Network Capture and Troubleshooting with tcpdump

Source: https://hackertarget.com/tcpdump-examples/

Display ASCII text

Adding -A to the command line will have the output include the ascii strings from the capture. This allows easy reading and the ability to parse the output using grep or other commands. Another option that shows both hexadecimal output and ASCII is the -X option.

```
:~$ sudo tcpdump -A -s0 port 80
```

Capture on Protocol

Filter on UDP traffic. Another way to specify this is to use **protocol 17** that is udp. These two commands will produce the same result. The equivalent of the tcp filter is **protocol 6**.

```
:~$ sudo tcpdump -i eth0 udp
:~$ sudo tcpdump -i eth0 proto 17
```

Capture Hosts based on IP address

Using the host filter will capture traffic going to (destination) and from (source) the IP address.

```
:~$ sudo tcpdump -i eth0 host 10.10.1.1
```

Alternatively capture only packets going one way using src or dst.

```
:~$ sudo tcpdump -i eth0 dst 10.10.1.20
```

Write a capture file

Writing a standard pcap file is a common command option. Writing a capture file to disk allows the file to be opened in Wireshark or other packet analysis tools.

```
:~$ sudo tcpdump -i eth0 -s0 -w test.pcap
```

Line Buffered Mode

Without the option to force line (-l) buffered (or packet buffered -C) mode you will not always get the **expected response when piping** the tcpdump output to another command such as grep. By using this option the output is sent immediately to the piped command giving an immediate response when troubleshooting.

```
:~$ sudo tcpdump -i eth0 -s0 -l port 80 | grep 'Server:'
```

Combine Filters

Throughout these examples you can use standard logic to combine different filters.

```
and or &&
or or ||
not or !
```

Practical Examples

In many of these examples there are a number of ways that the result could be achieved. As seen in some of the examples it is possible to focus the capture right down to individual bits in the packet.

The method you will use will depend on your desired output and how much traffic is on the wire. Capturing on a busy gigabit link may force you to use specific low level packet filters.

When troubleshooting you often simply want to get a result. Filtering on the port and selecting **ascii** output in combination with grep, cut or awk will often get that result. You can always go deeper into the packet if required.

For example when capturing HTTP requests and responses you could filter out all packets except the data by removing **SYN /ACK / FIN** however if you are using grep the noise will be filtered anyway. Keep it simple.

This can be seen in the following examples, where the aim is to get a result in the simplest (and

therefore fastest) manner.

1. Extract HTTP User Agents

Extract HTTP User Agent from HTTP request header.

```
:~$ sudo tcpdump -nn -A -s1500 -l | grep "User-Agent:"
```

By using egrep and multiple matches we can get the User Agent and the Host (or any other header) from the request.

```
:~$ sudo tcpdump -nn -A -s1500 -l | egrep -i 'User-Agent:|Host:'
```

2. Capture only HTTP GET and POST packets

Going deep on the filter we can specify only packets that match GET.

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

Alternatively we can select only on POST requests. Note that the POST data may not be included in the packet captured with this filter. It is likely that a POST request will be split across multiple TCP data packets.

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

The hexadecimal being matched in these expressions matches the ascii for GET and POST.

As an explanation **tcp[((tcp[12:1] & 0xf0) >> 2):4]** first determines the location of the bytes we

are interested in (after the TCP header) and then selects the 4 bytes we wish to match against.

3. Extract HTTP Request URL's

Parse **Host** and **HTTP Request location** from traffic. By not targeting **port 80** we may find these requests on any port such as HTTP services running on high ports.

```
:~$ sudo tcpdump -s 0 -v -n -l | egrep -i "POST /|GET /|Host:"

tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
 POST /wp-login.php HTTP/1.1
 Host: dev.example.com
 GET /wp-login.php HTTP/1.1
 Host: dev.example.com
 GET /favicon.ico HTTP/1.1
 Host: dev.example.com
 GET / HTTP/1.1
 Host: dev.example.com
```

4. Extract HTTP Passwords in POST Requests

Lets get some passwords from the POST data. Will include Host: and request location so we know what the password is used for.

```
:~$ sudo tcpdump -s 0 -A -n -l | egrep -i "POST /|pwd=|passwd=|password=|Host:"

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:25:54.799014 IP 10.10.1.30.39224 > 10.10.1.125.80: Flags [P.], seq 1458768667:145
8770008, ack 2440130792, win 704, options [nop,nop,TS val 461552632 ecr 208900561],
length 1341: HTTP: POST /wp-login.php HTTP/1.1
.....s..POST /wp-login.php HTTP/1.1
Host: dev.example.com
.....s..log=admin&pwd=notmypassword&wp-
submit=Log+In&redirect_to=http%3A%2F%2Fdev.example.com%2Fwp-admin%2F&testcookie=1
```

5. Capture Cookies from Server and from Client

MMMmmm Cookies! Capture cookies from the server by searching on Set-Cookie: (from Server) and Cookie: (from Client).

```
:~$ sudo tcpdump -nn -A -s0 -l | egrep -i 'Set-Cookie|Host:|Cookie:'

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlp58s0, link-type EN10MB (Ethernet), capture size 262144 bytes
Host: dev.example.com
Cookie:
 wordpress_86be02xxxxxxxxxxxxxxxxxxxc43=admin%7C152xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxfb3e15c744fdd6; _ga=GA1.2.21343434343421934; _gid=GA1.2.927343434349
426; wordpress_test_cookie=WP+Cookie+check; wordpress_logged_in_86be6546546456456
54645653fc43=admin%7C15275102testtesttesttestab7a61e; wp-settings-time-1=1527337439
```

6. Capture all ICMP packets

See all ICMP packets on the wire.

```
:~$ sudo tcpdump -n icmp

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:34:21.590380 IP 10.10.1.217 > 10.10.1.30: ICMP echo request, id 27948, seq 1, len
gth 64
11:34:21.590434 IP 10.10.1.30 > 10.10.1.217: ICMP echo reply, id 27948, seq 1, lengt
h 64
11:34:27.680307 IP 10.10.1.159 > 10.10.1.1: ICMP 10.10.1.189 udp port 59619 unreacha
ble, length 115
```

7. Show ICMP Packets that are not ECHO/REPLY (standard ping)

Filter on the icmp type to select on icmp packets that are not standard ping packets.

```
:~$ sudo tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:37:04.041037 IP 10.10.1.189 > 10.10.1.20: ICMP 10.10.1.189 udp port 36078 unreach
able, length 156
```

8. Capture SMTP / POP3 Email

It is possible to extract email body and other data, in this example we are only parsing the email recipients.

```
:~$ sudo tcpdump -nn -l port 25 | grep -i 'MAIL FROM\|RCPT TO'
```

9. Troubleshooting NTP Query and Response

In this example we see the NTP query and response.

```
:~$ sudo tcpdump dst port 123

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
21:02:19.112502 IP test33.ntp > 199.30.140.74.ntp: NTPv4, Client, length 48
21:02:19.113888 IP 216.239.35.0.ntp > test33.ntp: NTPv4, Server, length 48
21:02:20.150347 IP test33.ntp > 216.239.35.0.ntp: NTPv4, Client, length 48
21:02:20.150991 IP 216.239.35.0.ntp > test33.ntp: NTPv4, Server, length 48
```

10. Capture SNMP Query and Response

Using onesixtyone the fast SNMP protocol scanner we test an SNMP service on our local network and capture the GetRequest and GetResponse. For anyone who has had the (dis)pleasure of

troubleshooting SNMP, this is a great way to see exactly what is happening on the wire. You can see the OID clearly in the traffic, very helpful when wrestling with MIBS.

```
:~$ onesixtyone 10.10.1.10 public

Scanning 1 hosts, 1 communities
10.10.1.10 [public] Linux test33 4.15.0-20-generic #21-Ubuntu SMP Tue Apr 24 06:16:1
5 UTC 2018 x86_64
```

```
:~$ sudo tcpdump -n -s0  port 161 and udp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlp58s0, link-type EN10MB (Ethernet), capture size 262144 bytes
23:39:13.725522 IP 10.10.1.159.368
26 > 10.10.1.20.161:  GetRequest(28)  .1.3.6.1.2.1.1.1.0
23:39:13.728789 IP 10.10.1.20.161
 > 10.10.1.159.36826:  GetResponse(109)  .1.3.6.1.2.1.1.1.0="Linux testmachine 4.15.
0-20-generic #21-Ubuntu SMP Tue Apr 24 06:16:15 UTC 2018 x86_64"
```

11. Capture FTP Credentials and Commands

Capturing FTP commands and login details is straight forward. After the authentication is established an FTP session can be **active** or **passive** this will determine whether the data part of the session is conducted over TCP port 20 or another ephemeral port. With the following command you will USER and PASS in the output (which could be fed to grep) as well as the FTP commands such as LIST, CWD and PASSIVE.

```
:~$ sudo tcpdump -nn -v port ftp or ftp-data
```

12. Rotate Capture Files

When capturing large amounts of traffic or over a long period of time it can be helpful to automatically create new files of a fixed size. This is done using the parameters -W, -G and -C.

In this command the file **capture-(hour).pcap** will be created every (-G) **3600 seconds** (1 hour).

The files will be overwritten the following day. So you should end up with **capture-{1-24}.pcap**, if the hour was **15** the new file is (**/tmp/capture-15.pcap**).

```
:~$ tcpdump  -w /tmp/capture-%H.pcap -G 3600 -C 200
```

13. Capture IPv6 Traffic

Capture IPv6 traffic using the ip6 filter. In these examples we have specified the TCP and UDP protocols using proto 6 and proto 17.

```
tcpdump -nn ip6 proto 6
```

IPv6 with UDP and reading from a previously saved capture file.

```
tcpdump -nr ipv6-test.pcap ip6 proto 17
```

14. Detect Port Scan in Network Traffic

In the following example you can see the traffic coming from a single source to a single destination. The Flags [S] and [R] can be seen and matched against a seemingly random series of destination ports. These ports are seen in the RESET that is sent when the SYN finds a closed port on the destination system. This is standard behaviour for a port scan by a tool such as Nmap.

We have another tutorial on Nmap that details captured port scans (**open / closed / filtered**) in a number of Wireshark captures.

```
:~$ tcpdump -nn

21:46:19.693601 IP 10.10.1.10.6046
0 > 10.10.1.199.5432
: Flags [S], seq 116466344, win 29200, options [mss 1460,sackOK,TS val 3547090332 ec
r 0,nop,wscale 7], length 0
```

```
21:46:19.693626 IP 10.10.1.10.3547
0 > 10.10.1.199.513
: Flags [S], seq 3400074709, win 29200, options [mss 1460,sackOK,TS val 3547090332 e
cr 0,nop,wscale 7], length 0
21:46:19.693762 IP 10.10.1.10.4424
4 > 10.10.1.199.389
: Flags [S], seq 2214070267, win 29200, options [mss 1460,sackOK,TS val 3547090333 e
cr 0,nop,wscale 7], length 0
21:46:19.693772 IP 10.10.1.199.389 > 10.10.1.10.44244: Flags [R.], seq 0, ack 221407
0268, win 0, length 0
21:46:19.693783 IP 10.10.1.10.3517
2 > 10.10.1.199.1433
: Flags [S], seq 2358257571, win 29200, options [mss 1460,sackOK,TS val 3547090333 e
cr 0,nop,wscale 7], length 0
21:46:19.693826 IP 10.10.1.10.3302
2 > 10.10.1.199.49153
: Flags [S], seq 2406028551, win 29200, options [mss 1460,sackOK,TS val 3547090333 e
cr 0,nop,wscale 7], length 0
21:46:19.695567 IP 10.10.1.10.5513
0 > 10.10.1.199.49154
: Flags [S], seq 3230403372, win 29200, options [mss 1460,sackOK,TS val 3547090334 e
cr 0,nop,wscale 7], length 0
21:46:19.695590 IP 10.10.1.199.49154 > 10.10.1.10.55130: Flags [R.], seq 0, ack 3230
403373, win 0, length 0
21:46:19.695608 IP 10.10.1.10.3346
0 > 10.10.1.199.4915
2: Flags [S], seq 3289070068, win 29200, options [mss 1460,sackOK,TS val 3547090335
ecr 0,nop,wscale 7], length 0
21:46:19.695622 IP 10.10.1.199.49152 > 10.10.1.10.33460: Flags [R.], seq 0, ack 3289
070069, win 0, length 0
21:46:19.695637 IP 10.10.1.10.3494
0 > 10.10.1.199.1029
: Flags [S], seq 140319147, win 29200, options [mss 1460,sackOK,TS val 3547090335 ec
r 0,nop,wscale 7], length 0
21:46:19.695650 IP 10.10.1.199.1029 > 10.10.1.10.34940: Flags [R.], seq 0, ack 14031
9148, win 0, length 0
21:46:19.695664 IP 10.10.1.10.4564
8 > 10.10.1.199.5060
: Flags [S], seq 2203629201, win 29200, options [mss 1460,sackOK,TS val 3547090335 e
cr 0,nop,wscale 7], length 0
21:46:19.695775 IP 10.10.1.10.4902
8 > 10.10.1.199.2000
: Flags [S], seq 635990431, win 29200, options [mss 1460,sackOK,TS val 3547090335 ec
r 0,nop,wscale 7], length 0
21:46:19.695790 IP 10.10.1.199.2000 > 10.10.1.10.49028: Flags [R.], seq 0, ack 63599
0432, win 0, length 0
```

15. Example Filter Showing Nmap NSE Script Testing

In this example the Nmap NSE script http-enum.nse is shown testing for valid urls against an open

HTTP service.

On the Nmap machine:

```
:~$ nmap -p 80 --script=http-enum.nse targetip
```

On the target machine:

```
:~$ tcpdump -nn port 80 | grep "GET /"

GET /w3perl/ HTTP/1.1
GET /w-agora/ HTTP/1.1
GET /way-board/ HTTP/1.1
GET /web800fo/ HTTP/1.1
GET /webaccess/ HTTP/1.1
GET /webadmin/ HTTP/1.1
GET /webAdmin/ HTTP/1.1
```

16. Capture Start and End Packets of every non-local host

This example is straight out of the tcpdump man page. By selecting on the tcp-syn and tcp-fin packets we can show each established TCP conversation with timestamps but without the data. As with many filters this allows the amount of noise to be reduced in order to focus in on the information that you care about.

```
:~$ tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-
fin) != 0 and not src and dst net localnet'
```

17. Capture DNS Request and Response

Outbound DNS request to Google public DNS and the A record (ip address) response can be seen in this capture.

```
:~$ sudo tcpdump -i wlp58s0 -s0 port 53

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlp58s0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:19:06.879799 IP te
st.53852 > google-public-dns-a.google.com.domain
: 26977+ [1au] A? play.google.com. (44)
14:19:07.022618 IP google-public-
dns-a.google.com.domain > test.53852: 26977 1/0/1 A 216.58.203.110 (60)
```

18. Capture HTTP data packets

Only capture on HTTP data packets on port 80. Avoid capturing the TCP session setup (SYN / FIN / ACK).

```
tcpdump 'tcp port 80 and
(((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'
```

19. Capture with tcpdump and view in Wireshark

Parsing and analysis of full application streams such as HTTP is much easier to perform with Wireshark (or tshark) rather than tcpdump. It is often more practical to capture traffic on a remote system using tcpdump with the write file option. Then copy the pcap to the local workstation for analysis with Wireshark.

Other than manually moving the file from the remote system to the local workstation it is possible to feed the capture to Wireshark over the SSH connection in real time. This **tip is a favorite**, pipe the raw tcpdump output right into wireshark on your local machine. **Don't forget** the not port 22 so you are not capturing your SSH traffic.

```
:~$ ssh root@
remotesystem 'tcpdump -s0 -c 1000 -nn -w - not port 22' | wireshark -k -i -
```

Another tip is to use count -c on the remote tcpdump to allow the capture to finish otherwise hitting ctrl-c will not only kill tcpdump but also Wireshark and your capture.

20. Top Hosts by Packets

List the top talkers for a period of time or number of packets. Using simple command line field extraction to get the IP address, sort and count the occurrances. Capture is limited by the count option -c.

```
sudo tcpdump -nnn -t -c 200
 | cut -f 1,2,3,4 -d '.' | sort | uniq -c | sort -nr | head -n 20

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
200 packets captured
261 packets received by filter
0 packets dropped by kernel
    108 IP 10.10.211.181
     91 IP 10.10.1.30
      1 IP 10.10.1.50
```

21. Capture all the plaintext passwords

In this command we are focusing on standard plain text protocols and chosing to grep on anything user or password related. By selecting the -B5 option on grep the aim is to get the preceding 5 lines that may provide context around the captured password (hostname, ip address, system).

```
:~$ sudo tcpdump port http or port ftp or port smtp or port imap or port pop3 or por
t telnet -l -A | egrep -i -B5 'pass=|pwd=|log=|login=|user=|username=|pw=|passw=|pas
swd=|password=|pass:|user:|username:|password:|login:|pass |user '
```

22. DHCP Example

And our final tcpdump example is for monitoring DHCP request and reply. DHCP requests are seen on **port 67** and the reply is on **68**. Using the verbose parameter -v we get to see the protocol options and other details.

```
:~$ sudo tcpdump -v -n port 67 or 68

tcpdump: listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes
14:37:50.059662 IP (tos 0x10, ttl 128, id 0, offset 0, flags [none], proto UDP (17),
 length 328)
    0.0.0.0.68 > 255.255.255.255.67
: BOOTP/DHCP, Request from 00:0c:xx:xx:xx:d5, length 300, xid 0xc9779c2a, Flags [non
e]
   Client-Ethernet-Address 00:0c:xx:xx:xx:d5
   Vendor-rfc1048 Extensions
     Magic Cookie 0x63825363
     DHCP-Message Option 53, length 1: Request
     Requested-IP Option 50, length 4: 10.10.1.163
     Hostname Option 12, length 14: "test-ubuntu"
     Parameter-Request Option 55, length 16:
       Subnet-Mask, BR, Time-Zone, Default-Gateway
       Domain-Name, Domain-Name-Server, Option 119, Hostname
       Netbios-Name-Server, Netbios-Scope, MTU, Classless-Static-Route
       NTP, Classless-Static-Route-Microsoft, Static-Route, Option 252
14:37:50.059667 IP (tos 0x10, ttl 128, id 0, offset 0, flags [none], proto UDP (17),
 length 328)
    0.0.0.0.68 > 255.255.255.255.67
: BOOTP/DHCP, Request from 00:0c:xx:xx:xx:d5, length 300, xid 0xc9779c2a, Flags [non
e]
   Client-Ethernet-Address 00:0c:xx:xx:xx:d5
   Vendor-rfc1048 Extensions
     Magic Cookie 0x63825363
     DHCP-Message Option 53, length 1: Request
     Requested-IP Option 50, length 4: 10.10.1.163
     Hostname Option 12, length 14: "test-ubuntu"
     Parameter-Request Option 55, length 16:
       Subnet-Mask, BR, Time-Zone, Default-Gateway
       Domain-Name, Domain-Name-Server, Option 119, Hostname
       Netbios-Name-Server, Netbios-Scope, MTU, Classless-Static-Route
       NTP, Classless-Static-Route-Microsoft, Static-Route, Option 252
14:37:50.060780 IP (tos 0x0, ttl 64, id 53564, offset 0, flags [none], proto UDP (17
), length 339)
    10.10.1.1.67 > 10.10.1.163.68
: BOOTP/DHCP, Reply, length 311, xid 0xc9779c2a, Flags [none]
   Your-IP 10.10.1.163
   Server-IP 10.10.1.1
   Client-Ethernet-Address 00:0c:xx:xx:xx:d5
   Vendor-rfc1048 Extensions
     Magic Cookie 0x63825363
     DHCP-Message Option 53, length 1: ACK
     Server-ID Option 54, length 4: 10.10.1.1
     Lease-Time Option 51, length 4: 86400
     RN Option 58, length 4: 43200
     RB Option 59, length 4: 75600
     Subnet-Mask Option 1, length 4: 255.255.255.0
     BR Option 28, length 4: 10.10.1.255
     Domain-Name-Server Option 6, length 4: 10.10.1.1
     Hostname Option 12, length 14: "test-ubuntu"
     T252 Option 252, length 1: 10
     Default-Gateway Option 3, length 4: 10.10.1.1
```

Infrastructure

Unique solution ID: #1079
Author: n/a
Last update: 2021-02-25 13:38