ethical Hacking

Bug Bounty

Network Security Essentials

Network Pentest

Wireless Pentest



## ADVANCED

Burp Suite Pro

Web Services-API

Pro Infrastructure VAPT

Computer Forensics

Android Pentest

Advanced Metasploit

CTF



## EXPERT

Red Team Operation

Privilege Escalation

- APT's - MITRE Attack Tactics
- Active Directory Attack
- MSSQL Security Assessment

Windows

Linux

