

- A. Kali's MAC address from eth0 is 00:0c:29:7a:77:84
- B. Kali's main interface IP address 192.168.60.128
- C. Metasploitable's main MAC address is fe80::20c::29ff:feff:ae4d/64
- D. Metasploitable's main IP address is 192.168.60.129
- E. Kali's routing table

```
(kali㉿kali)-[~]
$ netstat -r
Kernel IP routing table
Destination        Gateway            Genmask           Flags     MSS Window  irtt Iface
e
default            192.168.60.2      0.0.0.0           UG        0 0      0 eth0
192.168.60.0       0.0.0.0           255.255.255.0     U        0 0      0 eth0

(kali㉿kali)-[~]
$ netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags     MSS Window  irtt Iface
0.0.0.0            192.168.60.2      0.0.0.0           UG        0 0      0 eth0
192.168.60.0       0.0.0.0           255.255.255.0     U        0 0      0 eth0
```

- F. Kali's ARP cache

```
(kali㉿kali)-[~]
$ arp -n
Address            HWtype  HWaddress         Flags Mask          Iface
192.168.60.254     ether   00:50:56:fa:6f:3f C               eth0
192.168.60.2       ether   00:50:56:ee:bc:c1 C               eth0

(kali㉿kali)-[~]
$ arp
Address            HWtype  HWaddress         Flags Mask          Iface
192.168.60.254     ether   00:50:56:fa:6f:3f C               eth0
192.168.60.2       ether   00:50:56:ee:bc:c1 C               eth0
```

- G. Metasploitable's routing table

```
msfadmin@metasploitable:~$ netstat -r
Kernel IP routing table
Destination        Gateway            Genmask           Flags     MSS Window  irtt Iface
192.168.60.0       *                 255.255.255.0     U        0 0      0 eth0
default            192.168.60.2      0.0.0.0           UG        0 0      0 eth0
msfadmin@metasploitable:~$ netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags     MSS Window  irtt Iface
192.168.60.0       0.0.0.0           255.255.255.0     U        0 0      0 eth0
0.0.0.0            192.168.60.2      0.0.0.0           UG        0 0      0 eth0
msfadmin@metasploitable:~$
```

H. Metasploitable's ARP cache

```
msfadmin@metasploitable:~$ arp
Address          HWtype  HWaddress      Flags Mask    Iface
192.168.60.2     ether   00:50:56:EE:BC:C1  C             eth0
192.168.60.254   ether   00:50:56:FA:6F:3F  C             eth0
msfadmin@metasploitable:~$ arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
192.168.60.2     ether   00:50:56:EE:BC:C1  C             eth0
192.168.60.254   ether   00:50:56:FA:6F:3F  C             eth0
msfadmin@metasploitable:~$
```

- I. To which MAC address should Metasploitable send the TCP SYN packet to get the whole HTTP query started? Explain why.

Metasploitable should send the TCP SYN packet to the MAC address of the Gateway, 00:50:56:EE:BC:C1 because this is the MAC address of 192.168.60.2, the Gateway as identified by netstat. The packet will be wrapped with 00:50:56:EE:BC:C1 as the destination hardware address, and the gateway will pass it on until the packet gets where it is supposed to go.

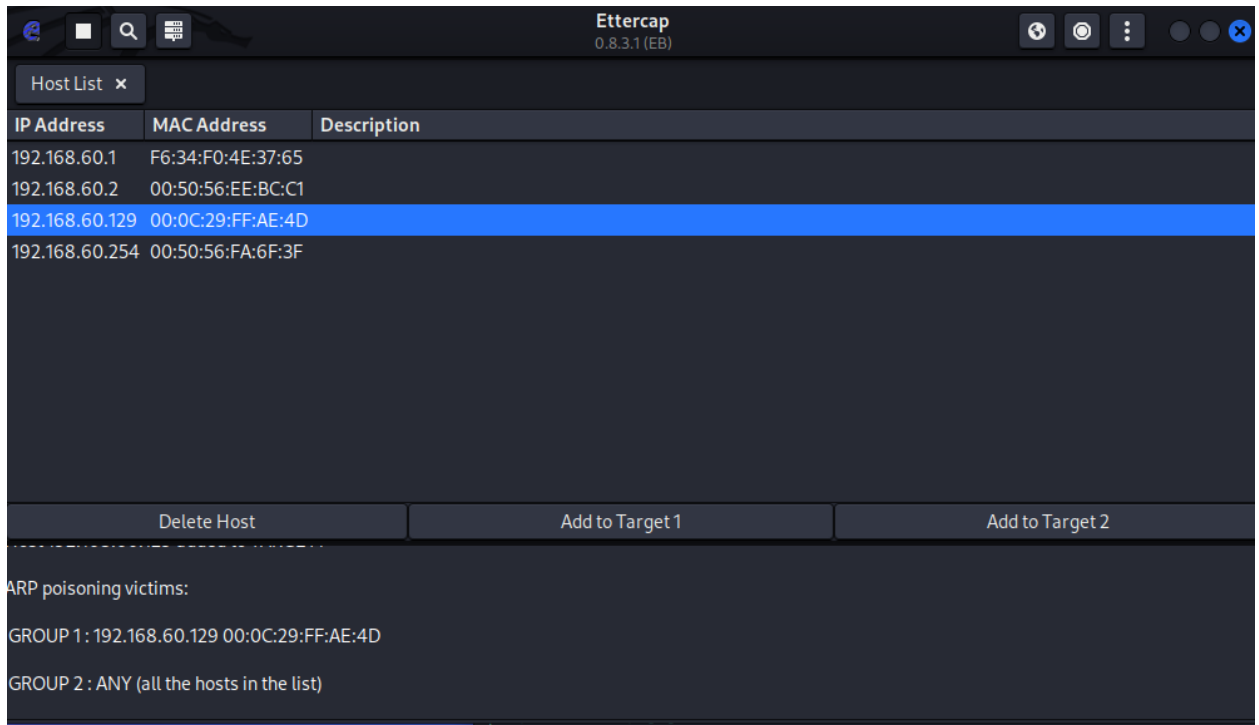
- J. Capturing "tcp port http" on wireshark, I get TCP packets going back and forth from 192.168.60.128 to 142.250.191.195 which must be <http://cs3338.jeffondich.com/>.

Wireshark packet capture showing TCP traffic between 192.168.60.128 and 142.250.191.195. The interface is "any (tcp port http)". The packet list shows a series of TCP packets, including a SYN packet (Seq=1, Ack=1, Win=63791) and an ACK packet (Seq=1, Ack=2, Win=0). The packet details pane shows the selected packet (Frame 2) with the following information:

- Source Port: 80
- Destination Port: 59428
- [Stream index: 0]
- [Conversation completeness: Incomplete (20)]
- [TCP Segment Len: 0]
- Sequence Number: 1 (relative sequence number)
- Sequence Number (raw): 3115491275
- [Next Sequence Number: 1 (relative sequence number)]
- Acknowledgment Number: 2 (relative ack number)
- Acknowledgment number (raw): 1605861736

Yes we do see packets, but only TCP ones

K. We ARP poisoned



L. Metasploitable's ARP cache has changed. The ARP cache of the Gateway's IP, 192.168.60.2, now has Kali's MAC address instead of the actual one. All MAC addresses in the ARP cache are Kali. Metasploitable now thinks Kali is the gateway.

```
msfadmin@metasploitable:~$ netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags   MSS Window  irtt Iface
192.168.60.0        0.0.0.0            255.255.255.0     U        0  0          0 eth0
0.0.0.0             192.168.60.2       0.0.0.0           UG        0  0          0 eth0
msfadmin@metasploitable:~$ _

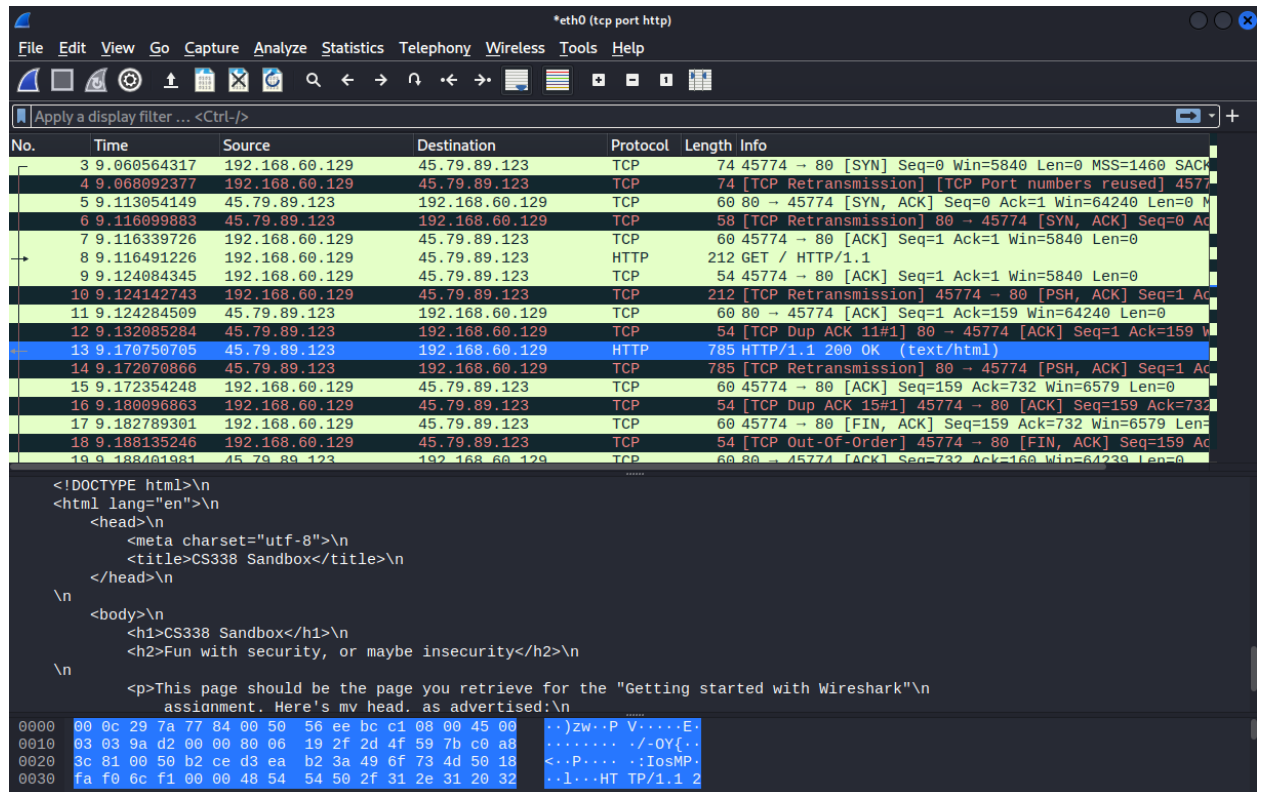
msfadmin@metasploitable:~$ arp
Address            HWtype  HWaddress          Flags Mask          Iface
192.168.60.1       ether   00:0C:29:7A:77:84   C         C         eth0
192.168.60.254     ether   00:0C:29:7A:77:84   C         C         eth0
192.168.60.2       ether   00:0C:29:7A:77:84   C         C         eth0
```

M. Predict what will happen if you execute "curl http://cs338.jeffondich.com/" on Metasploitable now. Specifically, to what MAC address will Metasploitable send the TCP SYN packet? Explain why.

Our MAC address is now the Gateway's MAC address because we made all MAC addresses in the ARP cache us. So when Metasploitable sends a packet, it is wrapped with the destination hardware address for Kali. Kali receives the packet, and forwards it to the actual gateway which ends up going to the website, and when the packet comes back it routes through Kali again.

N. Yeah we started the Wireshark capture again

- O. Yes we are correct, we can now see the HTTP packets and read them under “Line-based text data.” We have the same TCP things as before, but also we can now pick up the HTTP packets because we are the gateway. For the HTTP packets, we can see the “GET / HTTP/1.1” and then the “HTTP/1.1 200 OK” response from the website, and we can read the corresponding HTML code by looking at this packet.



The image shows a Wireshark packet capture window titled "*eth0 (tcp port http)". The packet list on the left shows a series of TCP and HTTP packets. Packet 13 is selected, showing an HTTP 200 OK response. The packet details pane on the right shows the structure of the HTTP response, including the status line "HTTP/1.1 200 OK (text/html)" and the HTML body content. The packet bytes pane at the bottom shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
3	9.060564317	192.168.60.129	45.79.89.123	TCP	74	45774 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK
4	9.068092377	192.168.60.129	45.79.89.123	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 4577
5	9.113054149	45.79.89.123	192.168.60.129	TCP	60	80 → 45774 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
6	9.116099883	45.79.89.123	192.168.60.129	TCP	58	[TCP Retransmission] 80 → 45774 [SYN, ACK] Seq=0 Ac
7	9.116339726	192.168.60.129	45.79.89.123	TCP	60	45774 → 80 [ACK] Seq=1 Ack=1 Win=5840 Len=0
8	9.116491226	192.168.60.129	45.79.89.123	HTTP	212	GET / HTTP/1.1
9	9.124084345	192.168.60.129	45.79.89.123	TCP	54	45774 → 80 [ACK] Seq=1 Ack=1 Win=5840 Len=0
10	9.124142743	192.168.60.129	45.79.89.123	TCP	212	[TCP Retransmission] 45774 → 80 [PSH, ACK] Seq=1 Ar
11	9.124284509	45.79.89.123	192.168.60.129	TCP	60	80 → 45774 [ACK] Seq=1 Ack=159 Win=64240 Len=0
12	9.132085284	45.79.89.123	192.168.60.129	TCP	54	[TCP Dup ACK 11#1] 80 → 45774 [ACK] Seq=1 Ack=159
13	9.170750705	45.79.89.123	192.168.60.129	HTTP	785	HTTP/1.1 200 OK (text/html)
14	9.172070866	45.79.89.123	192.168.60.129	TCP	785	[TCP Retransmission] 80 → 45774 [PSH, ACK] Seq=1 Ac
15	9.172354248	192.168.60.129	45.79.89.123	TCP	60	45774 → 80 [ACK] Seq=159 Ack=732 Win=6579 Len=0
16	9.180096863	192.168.60.129	45.79.89.123	TCP	54	[TCP Dup ACK 15#1] 45774 → 80 [ACK] Seq=159 Ack=732
17	9.182789301	192.168.60.129	45.79.89.123	TCP	60	45774 → 80 [FIN, ACK] Seq=159 Ack=732 Win=6579 Len=
18	9.188135246	192.168.60.129	45.79.89.123	TCP	54	[TCP Out-Of-Order] 45774 → 80 [FIN, ACK] Seq=159 Ac
19	9.188401981	45.79.89.123	192.168.60.129	TCP	60	80 → 45774 [ACK] Seq=732 Ack=160 Win=64230 Len=0

```
<!DOCTYPE html>\n<html lang="en">\n  <head>\n    <meta charset="utf-8">\n    <title>CS338 Sandbox</title>\n  </head>\n  \n  <body>\n    <h1>CS338 Sandbox</h1>\n    <h2>Fun with security, or maybe insecurity</h2>\n  \n  <p>This page should be the page you retrieve for the "Getting started with Wireshark"\n    assignment. Here's mv head, as advertised:\n  </p>\n</body>\n</html>
```

- P. Explain:
- When we started ARP poisoning we changed all of the MAC addresses in the ARP cache to our MAC address. This means instead of sending packets to the real gateway to send them off the local network, Metasploitable is sending them to us and we read them before forwarding them to the real gateway. Then, the gateway sends the packets back to us because we are the ones that sent the request through, and so we see everything.
- Q. If we made a detector, we could have it make sure that the MAC addresses for each device in the ARP cache are unique. A false positive that could arise from this is if one device somehow had multiple connections to the network. We could prevent this by only checking that the MAC address of the gateway is not duplicated in the network. Also there could be a false positive if the randomly generated MAC address of some program like Kali, that generates in a certain range, could collide with another iteration of Kali. [This question on vmware.com](#)

asks a similar question if virtual machine generated MAC addresses could end up being the same, and yes they could if the number of virtual machines exceeds the number of addresses the server can assign to them. But checking for duplicates against specifically the gateway could still work to stop PITM attacks. In the case of duplicates due to too many virtual machines, the computer just looking for duplicates might accidentally send a warning about a PITM attack when it's actually just someone using an absolutely ridiculous number of virtual machines.