Tools and their uses

Dam Security
Sunwoo Kim, Billy Endicott, Ian J Delgado, Alanna Croysdale
3 February 2023
CS 46X

Success Measures

Identification of the tools being used at each step of the security assessment. The purpose of this document is to inform of the general overview of each tool and basic commands being used, while also attaching resources that can be used for someone to repeat the process. This will be recommendations based on what we believe we need for each project. Some of these tools have far greater capabilities than what will be presented. It's in the hands of the user to research further to gain additional knowledge and information while using the tools.

Reconnaissance

Reconnaissance is one of the first five phases listed on the ec-council's website for phases of penetration testing. They describe this phase as gathering as much information about a target system as possible. This will include network topology, the operating system the target is on, applications, and user accounts. As someone new to ethical hacking it's easy to see the importance of this step. Afterall, it is better to know the target to figure out what tools we can use on the target's system.

Passive Reconnaissance

The ec-council talks about the two categories of reconnaissance. The first is passive and this could consist of publicly available information about a target. This could include gathering domain names, email addresses, IP addresses, and much more. In an article written by Shimon Brathwaite, titled "Active vs Passive Cyber Reconnaissance in Information Security." Brathwaite gives passive reconnaissance a familiar name I have heard before. OSINT is short for open-source intelligence. Brathwaite lists common tools used in the process of gaining intelligence on a target. The three listed include Google Hacking, Netcraft, and Shodan but many more are available.

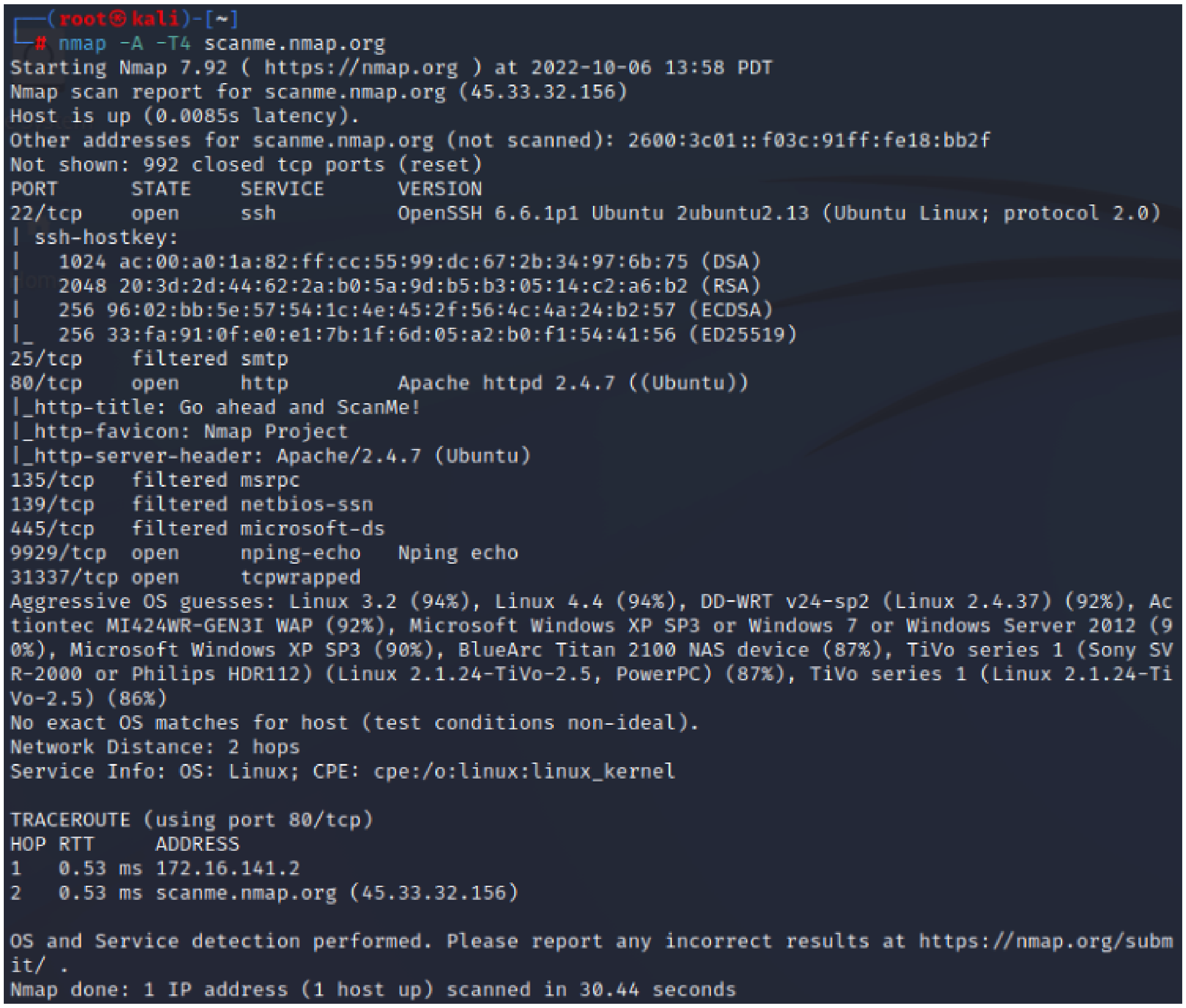
Active Reconnaissance

Active reconnaissance requires interacting with the target to gain information. This will require a set of tools to scan a network to find out information. NMAP which is short for network mapper is used to scan systems giving various information on a target. NMAP is a focus for this section and the tool we used to scan target systems and will be discussed in detail shortly. Active reconnaissance has the chance of being detected by the target.

The information gathered at this part of reconnaissance includes but is not limited to finding out if a port is opened or closed, the operating system the machine is using, the services the target is running, and discovering if the host has vulnerable applications or ports. Since this is intrusive it is vital as mentioned before not to scan anything we do not own or have permission to scan. Scanning a network can trigger intrusion detection systems and intrusion prevention systems.

NMAP (network mapping)

NMAP is the first tool introduced for our journey of reconnaissance. This tool has many features and capabilities, few of which will be discussed here. The first part of understanding nmap will begin with scanning scanme.nmap.org checking for its operating system, version, script, and traceroute. In figure 1 below the results of our first scan gave me great details about the target. This gives a great overview of the target with many details that can be a little overwhelming at first glance.

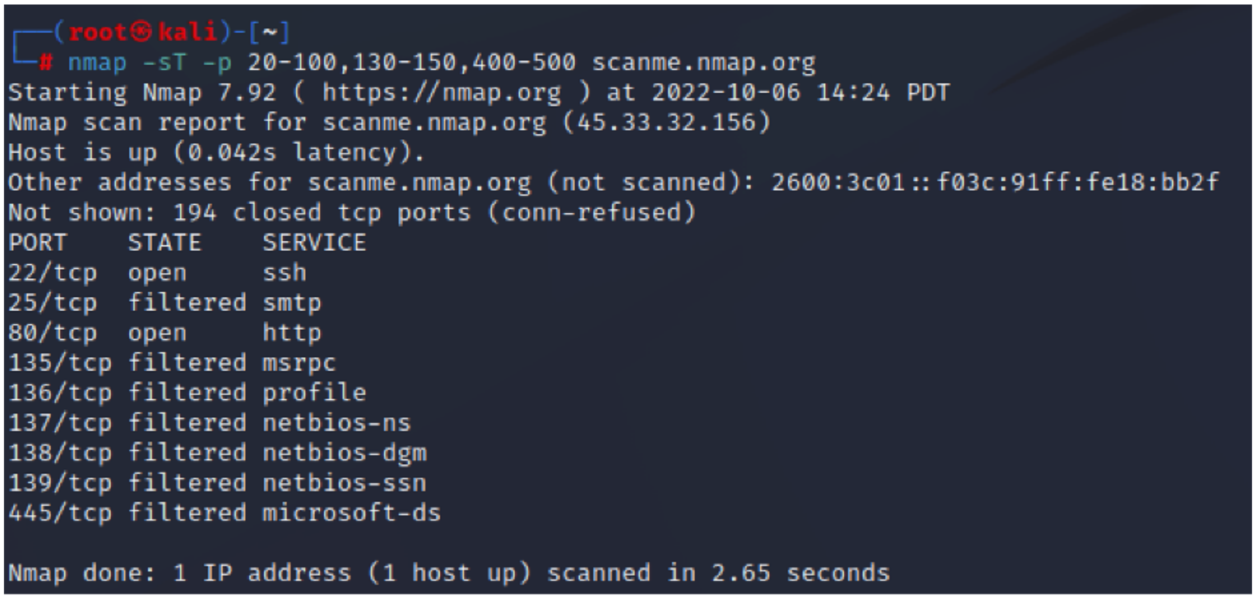


```
┌──(root㉿kali)-[~]
└─# nmap -A -T4 scanme.nmap.org
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-06 13:58 PDT
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.0085s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 992 closed tcp ports (reset)
PORT      STATE    SERVICE      VERSION
22/tcp    open     ssh          OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   1024 ac:00:a0:1a:82:ff:cc:55:99:dc:67:2b:34:97:6b:75 (DSA)
|   2048 20:3d:2d:44:62:2a:b0:5a:9d:b5:b3:05:14:c2:a6:b2 (RSA)
|   256 96:02:bb:5e:57:54:1c:4e:45:2f:56:4c:4a:24:b2:57 (ECDSA)
|_  256 33:fa:91:0f:e0:e1:7b:1f:6d:05:a2:b0:f1:54:41:56 (ED25519)
25/tcp    filtered smtp
80/tcp    open     http         Apache httpd 2.4.7 ((Ubuntu))
|_http-title: Go ahead and ScanMe!
|_http-favicon: Nmap Project
|_http-server-header: Apache/2.4.7 (Ubuntu)
135/tcp   filtered msrpc
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
9929/tcp  open     nping-echo   Nping echo
31337/tcp open     tcpwrapped
Aggressive OS guesses: Linux 3.2 (94%), Linux 4.4 (94%), DD-WRT v24-sp2 (Linux 2.4.37) (92%), Ac
tiontec MI424WR-GEN3I WAP (92%), Microsoft Windows XP SP3 or Windows 7 or Windows Server 2012 (9
0%), Microsoft Windows XP SP3 (90%), BlueArc Titan 2100 NAS device (87%), TiVo series 1 (Sony SV
R-2000 or Philips HDR112) (Linux 2.1.24-TiVo-2.5, PowerPC) (87%), TiVo series 1 (Linux 2.1.24-Ti
Vo-2.5) (86%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 80/tcp)
HOP RTT     ADDRESS
1   0.53 ms 172.16.141.2
2   0.53 ms scanme.nmap.org (45.33.32.156)

OS and Service detection performed. Please report any incorrect results at https://nmap.org/subm
it/ .
Nmap done: 1 IP address (1 host up) scanned in 30.44 seconds
```

Figure 1. nmap scan of scanme.nmap.org

NMAP options

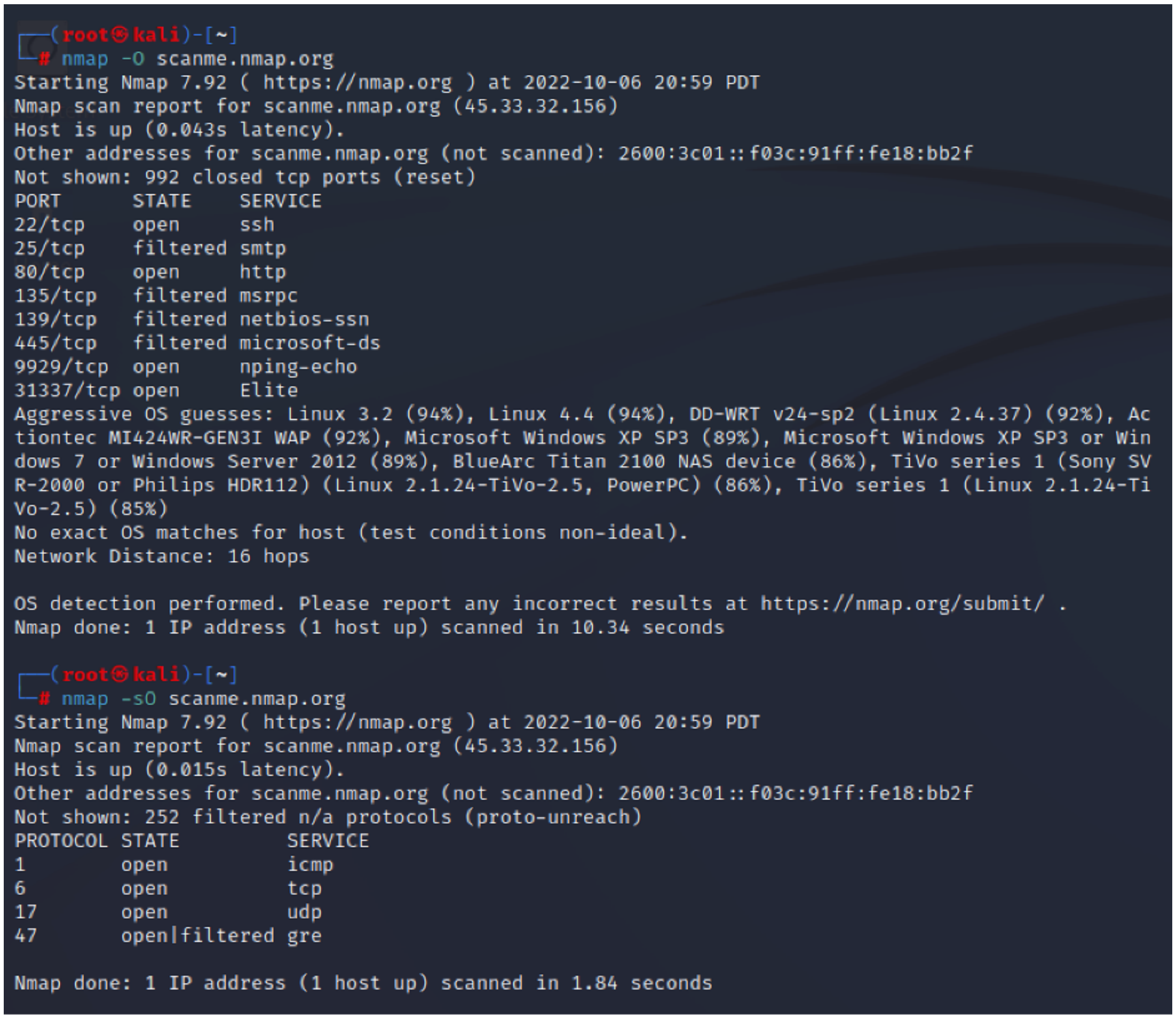
The journey of nmap and learning it’s options will be long and take some effort and will expand much further than this class. Let’s start to explore the more specific options available within nmap. First, we are learning how to scan for TCP and UDP of specific ports. One issue I encountered while trying to scan certain ports was failing to include -p <ports> I instead attempted to clump the port numbers into the port scan I was attempting. So, in the case where I was scanning for TCP connect and wanted to scan ports 20-100 I tried #nmap -sT20-100 scanme.nmap.org. This was the first example of me not reading the man pages correctly.



```
┌──(root㉿kali)-[~]
└─# nmap -sT -p 20-100,130-150,400-500 scanme.nmap.org
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-06 14:24 PDT
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.042s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 194 closed tcp ports (conn-refused)
PORT    STATE    SERVICE
22/tcp  open     ssh
25/tcp  filtered smtp
80/tcp  open     http
135/tcp filtered msrpc
136/tcp filtered profile
137/tcp filtered netbios-ns
138/tcp filtered netbios-dgm
139/tcp filtered netbios-ssn
445/tcp filtered microsoft-ds

Nmap done: 1 IP address (1 host up) scanned in 2.65 seconds
```

Figure 2. TCP connect scan of scanme.nmap.org

The next two individual actions I ran with nmap are detection of the operating system and IP protocol scan. Each of which are important. Figure 3 shows the result of the separate scans but in the same screenshot. In a video about using nmap created by Simplilearn the narrator explains and walks through about how to use nmap to find that port 445 is open. He explains that the eternal blue exploit might be a vulnerability that could be used and showed how that worked. This gave concrete evidence of how important nmap can be to scan our system as a Whitehat to ensure we fix vulnerabilities.



```
┌──(root㉿kali)-[~]
└─# nmap -O scanme.nmap.org
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-06 20:59 PDT
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.043s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 992 closed tcp ports (reset)
PORT      STATE    SERVICE
22/tcp    open     ssh
25/tcp    filtered smtp
80/tcp    open     http
135/tcp   filtered msrpc
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
9929/tcp  open     nping-echo
31337/tcp open     Elite
Aggressive OS guesses: Linux 3.2 (94%), Linux 4.4 (94%), DD-WRT v24-sp2 (Linux 2.4.37) (92%), Ac
tiontec MI424WR-GEN3I WAP (92%), Microsoft Windows XP SP3 (89%), Microsoft Windows XP SP3 or Win
dows 7 or Windows Server 2012 (89%), BlueArc Titan 2100 NAS device (86%), TiVo series 1 (Sony SV
R-2000 or Philips HDR112) (Linux 2.1.24-TiVo-2.5, PowerPC) (86%), TiVo series 1 (Linux 2.1.24-Ti
Vo-2.5) (85%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 16 hops

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.34 seconds

┌──(root㉿kali)-[~]
└─# nmap -sO scanme.nmap.org
Starting Nmap 7.92 ( https://nmap.org ) at 2022-10-06 20:59 PDT
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.015s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 252 filtered n/a protocols (proto-unreach)
PROTOCOL STATE         SERVICE
1        open          icmp
6        open          tcp
17       open          udp
47       open|filtered gre

Nmap done: 1 IP address (1 host up) scanned in 1.84 seconds
```

Figure 3. nmap OS detection scan on the top and IP protocol scan on the bottom.

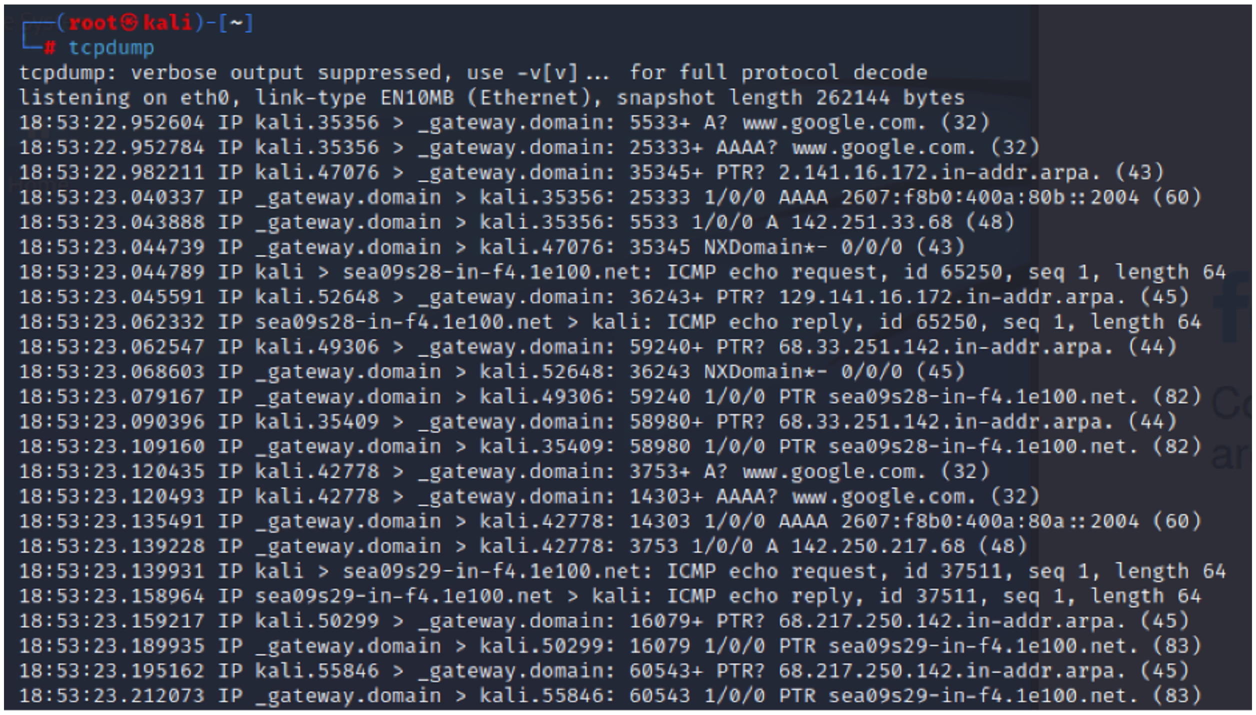
Packet capture

Capturing packets can be accomplished in multiple ways. This includes direct access to the channel of the network that includes ethernet cables, copper wire, and RF receiver. Having direct access can use different tools including a network tap, switches, and setting up a computer with dual ethernet ports to mirror a switch and capture packets as they travel through the machine. Another method is using a Wireless access point to capture packets with a WiFi card installed and set to promiscuous mode. The last method is having control of an endpoint This is the more practical case and the one we will begin investigating.

Our overall goal of capturing packets can range from checking the quality of service to checking for malicious activity that is happening on the network. The malicious actors can use it for eavesdropping on data on the network all the way to espionage. This is a key place where having physical security is important to ensure no one has access to routers, network jacks, and endpoints. Just like many topics we have covered and will cover we can't tap or eavesdrop on any network that we do not own.

tcpdump

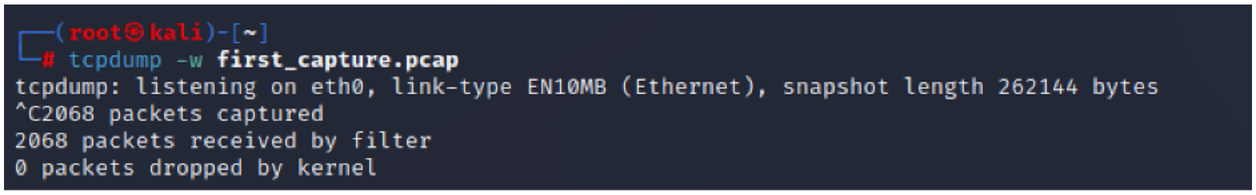
Tcpdump is one of the command line tools that is a foundational tool used to capture packets. This can be configured in many ways to capture in and outbound packets or both at the same time. This can be done while writing to a file to later filter the results in ways to best fits our needs. We will be going over different commands explained in this module. The first command I entered to play with was #tcpdump this results in immediate results being displayed to the console. This produces results at such a rapid rate that it can be overwhelming to understand what is being displayed. Plus, we don't have a way to narrow down the current capture since the results aren't going to a file.



```
┌──(root㉿kali)-[~]
└─# tcpdump
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
18:53:22.952604 IP kali.35356 > _gateway.domain: 5533+ A? www.google.com. (32)
18:53:22.952784 IP kali.35356 > _gateway.domain: 25333+ AAAA? www.google.com. (32)
18:53:22.982211 IP kali.47076 > _gateway.domain: 35345+ PTR? 2.141.16.172.in-addr.arpa. (43)
18:53:23.040337 IP _gateway.domain > kali.35356: 25333 1/0/0 AAAA 2607:f8b0:400a:80b::2004 (60)
18:53:23.043888 IP _gateway.domain > kali.35356: 5533 1/0/0 A 142.251.33.68 (48)
18:53:23.044739 IP _gateway.domain > kali.47076: 35345 NXDomain*- 0/0/0 (43)
18:53:23.044789 IP kali > sea09s28-in-f4.1e100.net: ICMP echo request, id 65250, seq 1, length 64
18:53:23.045591 IP kali.52648 > _gateway.domain: 36243+ PTR? 129.141.16.172.in-addr.arpa. (45)
18:53:23.062332 IP sea09s28-in-f4.1e100.net > kali: ICMP echo reply, id 65250, seq 1, length 64
18:53:23.062547 IP kali.49306 > _gateway.domain: 59240+ PTR? 68.33.251.142.in-addr.arpa. (44)
18:53:23.068603 IP _gateway.domain > kali.52648: 36243 NXDomain*- 0/0/0 (45)
18:53:23.079167 IP _gateway.domain > kali.49306: 59240 1/0/0 PTR sea09s28-in-f4.1e100.net. (82)
18:53:23.090396 IP kali.35409 > _gateway.domain: 58980+ PTR? 68.33.251.142.in-addr.arpa. (44)
18:53:23.109160 IP _gateway.domain > kali.35409: 58980 1/0/0 PTR sea09s28-in-f4.1e100.net. (82)
18:53:23.120435 IP kali.42778 > _gateway.domain: 3753+ A? www.google.com. (32)
18:53:23.120493 IP kali.42778 > _gateway.domain: 14303+ AAAA? www.google.com. (32)
18:53:23.135491 IP _gateway.domain > kali.42778: 14303 1/0/0 AAAA 2607:f8b0:400a:80a::2004 (60)
18:53:23.139228 IP _gateway.domain > kali.42778: 3753 1/0/0 A 142.250.217.68 (48)
18:53:23.139931 IP kali > sea09s29-in-f4.1e100.net: ICMP echo request, id 37511, seq 1, length 64
18:53:23.158964 IP sea09s29-in-f4.1e100.net > kali: ICMP echo reply, id 37511, seq 1, length 64
18:53:23.159217 IP kali.50299 > _gateway.domain: 16079+ PTR? 68.217.250.142.in-addr.arpa. (45)
18:53:23.189935 IP _gateway.domain > kali.50299: 16079 1/0/0 PTR sea09s29-in-f4.1e100.net. (83)
18:53:23.195162 IP kali.55846 > _gateway.domain: 60543+ PTR? 68.217.250.142.in-addr.arpa. (45)
18:53:23.212073 IP _gateway.domain > kali.55846: 60543 1/0/0 PTR sea09s29-in-f4.1e100.net. (83)
```

Figure 1. result of entering just tcpdump with no flags or extra options or file being created.

The next command we will use will include a method to write the results to a file name of our choosing. This time I will be first_capture.pcap. To do this we will include the flag -w after our tcpdump command. Figure two will have the results of the command line being entered. The image displays the number of packets captured as well as the packets received by the filter. If those first two numbers are different, then that means the difference in the value of packets that were received but not processed before exiting tcpdump. The last output will include the number of packets dropped by the kernel. In our instance it was zero.

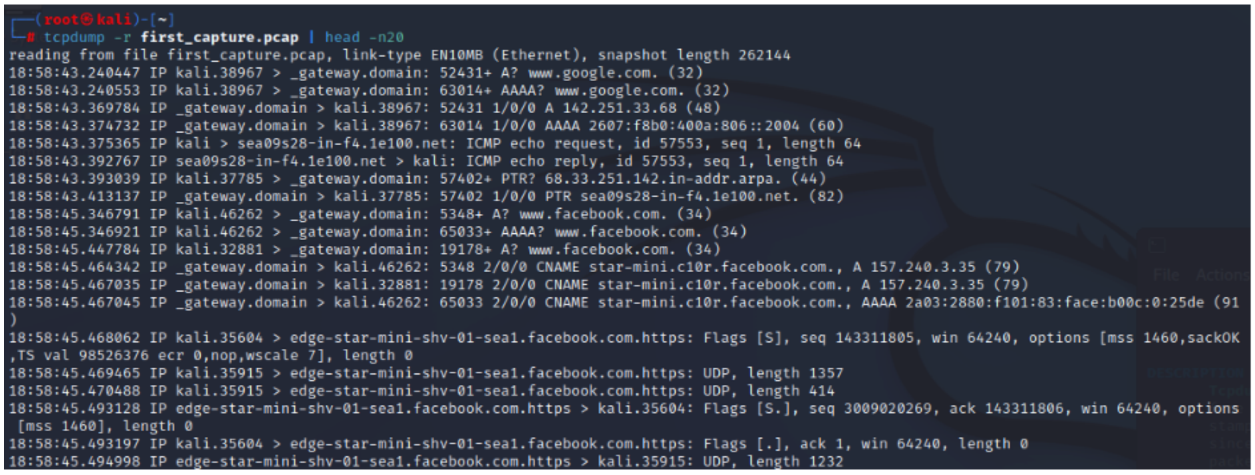


```
┌──(root㉿kali)-[~]
└─# tcpdump -w first_capture.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C2068 packets captured
2068 packets received by filter
0 packets dropped by kernel
```

Figure 2. command line directs the packets to a readable file.

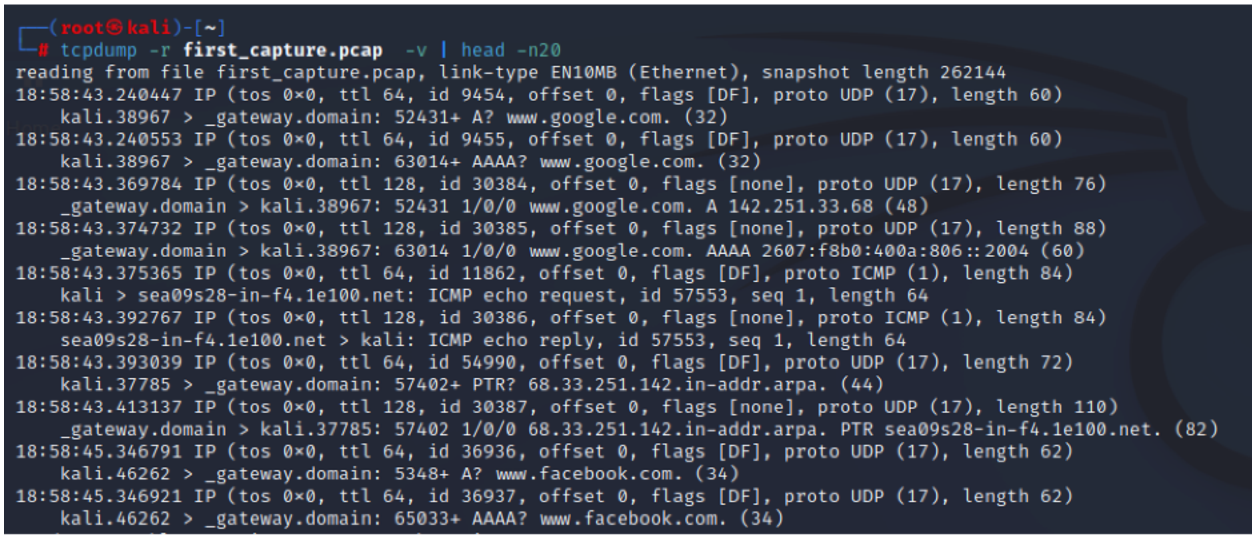
Viewing results

Viewing the results of the packets can be done by adding the flag for reading, followed by the file name, and then if we would like a verbose version of the results. Then ending with a pipe and the number of lines we would like to display for a small set of data from the file. Figures three and four will display the results we received from the first packets we captured. Figure three will be without verbose and figure four will include the verbose flag. As we can see while I was capturing packets, I went to Facebook. We can also see what method of communication we used and the size of each packet.



```
┌──(root㉿kali)-[~]
└─# tcpdump -r first_capture.pcap | head -n20
reading from file first_capture.pcap, link-type EN10MB (Ethernet), snapshot length 262144
18:58:43.240447 IP kali.38967 > _gateway.domain: 52431+ A? www.google.com. (32)
18:58:43.240553 IP kali.38967 > _gateway.domain: 63014+ AAAA? www.google.com. (32)
18:58:43.369784 IP _gateway.domain > kali.38967: 52431 1/0/0 A 142.251.33.68 (48)
18:58:43.374732 IP _gateway.domain > kali.38967: 63014 1/0/0 AAAA 2607:f8b0:400a:806::2004 (60)
18:58:43.375365 IP kali > sea09s28-in-f4.1e100.net: ICMP echo request, id 57553, seq 1, length 64
18:58:43.392767 IP sea09s28-in-f4.1e100.net > kali: ICMP echo reply, id 57553, seq 1, length 64
18:58:43.393039 IP kali.37785 > _gateway.domain: 57402+ PTR? 68.33.251.142.in-addr.arpa. (44)
18:58:43.413137 IP _gateway.domain > kali.37785: 57402 1/0/0 PTR sea09s28-in-f4.1e100.net. (82)
18:58:45.346791 IP kali.46262 > _gateway.domain: 5348+ A? www.facebook.com. (34)
18:58:45.346921 IP kali.46262 > _gateway.domain: 65033+ AAAA? www.facebook.com. (34)
18:58:45.447784 IP kali.32881 > _gateway.domain: 19178+ A? www.facebook.com. (34)
18:58:45.464342 IP _gateway.domain > kali.46262: 5348 2/0/0 CNAME star-mini.c10r.facebook.com., A 157.240.3.35 (79)
18:58:45.467035 IP _gateway.domain > kali.32881: 19178 2/0/0 CNAME star-mini.c10r.facebook.com., A 157.240.3.35 (79)
18:58:45.467045 IP _gateway.domain > kali.46262: 65033 2/0/0 CNAME star-mini.c10r.facebook.com., AAAA 2a03:2880:f101:83:face:b00c:0:25de (91
)
18:58:45.468062 IP kali.35604 > edge-star-mini-shv-01-sea1.facebook.com.https: Flags [S], seq 143311805, win 64240, options [mss 1460,sackOK
,TS val 98526376 ecr 0,nop,wscale 7], length 0
18:58:45.469465 IP kali.35915 > edge-star-mini-shv-01-sea1.facebook.com.https: UDP, length 1357
18:58:45.470488 IP kali.35915 > edge-star-mini-shv-01-sea1.facebook.com.https: UDP, length 414
18:58:45.493128 IP edge-star-mini-shv-01-sea1.facebook.com.https > kali.35604: Flags [S.], seq 3009020269, ack 143311806, win 64240, options
 [mss 1460], length 0
18:58:45.493197 IP kali.35604 > edge-star-mini-shv-01-sea1.facebook.com.https: Flags [.], ack 1, win 64240, length 0
18:58:45.494998 IP edge-star-mini-shv-01-sea1.facebook.com.https > kali.35915: UDP, length 1232
```

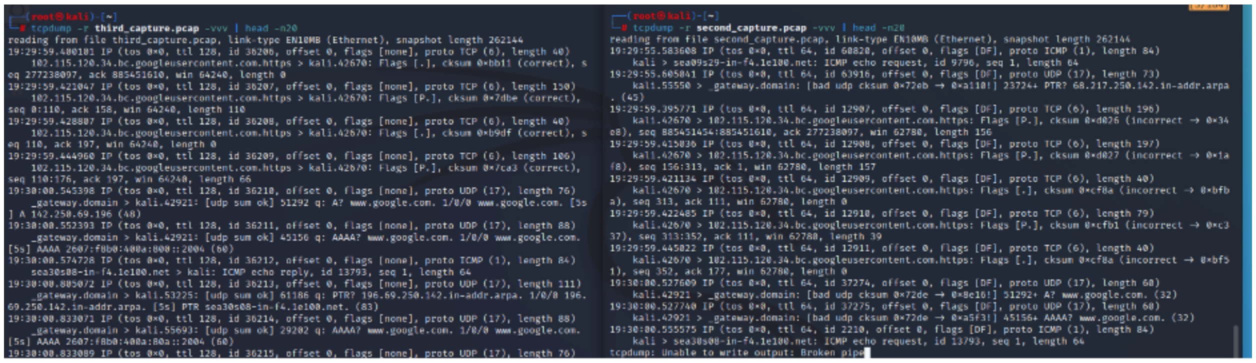
Figure 3. without viewing 20 lines without the verbose flag



```
┌──(root㉿kali)-[~]
└─# tcpdump -r first_capture.pcap  -v | head -n20
reading from file first_capture.pcap, link-type EN10MB (Ethernet), snapshot length 262144
18:58:43.240447 IP (tos 0×0, ttl 64, id 9454, offset 0, flags [DF], proto UDP (17), length 60)
    kali.38967 > _gateway.domain: 52431+ A? www.google.com. (32)
18:58:43.240553 IP (tos 0×0, ttl 64, id 9455, offset 0, flags [DF], proto UDP (17), length 60)
    kali.38967 > _gateway.domain: 63014+ AAAA? www.google.com. (32)
18:58:43.369784 IP (tos 0×0, ttl 128, id 30384, offset 0, flags [none], proto UDP (17), length 76)
    _gateway.domain > kali.38967: 52431 1/0/0 www.google.com. A 142.251.33.68 (48)
18:58:43.374732 IP (tos 0×0, ttl 128, id 30385, offset 0, flags [none], proto UDP (17), length 88)
    _gateway.domain > kali.38967: 63014 1/0/0 www.google.com. AAAA 2607:f8b0:400a:806::2004 (60)
18:58:43.375365 IP (tos 0×0, ttl 64, id 11862, offset 0, flags [DF], proto ICMP (1), length 84)
    kali > sea09s28-in-f4.1e100.net: ICMP echo request, id 57553, seq 1, length 64
18:58:43.392767 IP (tos 0×0, ttl 128, id 30386, offset 0, flags [none], proto ICMP (1), length 84)
    sea09s28-in-f4.1e100.net > kali: ICMP echo reply, id 57553, seq 1, length 64
18:58:43.393039 IP (tos 0×0, ttl 64, id 54990, offset 0, flags [DF], proto UDP (17), length 72)
    kali.37785 > _gateway.domain: 57402+ PTR? 68.33.251.142.in-addr.arpa. (44)
18:58:43.413137 IP (tos 0×0, ttl 128, id 30387, offset 0, flags [none], proto UDP (17), length 110)
    _gateway.domain > kali.37785: 57402 1/0/0 68.33.251.142.in-addr.arpa. PTR sea09s28-in-f4.1e100.net. (82)
18:58:45.346791 IP (tos 0×0, ttl 64, id 36936, offset 0, flags [DF], proto UDP (17), length 62)
    kali.46262 > _gateway.domain: 5348+ A? www.facebook.com. (34)
18:58:45.346921 IP (tos 0×0, ttl 64, id 36937, offset 0, flags [DF], proto UDP (17), length 62)
    kali.46262 > _gateway.domain: 65033+ AAAA? www.facebook.com. (34)
```

Figure 4. Results with verbose mode.

The image will include the results of capture packets going out from the second packet capture and the results of packets coming in from the third capture. For each of these, I included an even more verbose flag to show just how detailed we can get with these results. We can monitor whatever direction we desire or both directions at the same time. Figure five shows the results of in and out traffic independently.



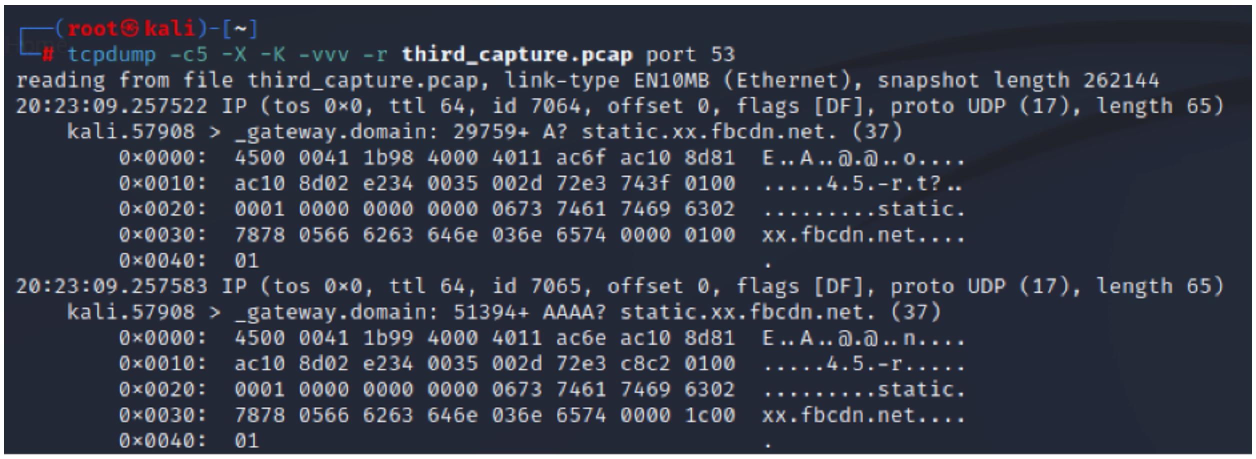
```
┌──(root㉿kali)-[~]
└─# tcpdump -r third_capture.pcap -vvv | head -n20
reading from file third_capture.pcap, link-type EN10MB (Ethernet), snapshot length 262144
19:29:59.400101 IP (tos 0×0, ttl 128, id 36206, offset 0, flags [none], proto TCP (6), length 40)
    102.115.120.34.bc.googleusercontent.com.https > kali.42670: Flags [.], cksum 0×bb11 (correct), s
eq 277238097, ack 885451610, win 64240, length 0
19:29:59.421047 IP (tos 0×0, ttl 128, id 36207, offset 0, flags [none], proto TCP (6), length 150)
    102.115.120.34.bc.googleusercontent.com.https > kali.42670: Flags [P.], cksum 0×7dbe (correct),
seq 0:110, ack 158, win 64240, length 110
19:29:59.428807 IP (tos 0×0, ttl 128, id 36208, offset 0, flags [none], proto TCP (6), length 40)
    102.115.120.34.bc.googleusercontent.com.https > kali.42670: Flags [.], cksum 0×b9df (correct), s
eq 110, ack 197, win 64240, length 0
19:29:59.444960 IP (tos 0×0, ttl 128, id 36209, offset 0, flags [none], proto TCP (6), length 106)
    102.115.120.34.bc.googleusercontent.com.https > kali.42670: Flags [P.], cksum 0×7ca3 (correct),
seq 110:176, ack 197, win 64240, length 66
19:30:00.545398 IP (tos 0×0, ttl 128, id 36210, offset 0, flags [none], proto UDP (17), length 76)
    _gateway.domain > kali.42921: [udp sum ok] 51292 q: A? www.google.com. 1/0/0 www.google.com. [5s
] A 142.250.69.196 (48)
19:30:00.552393 IP (tos 0×0, ttl 128, id 36211, offset 0, flags [none], proto UDP (17), length 88)
    _gateway.domain > kali.42921: [udp sum ok] 45156 q: AAAA? www.google.com. 1/0/0 www.google.com.
[5s] AAAA 2607:f8b0:400a:800::2004 (60)
19:30:00.574728 IP (tos 0×0, ttl 128, id 36212, offset 0, flags [none], proto ICMP (1), length 84)
    sea30s08-in-f4.1e100.net > kali: ICMP echo reply, id 13793, seq 1, length 64
19:30:00.805072 IP (tos 0×0, ttl 128, id 36213, offset 0, flags [none], proto UDP (17), length 111)
    _gateway.domain > kali.53225: [udp sum ok] 61186 q: PTR? 196.69.250.142.in-addr.arpa. 1/0/0 196.
69.250.142.in-addr.arpa. [5s] PTR sea30s08-in-f4.1e100.net. (83)
19:30:00.833071 IP (tos 0×0, ttl 128, id 36214, offset 0, flags [none], proto UDP (17), length 88)
    _gateway.domain > kali.55693: [udp sum ok] 29202 q: AAAA? www.google.com. 1/0/0 www.google.com.
[5s] AAAA 2607:f8b0:400a:80a::2004 (60)
19:30:00.833089 IP (tos 0×0, ttl 128, id 36215, offset 0, flags [none], proto UDP (17), length 76)
```

```
┌──(root㉿kali)-[~]
└─# tcpdump -r second_capture.pcap -vvv | head -n20
reading from file second_capture.pcap, link-type EN10MB (Ethernet), snapshot length 262144
19:29:55.583608 IP (tos 0×0, ttl 64, id 60820, offset 0, flags [DF], proto ICMP (1), length 84)
    kali > sea09s29-in-f4.1e100.net: ICMP echo request, id 9796, seq 1, length 64
19:29:55.605041 IP (tos 0×0, ttl 64, id 63916, offset 0, flags [DF], proto UDP (17), length 73)
    kali.55550 > _gateway.domain: [bad udp cksum 0×72eb → 0×a110!] 23724+ PTR? 68.217.250.142.in-addr.arpa
. (45)
19:29:59.395771 IP (tos 0×0, ttl 64, id 12907, offset 0, flags [DF], proto TCP (6), length 196)
    kali.42670 > 102.115.120.34.bc.googleusercontent.com.https: Flags [P.], cksum 0×d026 (incorrect → 0×34
e8), seq 885451454:885451610, ack 277238097, win 62780, length 156
19:29:59.415036 IP (tos 0×0, ttl 64, id 12908, offset 0, flags [DF], proto TCP (6), length 197)
    kali.42670 > 102.115.120.34.bc.googleusercontent.com.https: Flags [P.], cksum 0×d027 (incorrect → 0×1a
f8), seq 156:313, ack 1, win 62780, length 157
19:29:59.421134 IP (tos 0×0, ttl 64, id 12909, offset 0, flags [DF], proto TCP (6), length 40)
    kali.42670 > 102.115.120.34.bc.googleusercontent.com.https: Flags [.], cksum 0×cf8a (incorrect → 0×bfb
a), seq 313, ack 111, win 62780, length 0
19:29:59.422485 IP (tos 0×0, ttl 64, id 12910, offset 0, flags [DF], proto TCP (6), length 79)
    kali.42670 > 102.115.120.34.bc.googleusercontent.com.https: Flags [P.], cksum 0×cfb1 (incorrect → 0×c3
37), seq 313:352, ack 111, win 62780, length 39
19:29:59.445022 IP (tos 0×0, ttl 64, id 12911, offset 0, flags [DF], proto TCP (6), length 40)
    kali.42670 > 102.115.120.34.bc.googleusercontent.com.https: Flags [.], cksum 0×cf8a (incorrect → 0×bf5
1), seq 352, ack 177, win 62780, length 0
19:30:00.527609 IP (tos 0×0, ttl 64, id 37274, offset 0, flags [DF], proto UDP (17), length 60)
    kali.42921 > _gateway.domain: [bad udp cksum 0×72de → 0×8e16!] 51292+ A? www.google.com. (32)
19:30:00.527740 IP (tos 0×0, ttl 64, id 37275, offset 0, flags [DF], proto UDP (17), length 60)
    kali.42921 > _gateway.domain: [bad udp cksum 0×72de → 0×a5f3!] 45156+ AAAA? www.google.com. (32)
19:30:00.555575 IP (tos 0×0, ttl 64, id 2210, offset 0, flags [DF], proto ICMP (1), length 84)
    kali > sea30s08-in-f4.1e100.net: ICMP echo request, id 13793, seq 1, length 64
tcpdump: Unable to write output: Broken pipe
```

Figure 5. In traffic on the left and out traffic on the right.

Tcpdump filters

Next, we have the filters to narrow down the specific information we are looking for. We can filter to view UDP or TCP, what port we want, and the port range. The filter can even include Boolean searches to include AND, OR, and even NOT. This is where things become interesting and very specific and helps when we have a goal in mind. The following images will include examples of the power we have with filters. The first filter I picked shows data from only port 53 which is DNS. I also included a few new flags I haven't discussed which are -c5, -X, -K. The first one is to only show a count of 5 results. The X flag is used to print the hex and ascii dump and K flag not to validate the checksum.

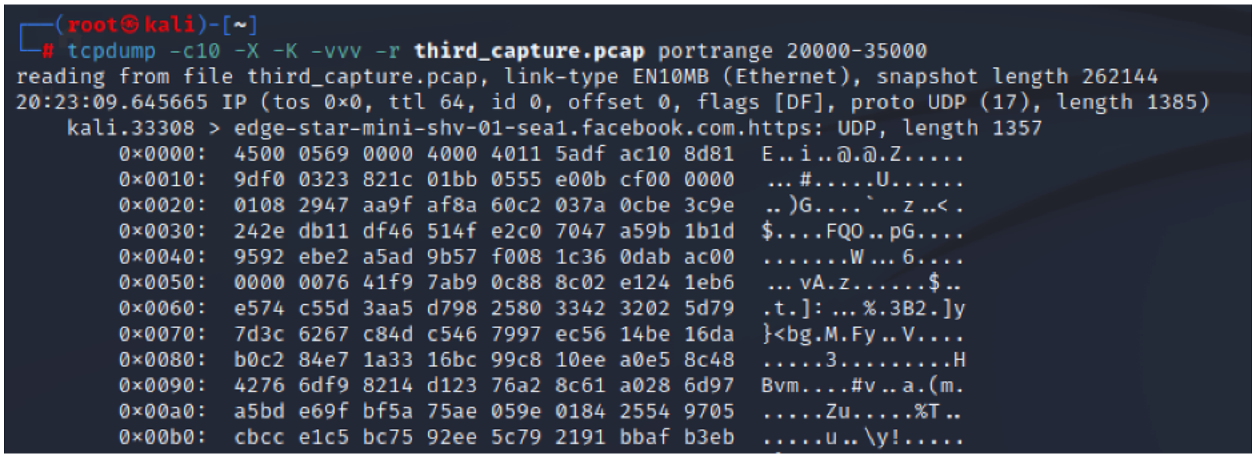


```
┌──(root㉿kali)-[~]
└─# tcpdump -c5 -X -K -vvv -r third_capture.pcap port 53
reading from file third_capture.pcap, link-type EN10MB (Ethernet), snapshot length 262144
20:23:09.257522 IP (tos 0×0, ttl 64, id 7064, offset 0, flags [DF], proto UDP (17), length 65)
    kali.57908 > _gateway.domain: 29759+ A? static.xx.fbcdn.net. (37)
        0×0000:  4500 0041 1b98 4000 4011 ac6f ac10 8d81  E..A..@.@..o....
        0×0010:  ac10 8d02 e234 0035 002d 72e3 743f 0100  .....4.5.-r.t?..
        0×0020:  0001 0000 0000 0000 0673 7461 7469 6302  .........static.
        0×0030:  7878 0566 6263 646e 036e 6574 0000 0100  xx.fbcdn.net....
        0×0040:  01                                       .
20:23:09.257583 IP (tos 0×0, ttl 64, id 7065, offset 0, flags [DF], proto UDP (17), length 65)
    kali.57908 > _gateway.domain: 51394+ AAAA? static.xx.fbcdn.net. (37)
        0×0000:  4500 0041 1b99 4000 4011 ac6e ac10 8d81  E..A..@.@..n....
        0×0010:  ac10 8d02 e234 0035 002d 72e3 c8c2 0100  .....4.5.-r.....
        0×0020:  0001 0000 0000 0000 0673 7461 7469 6302  .........static.
        0×0030:  7878 0566 6263 646e 036e 6574 0000 1c00  xx.fbcdn.net....
        0×0040:  01                                       .
```

Figure 6. The first example of filters. This shows DNS packet capture with hex and ascii dump.

Conclusion

Figure seven shows the example of filtering for just a port range. This will show the results of all results if they are included in that range. We can further narrow the search down by adding tcp/udp of a set of selected ports as well as a src and dst. This module shows a vast array of different options used to capture packets. Then after capturing the packets, we have the tools to filter in whatever manner we need. The skill to manipulate the results seems endless. The next step is learning to use the data in a way to help protect our network.



```
┌──(root㉿kali)-[~]
└─# tcpdump -c10 -X -K -vvv -r third_capture.pcap portrange 20000-35000
reading from file third_capture.pcap, link-type EN10MB (Ethernet), snapshot length 262144
20:23:09.645665 IP (tos 0×0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 1385)
    kali.33308 > edge-star-mini-shv-01-sea1.facebook.com.https: UDP, length 1357
        0×0000:  4500 0569 0000 4000 4011 5adf ac10 8d81  E..i..@.@.Z.....
        0×0010:  9df0 0323 821c 01bb 0555 e00b cf00 0000  ...#.....U......
        0×0020:  0108 2947 aa9f af8a 60c2 037a 0cbe 3c9e  ..)G....`..z..<.
        0×0030:  242e db11 df46 514f e2c0 7047 a59b 1b1d  $....FQO..pG....
        0×0040:  9592 ebe2 a5ad 9b57 f008 1c36 0dab ac00  .......W...6....
        0×0050:  0000 0076 41f9 7ab9 0c88 8c02 e124 1eb6  ...vA.z......$..
        0×0060:  e574 c55d 3aa5 d798 2580 3342 3202 5d79  .t.]:...%.3B2.]y
        0×0070:  7d3c 6267 c84d c546 7997 ec56 14be 16da  }<bg.M.Fy..V....
        0×0080:  b0c2 84e7 1a33 16bc 99c8 10ee a0e5 8c48  .....3.........H
        0×0090:  4276 6df9 8214 d123 76a2 8c61 a028 6d97  Bvm....#v..a.(m.
        0×00a0:  a5bd e69f bf5a 75ae 059e 0184 2554 9705  .....Zu.....%T..
        0×00b0:  cbcc e1c5 bc75 92ee 5c79 2191 bbaf b3eb  .....u..\y!.....
```

Figure 7. filter showing just results from port 20000-35000

Works Cited

Brathwaite, Shimon. “Active Vs Passive Cybersecurity Reconnaissance in Information Security.” SecurityMadeSimple, 23 Feb. 2021, www.securitymadesimple.org/cybersecurity-blog/active-vs-passive-cyber-reconnaissance-in-information-security.

“Understanding the Five Phases of the Penetration Testing Process.” *Cybersecurity Exchange*, 1 Aug. 2022, www.eccouncil.org/cybersecurity-exchange/penetration-testing/penetration-testing-phases.

undefined [Simplilearn]. “Nmap Tutorial for Beginners | How to Scan Your Network Using Nmap | Ethical Hacking | Simplilearn.” YouTube, 9 Apr. 2022, www.youtube.com/watch?v=NtPcoDtetvk.