

# Comprehensive Guide to tcpdump (Part 3)

March 24, 2020 By Raj Chandel

This is the third article in the Comprehensive Guide to tcpdump Series. Please find the first and second articles of the series below.

- Comprehensive Guide to tcpdump (Part 1).
- Comprehensive Guide to tcpdump (Part 2).

In this part, we will cover some of the advance features which we were unable to cover in the previous parts of the series. So we can get more benefits from this tool.

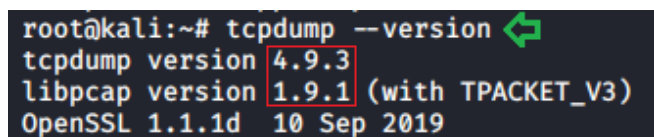
## Table of Content

- Version Information
- Quick mode
- Verbose mode
- HTTP Requests
- User Agent
- Port Range
- Destination
- Source
- Network
- TCP Packets
- Tcpdump to Wireshark

## Version Information

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

```
tcpdump --version
```



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

## Quick Mode

Arguably if the network is very quite, performing any operation during that time will take more time than usual. The person who developed tcpdump thought of this conundrum and gave us the way to speed up the process by

using the “-q” parameter. It will print less information about protocols and data packets to save time.

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

```
root@kali:~# tcpdump -i eth0 -c 5 ↩
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:39:00.490605 ARP, Request who-has 192.168.0.6 tell 192.168.0.5, length 46
14:39:00.491548 IP 192.168.0.107.55213 > dns.google.domain: 12741+ PTR? 6.0.168.192.in-
14:39:00.495547 IP dns.google.domain > 192.168.0.107.55213: 12741 NXDomain* 0/1/0 (112)
14:39:00.495668 IP 192.168.0.107.59775 > dns.google.domain: 13430+ PTR? 5.0.168.192.in-
14:39:00.499695 IP dns.google.domain > 192.168.0.107.59775: 13430 NXDomain* 0/1/0 (112)
5 packets captured
9 packets received by filter
0 packets dropped by kernel
root@kali:~# tcpdump -i eth0 -c 5 -q ↩
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:39:04.865940 IP 192.168.0.107 > 104.28.6.89: ICMP echo request, id 1320, seq 4228, l
14:39:04.866477 IP 192.168.0.107.40767 > dns.google.domain: UDP, length 42
14:39:04.870787 ARP, Request who-has 192.168.0.6 tell 192.168.0.5, length 46
14:39:04.870798 IP 192.168.0.5.5050 > 239.255.255.250.5050: UDP, length 43
14:39:04.870801 IP dns.google.domain > 192.168.0.107.40767: UDP, length 137
5 packets captured
16 packets received by filter
0 packets dropped by kernel
```

## Verbose Mode

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

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

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

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

```

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

```

## HTTP Requests

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

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

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

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

```

root@kali:~# tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:30:01.926242 IP (tos 0x0, ttl 64, id 42167, offset 0, flags [DF], proto TCP (6),
    192.168.0.107.48234 > rs202995.rs.hosteurope.de.http: Flags [P.], cksum 0xa554
    gth 345: HTTP, length: 345
        GET / HTTP/1.1
        Host: testphp.vulnweb.com
        User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox
        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Connection: keep-alive
        Upgrade-Insecure-Requests: 1
        Cache-Control: max-age=0

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

```

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

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

```

root@kali:~# tcpdump -s 0 -A -vv 'tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x504f5354'
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:32:35.944089 IP (tos 0x0, ttl 64, id 59541, offset 0, flags [DF], proto TCP (6),
    192.168.0.107.33312 > 117.18.237.29.http: Flags [P.], cksum 0x24dd (incorrect
    TTP, length: 371
        POST / HTTP/1.1
        Host: ocsf.digicert.com
        User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox
        Accept: */*
        Accept-Language: en-US,en;q=0.5
        Accept-Encoding: gzip, deflate
        Content-Type: application/ocsp-request
        Content-Length: 83
        Connection: keep-alive

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

```

- **Request-URL-** It is a uniform resource identifier, which identifies the resource on which we need to apply requests. The most common form of this is used to identify a resource on a server. If a client wants to retrieve the data directly from the server, where it originated, then it would create a connection to port 80 of the host and send the request. These requests can be captured using the following commands:

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

```
root@kali:~# tcpdump -v -n -l | egrep -i "POST /|GET /|Host:" ↵
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
GET /artists.php HTTP/1.1
Host: testphp.vulnweb.com
GET /style.css HTTP/1.1
Host: testphp.vulnweb.com
GET /images/logo.gif HTTP/1.1
Host: testphp.vulnweb.com
GET /index.php HTTP/1.1
Host: testphp.vulnweb.com
^C80 packets captured
85 packets received by filter
0 packets dropped by kernel
```

## User Agent

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

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

```
root@kali:~# tcpdump -nn -A -s150 -l | grep "User-Agent:" ↵
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 150 bytes
User-Agent: Mozilla/5.0 (X11;
User-Agent: Mozilla/5.0 (X11; Li
User-Agent: Mozilla/5.0 (X
^C71 packets captured
72 packets received by filter
```

## Port Range

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

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



```

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

```

## Destination

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

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

```

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

```

## Source

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

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

```

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

```

## Network

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

```
tcpdump net 192.168.0.1 -c5
```

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

## TCP Packets

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

- **SYN**- SYN flag is known as Synchronizes sequence numbers to initiate a TCP connection. We can capture this particular packet from traffic with the help of tcpdump.

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

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

- **RST**- RST flag is known as reset flag. This flag is sent from the receiver to the sender if a packet is sent to a particular host that was expecting it. RST flag is used to re-establish a TCP end-to-end connection. We can capture this flag from our data traffic with the help of tcpdump.

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



```

root@kali:~# tcpdump 'tcp[tcpflags] = tcp-rst' ↩
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
04:15:58.610080 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.610178 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917610 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917684 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917698 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:15:58.917711 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:16:03.730516 IP 192.168.0.107.57670 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:03.730792 IP 192.168.0.107.57670 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:05.983486 IP 192.168.0.107.57674 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:05.983838 IP 192.168.0.107.57674 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:07.853537 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:16:07.853847 IP ec2-54-208-84-166.compute-1.amazonaws.com.https > 192.168.0.107.
04:16:08.021311 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:08.021497 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:08.022655 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:08.023500 IP 192.168.0.107.57676 > rs202995.rs.hosteurope.de.http: Flags [R],
04:16:08.052121 IP 192.168.0.107.57678 > rs202995.rs.hosteurope.de.http: Flags [R],

```

- **ACK-** ACK flag is known as the Acknowledgement flag. This flag is used to acknowledge that our data packet has been successfully received. We can capture these flags with tcpdump to study our data traffic.

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

```

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

```

## Tcpdump to Wireshark

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

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

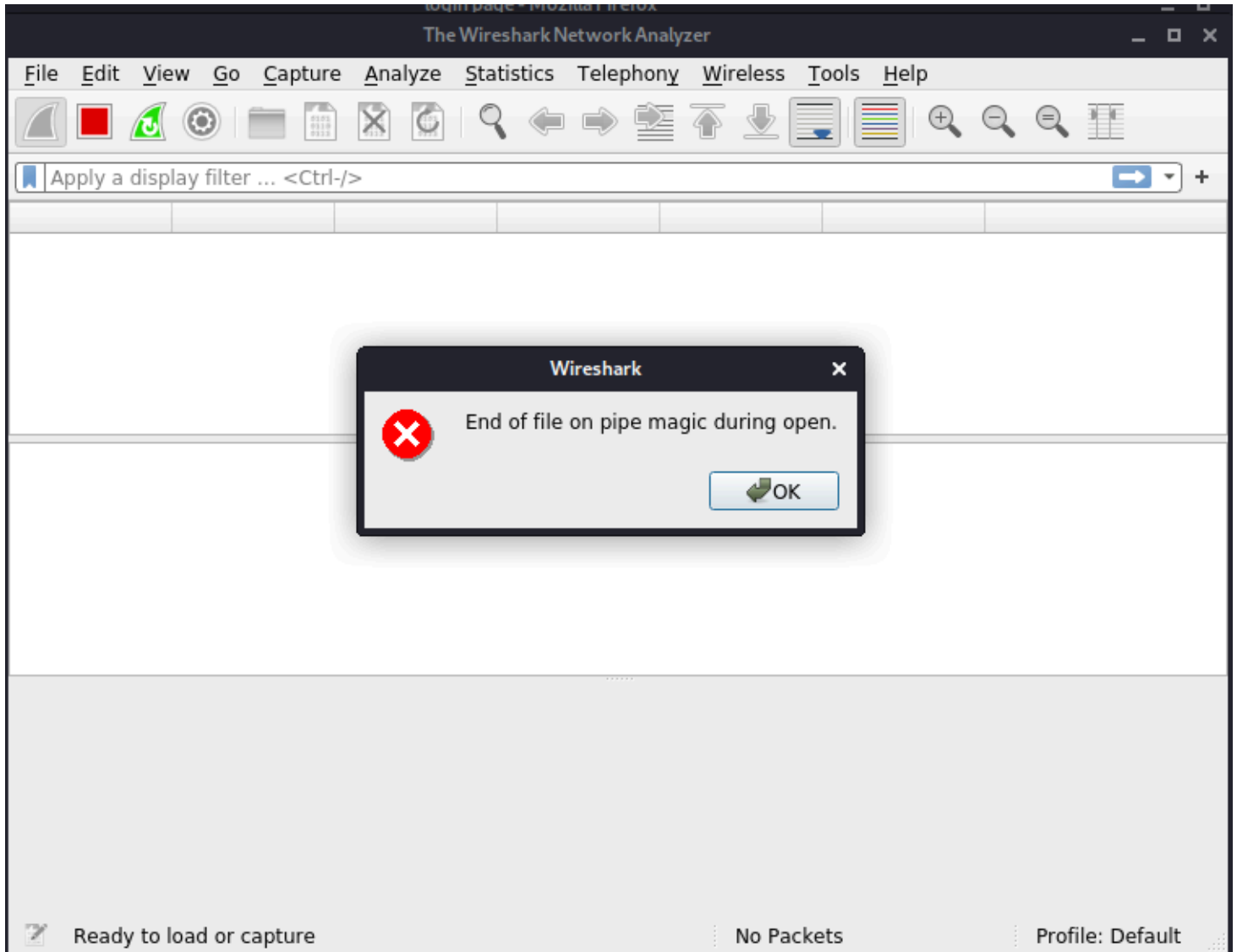
```

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

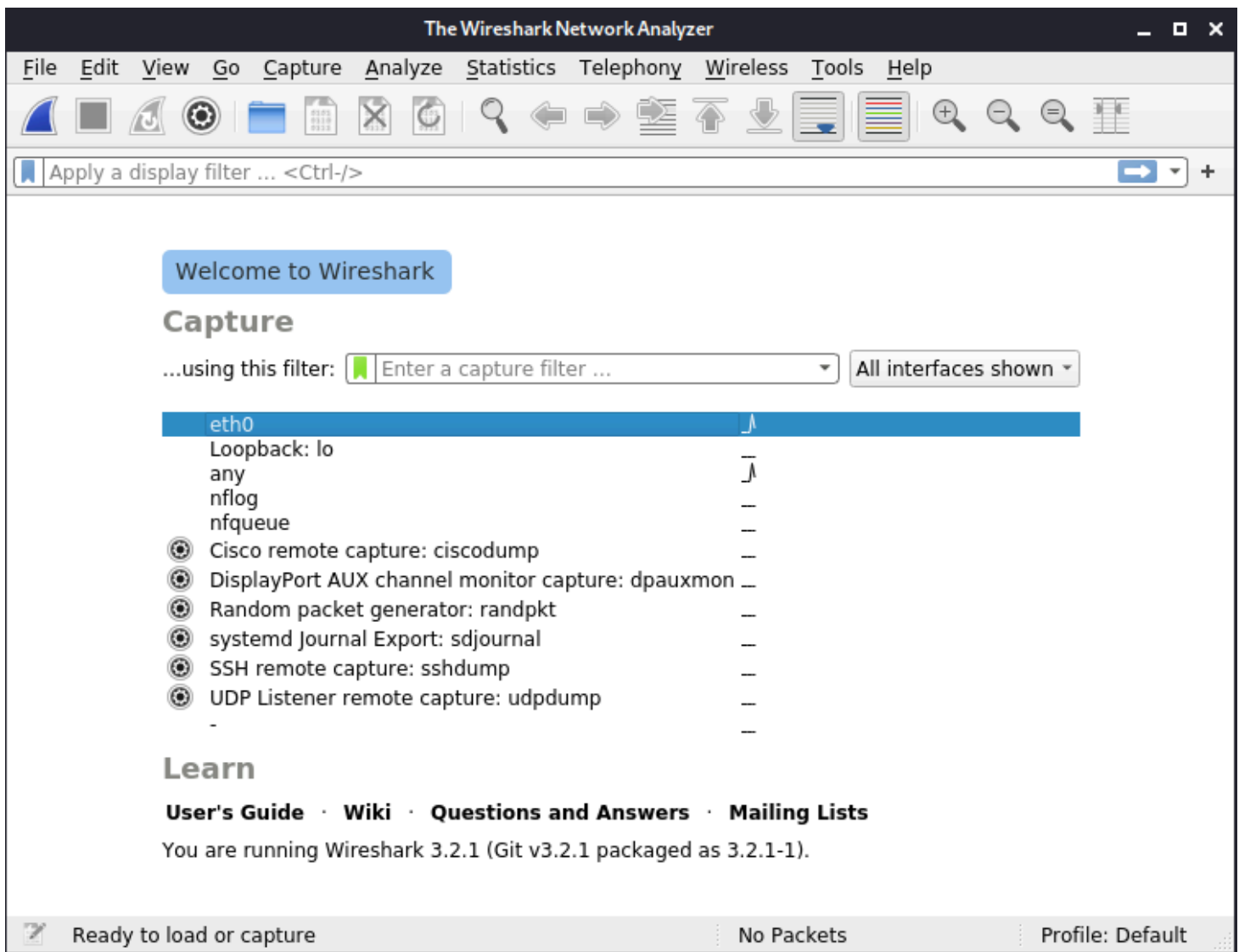
```



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



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



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

Capturing from eth0						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.0.107	8.8.8.8	DNS	79	Standard query 0x408c A
2	0.005912502	8.8.8.8	192.168.0.107	DNS	95	Standard query response
3	0.006481865	192.168.0.107	176.28.50.165	TCP	74	36682 → 80 [SYN] Seq=0
4	0.028459282	192.168.0.107	176.28.50.165	TCP	74	36684 → 80 [SYN] Seq=0
5	0.197669888	176.28.50.165	192.168.0.107	TCP	74	80 → 36682 [SYN, ACK] S
6	0.197727671	192.168.0.107	176.28.50.165	TCP	66	36682 → 80 [ACK] Seq=1
7	0.197931113	176.28.50.165	192.168.0.107	TCP	74	80 → 36684 [SYN, ACK] S
8	0.197954082	192.168.0.107	176.28.50.165	TCP	66	36684 → 80 [ACK] Seq=1
9	1.244022339	192.168.0.107	8.8.8.8	DNS	76	Standard query 0x2553 A
10	1.244341348	192.168.0.107	8.8.8.8	DNS	76	Standard query 0xa762 A
11	1.250577810	8.8.8.8	192.168.0.107	DNS	161	Standard query response
12	1.426451311	8.8.8.8	192.168.0.107	DNS	92	Standard query response
13	1.427402369	192.168.0.107	54.208.84.166	TCP	74	47988 → 443 [SYN] Seq=6
14	1.733945850	54.208.84.166	192.168.0.107	TCP	74	443 → 47988 [SYN, ACK]
15	1.734041013	192.168.0.107	54.208.84.166	TCP	66	47988 → 443 [ACK] Seq=1
16	1.736890085	192.168.0.107	54.208.84.166	TLSv1	583	Client Hello
17	2.042348627	54.208.84.166	192.168.0.107	TCP	66	443 → 47988 [ACK] Seq=1
18	2.042389246	54.208.84.166	192.168.0.107	TLSv1.2	1414	Server Hello
19	2.042407085	192.168.0.107	54.208.84.166	TCP	66	47988 → 443 [ACK] Seq=5
20	2.043217095	54.208.84.166	192.168.0.107	TCP	1414	443 → 47988 [ACK] Seq=1

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