ARP_Attack- Lab 3

By Hari Kishan Reddy (ha2755)

Task 1: ARP Cache Poisoning (20 points)

Task 1.A (using ARP request). (6 points)

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.

Yes, The Attack is successful, and we can see that the Arp entries of host A is modified/spoofed.

Final objective: On Host A, we should get Arp entry as
IP B, MAC M

Code snippet:

cloud.digitalocean.com

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

# Construct Ethernet frame
E = Ether(
src = '02:42:0a:09:00:69', # MAC M
dst = '02:42:0a:09:00:05') # MAC A

# Construct ARP request packet
A = ARP(
op=1,
pdst='10.9.0.5',             # IP A
hwdst='02:42:0a:09:00:69',   # MAC M
psrc='10.9.0.6')             # IP B

# op=1 for ARP request

# Combine Ethernet frame and ARP request packet
pkt = E / A

# Send the packet to host A
sendp(pkt)

~
~
~
~
```

Code Explanation:

1. Importing Scapy: The code begins by importing Scapy, a powerful packet manipulation library in Python.

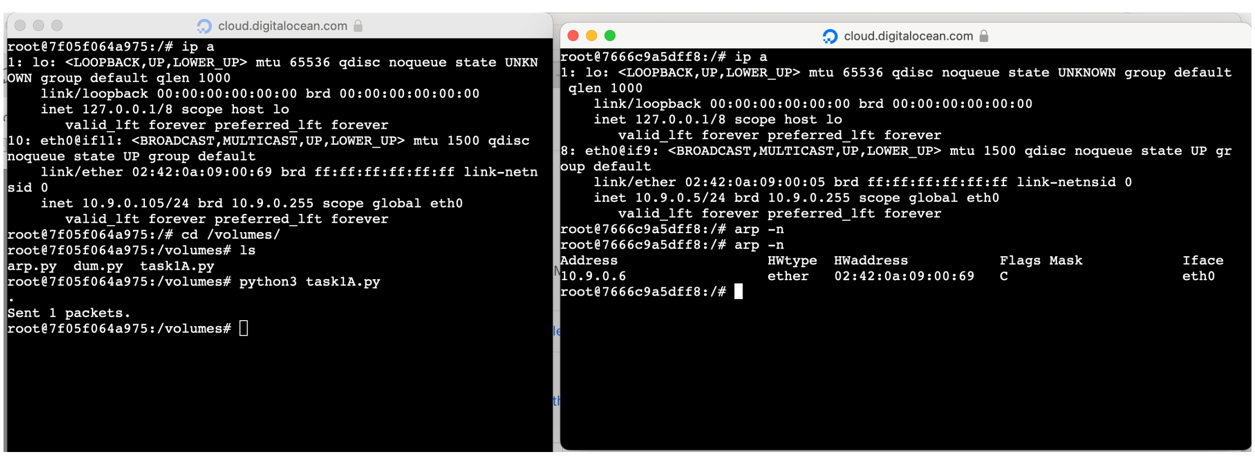
2. Constructing Ethernet Frame: The `Ether` function is used to construct an Ethernet frame. It specifies the source MAC address (`src`) as the MAC address of Host M (`02:42:0a:09:00:69`) and the destination MAC address (`dst`) as the MAC address of Host A (`02:42:0a:09:00:05`).

3. Constructing ARP Request Packet: The `ARP` function is used to construct an ARP request packet. It sets the operation code (`op`) to 1, indicating an ARP request. Other parameters include the destination IP address (`pdst`) as the IP address of Host A (`10.9.0.5`), the destination MAC address (`hwdst`) as the MAC address of Host M (`02:42:0a:09:00:69`), and the source IP address (`psrc`) as the IP address of Host B (`10.9.0.6`).

4. Combining Ethernet Frame and ARP Packet: The constructed Ethernet frame and ARP request packet are combined using the `/` operator to create a single packet (`pkt`).

5. Sending the Packet: The `sendp` function is used to send the packet (`pkt`) to Host A. This sends the ARP request packet to Host A, attempting to map Host B's IP address to Host M's MAC address in Host A's ARP cache.

Executing python code on Host M and checking ARP Tables on Host A:



Observation & Explanation:

We used Scapy to create an ARP request that tricked Host A into believing that Host M was associated with Host B's IP address. This was accomplished by setting the source MAC address to that of Host M and the source IP address to that of Host B. The ARP request effectively misled Host A, redirecting the traffic meant for Host B to Host M.

Attack is successful and we can see that the IP address of B is mapped to M's MAC address in Host A's ARP tables.

Task 1.B (using ARP reply) (7 points)

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. Try the attackunder the following two scenarios, and report the results of your attack:

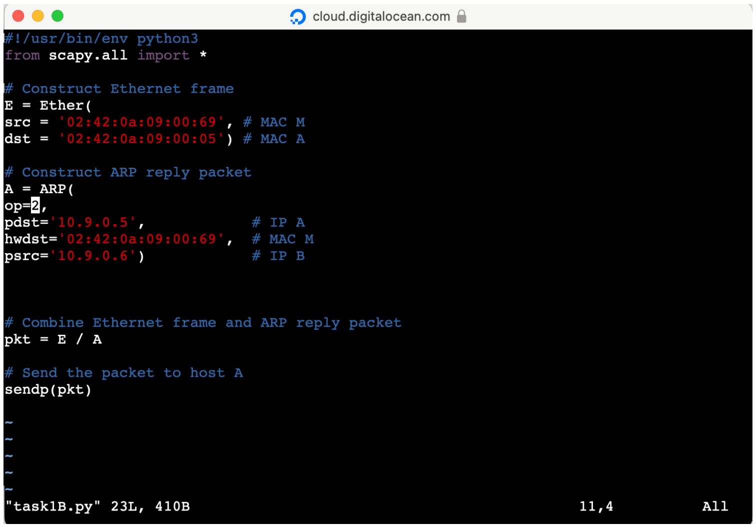
– Scenario 1: B's IP is already in A's cache.

– Scenario 2: B's IP is not in A's cache. You can use the command "arp -d a.b.c.d" to remove the ARP cache entry for the IP address a.b.c.d.

Scenario 1:

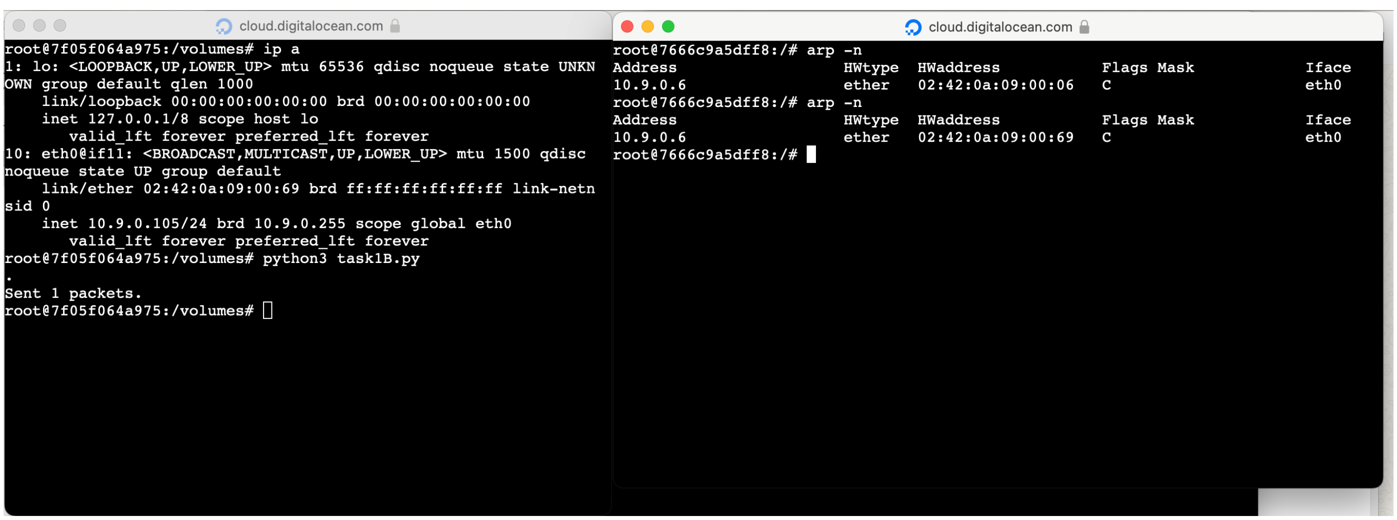
Code snippet I used to construct and send ARP reply packet to Host A by mapping Ip address of host B with MAC M.

Code:



Code Explanation:
Same as task 1A, but since we need to send ARP reply, we need to change the operation code. Op = 2.

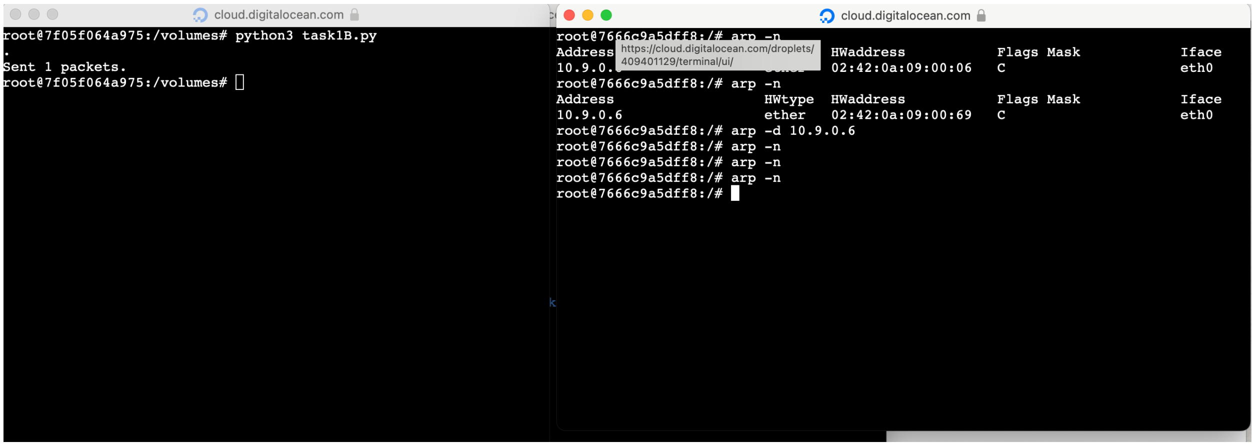
Executing python code on Host M and checking ARP Tables on Host A:



Observation & Explanation: When Host B's IP was already present in Host A's ARP cache, the spoofing was immediately effective, and the traffic was rerouted through Host M.

Attack is successful.

Scenario 2:



Explanation & Observation:

The ARP reply packets were crafted to deceive Host A into associating Host B's IP address with the Attacker's MAC address. However, when no cache entry existed for Host B's IP, repeated ARP replies failed to poison Host A's ARP cache.

This observation suggests a possible protective mechanism or configuration on Host A that resists ARP table updates without prior communication. Such protective measures can enhance network security by detecting and mitigating ARP spoofing attacks.

Attack isn’t successful.

Task 1.C (using ARP gratuitous message) (7 points)

On host M, construct an ARP gratuitous packet, and use it to map B’s IP address to M’sMAC address. Please launch the attack under the same two scenarios as those described in Task 1.B. ARP gratuitous packet is a special ARP request packet. It is usedwhen a host machine needs to update outdated information on all the other machine’s ARP cache.

Code snippet:

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

# Construct Ethernet frame
E = Ether(
    src='02:42:0a:09:00:69',  # MAC M
    dst='ff:ff:ff:ff:ff:ff')  # Broadcast MAC address

# Construct ARP gratuitous packet
A = ARP(
    op=2,  # ARP reply
    psrc='10.9.0.6',  # IP B
    hwsrc='02:42:0a:09:00:69',  # MAC M
    pdst='10.9.0.6',  # IP B (same as source IP)
    hwdst='ff:ff:ff:ff:ff:ff')  # Broadcast MAC address

# Combine Ethernet frame and ARP packet
pkt = E / A

# Send the gratuitous ARP packet to the network
sendp(pkt)

~
~
~
~
~
~
"task1C.py" 22L, 533B                                        22,0-1         All
```

Code Explanation:
same as task1B, but we change the ethernet destination and hwdst address to ff:ff:ff:ff:ff:ff:ff (broadcast address since we are sending gratuitous ARP packet)

Scenario 1:

Executing python code on Host M and checking ARP Tables on Host A:

```
root@7f05f064a975:/volumes# python3 task1C.py
.
Sent 1 packets.
root@7f05f064a975:/volumes#
```

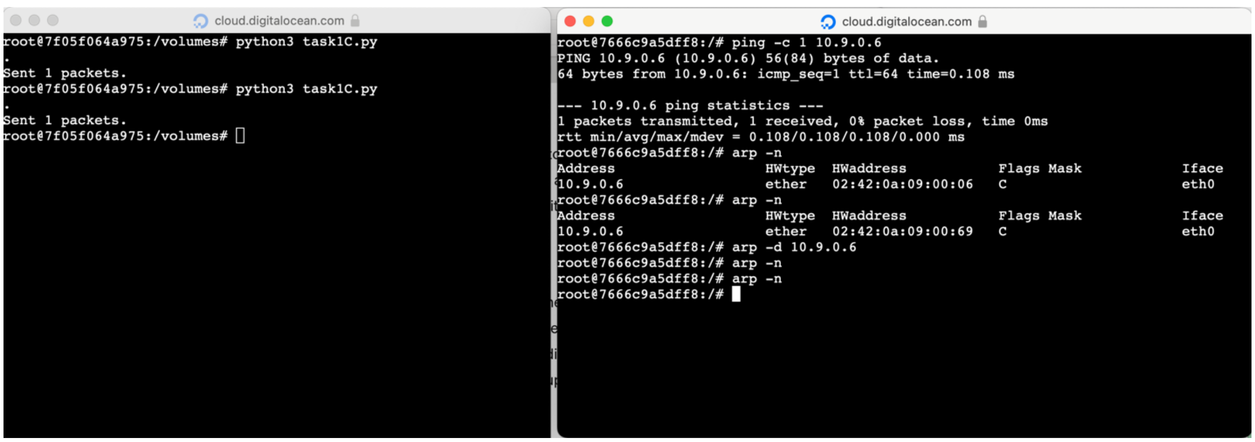
```
root@7666c9a5dff8:/# ping -c 1 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.108 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.108/0.108/0.108/0.000 ms
root@7666c9a5dff8:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@7666c9a5dff8:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@7666c9a5dff8:/#
```

Observation & Explanation:

The gratuitous ARP packet successfully updated Host A's ARP cache, demonstrating its potential for network disruption even when an entry for Host B already existed. This highlights the effectiveness of gratuitous ARP messages in manipulating ARP caches and potentially redirecting network traffic.
Attack is successful.

Scenario 2:

Executing python code on Host M and checking ARP Tables on Host A:



Explanation and Observation:

In scenario 2, like Task 1.B, the sending of gratuitous ARP messages did not lead to an ARP cache update on Host A when no prior entry existed for Host B. This suggests a level of ARP cache integrity under certain conditions, where Host A did not accept gratuitous ARP messages to update its cache without prior communication.

Despite sending gratuitous ARP messages in scenario 2, Host A's ARP cache remained unchanged when no prior entry existed for Host B. This observation indicates that Host A's ARP cache may have protective mechanisms or configurations that resist updating from gratuitous ARP messages without prior communication or verification.

Attack isn't successful.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Task 2: MITM Attack

We carried out a man-in-the-middle attack and observed the redirection of ICMP traffic between Hosts A and B through Host M, confirming the efficacy of ARP cache poisoning for intercepting network traffic.

Task 2.1: Launch the ARP Cache Poisoning Attack

The first step involved executing an ARP cache poisoning attack to deceive both Hosts A and B into sending their traffic through Host M. This was achieved by sending malicious ARP replies to both hosts, falsely associating the MAC address of Host M with the IP addresses of the other hosts.

cloud.digitalocean.com

```
from scapy.all import ARP, send

def arp_poison(target_ip, target_mac, host_ip, host_mac):
    # Construct ARP reply packet for ARP cache poisoning
    arp_reply = ARP(pdst=target_ip, hwdst=target_mac, psrc=host_ip, hwsrc=host_mac, op=2)

    # Send ARP reply packet to poison the target's ARP cache
    send(arp_reply, verbose=False)
    print(f"ARP reply sent to {target_ip} to poison cache")

# Define host and target information
host_m_ip = "10.9.0.105"  # IP of Host M
host_m_mac = "02:42:0a:09:00:69"  # MAC address of Host M

host_a_ip = "10.9.0.5"  # IP of Host A
host_a_mac = "02:42:0a:09:00:05"  # MAC address of Host A

host_b_ip = "10.9.0.6"  # IP of Host B
host_b_mac = "02:42:0a:09:00:06"  # MAC address of Host B

# Poison Host A's ARP cache
arp_poison(host_a_ip, host_a_mac, host_b_ip, host_m_mac)

# Poison Host B's ARP cache
arp_poison(host_b_ip, host_b_mac, host_a_ip, host_m_mac)

"task2.py" 26L, 905B                                                    1,1
```

Code Explanation:

1. Importing Scapy Modules: The code starts by importing the necessary Scapy modules (`ARP` and `send`) for constructing ARP packets and sending them.

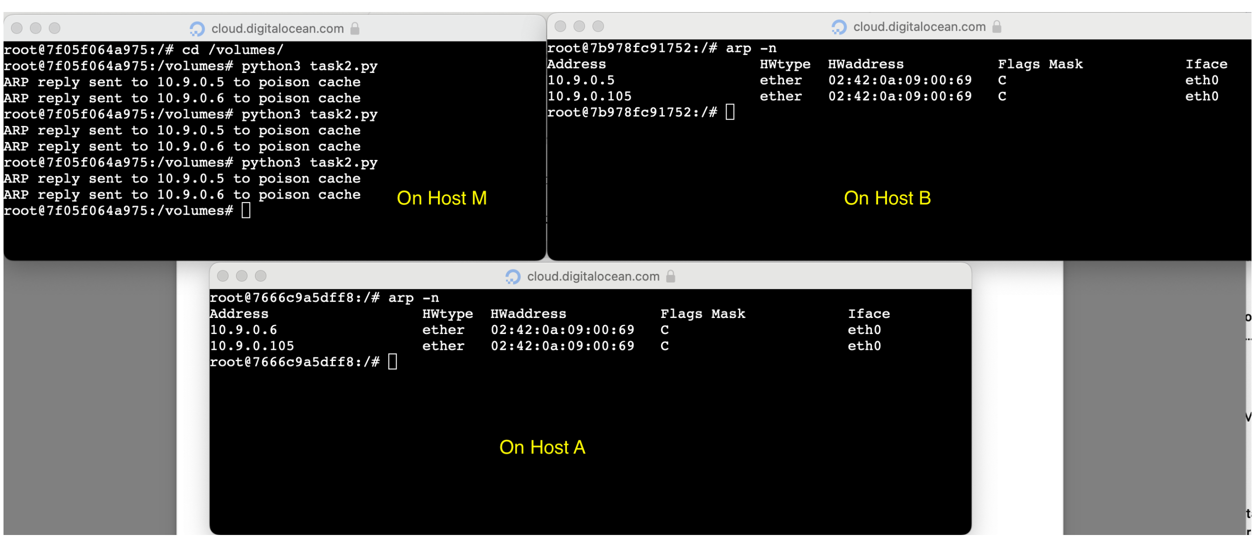
2. ARP Poisoning Function (`arp_poison`):

- The `arp_poison` function takes four parameters: `target_ip` (IP of the target host), `target_mac` (MAC address of the target host), `host_ip` (IP of the host sending the ARP reply), and `host_mac` (MAC address of the host sending the ARP reply)
- Inside the function, an ARP reply packet (`arp_reply`) is constructed using the `ARP` function with the specified parameters (`pdst`, `hwdst`, `psrc`, `hwsrc`, `op`).
- The ARP reply packet is then sent using the `send` function with `verbose=False` to avoid printing verbose output. A message indicating the ARP reply sent is printed for each target IP.

3. Defining Host and Target Information:

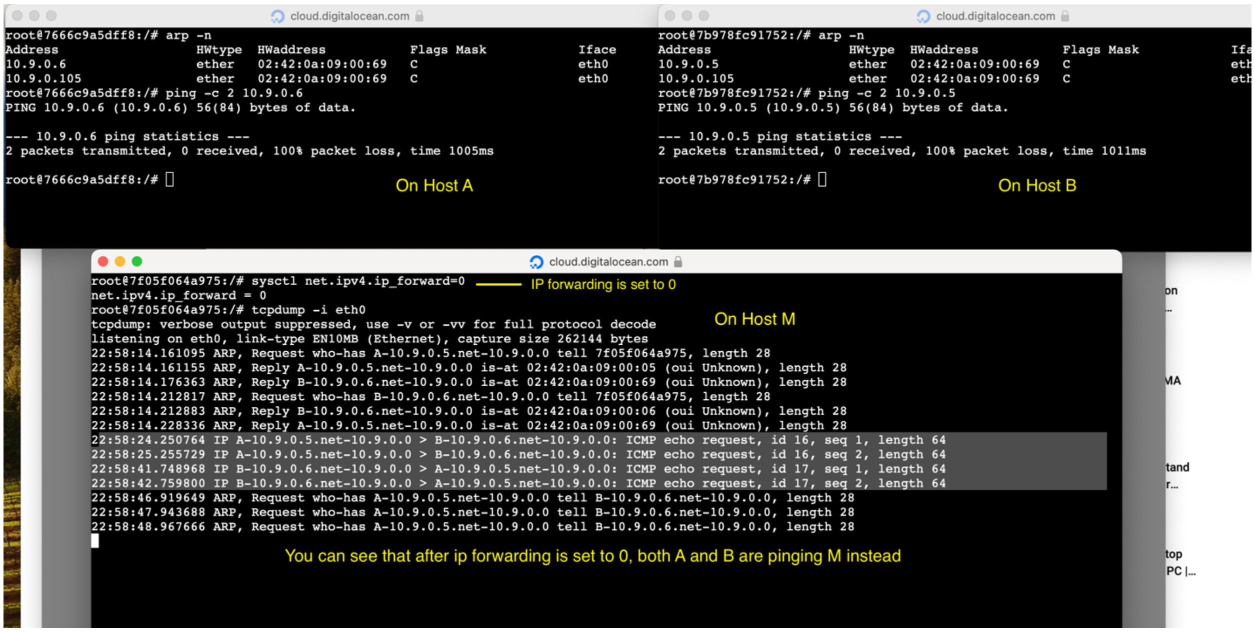
- Host M's IP address (`host_m_ip`) and MAC address (`host_m_mac`) are defined.
- Host A's IP address (`host_a_ip`) and MAC address (`host_a_mac`) are defined.
- Host B's IP address (`host_b_ip`) and MAC address (`host_b_mac`) are defined.

4. ARP Cache Poisoning Calls: The `arp_poison` function is called twice: first to poison Host A's ARP cache by sending an ARP reply from Host B's IP to Host A's IP with Host M's MAC address, and then to poison Host B's ARP cache similarly but in reverse.



Task 2.2: Testing

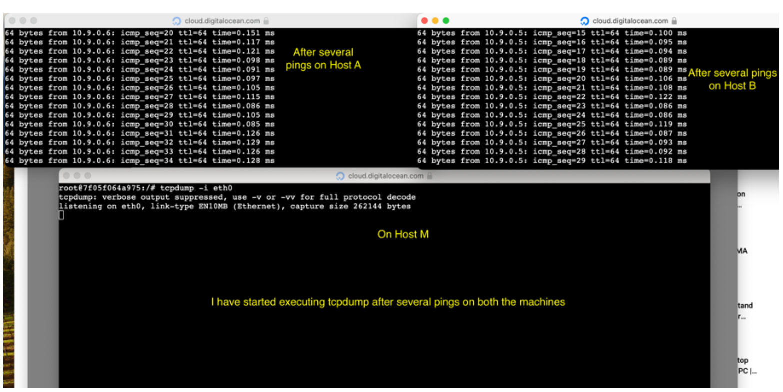
After the ARP poisoning was successful, the communication between Hosts A and B was monitored to confirm that the data was indeed passing through Host M. This was essential to ensure that the subsequent steps could be performed with Host M effectively in the middle of the communication channel.



Note: I have used tcpdump instead of wireshark to capture/track packet communication between A and B on Host M as I had problem with VNC Viewer.

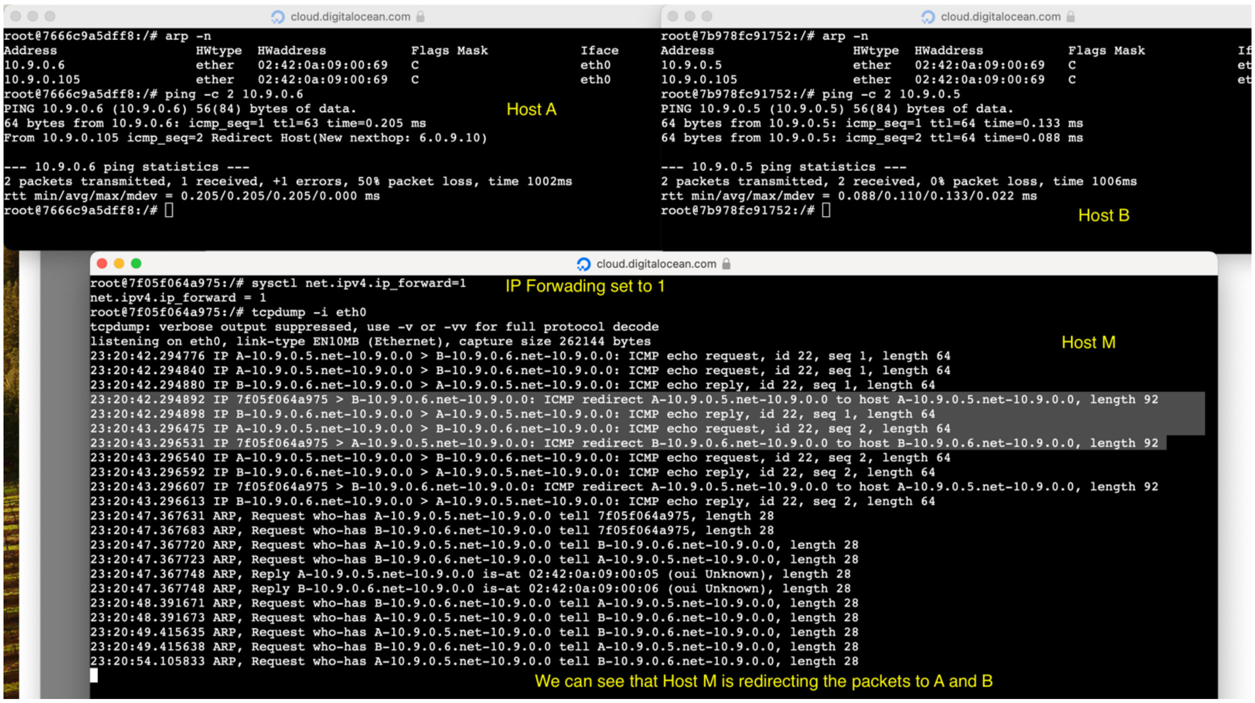
When both A and B ping each other, they both ping M instead. These packets can be captured on M through TCPdump as shown in the screenshot above.M does not respond to the pings. After around several pings, the correct MAC addresses are restored for A and B, and they start to ping each other instead.

After several pings:



Task 2.3: Turn on IP Forwarding

To allow Host M to forward packets between Hosts A and B, IP forwarding was enabled on Host M. This allowed Host M to receive packets from one host and send them to the other, maintaining the illusion that they were communicating directly with each other.



Note: I have used tcpdump instead of wireshark to capture/track packet communication between A and B on Host M as I had problem with VNC Viewer.

Based on the packets captured on M through Wireshark, the pings are forwarded by M to their respective machines. A and B are unaware of the redirection and do not restore the MAC address cached to the correct ones.

Task 2.4: Launch the MITM Attack

With Host M positioned in the middle of the Telnet session, the MITM attack was launched. Host M intercepted the Telnet packets, read their contents, and had the ability to modify the data before forwarding it on to the intended recipient. This

demonstrated the dangerous potential of ARP cache poisoning in compromising the integrity of data on a network.

Code snippet:

```
cloud.digitalocean.com
#!/usr/bin/env python3
from scapy.all import *

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 IP in pkt and TCP in pkt:
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
            # Create a new packet based on the captured one.
            # 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.
            # 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
                newdata = b'Z' * len(data)    # Replace each character with 'Z'
                changed_chars = [f"changed from {c} to Z" for c in data.decode()]
                print(", ".join(changed_chars))
                send(newpkt/newdata, verbose=False)  # Suppress send message
            else:
                send(newpkt, verbose=False)  # Suppress send message
        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, verbose=False)  # Suppress send message

f = 'tcp'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)

~
~
~
"MIM.py" 42L, 1702B                                    42,0-1        All
```

Code Explanation:

1. Importing Scapy: The script imports necessary functions and modules from Scapy.

2. Defining IP and MAC Addresses: It defines the IP and MAC addresses for Hosts A and B.

3. Spoofing Function (`spoof_pkt`):

The `spoof_pkt` function is defined to handle captured packets (`pkt`).

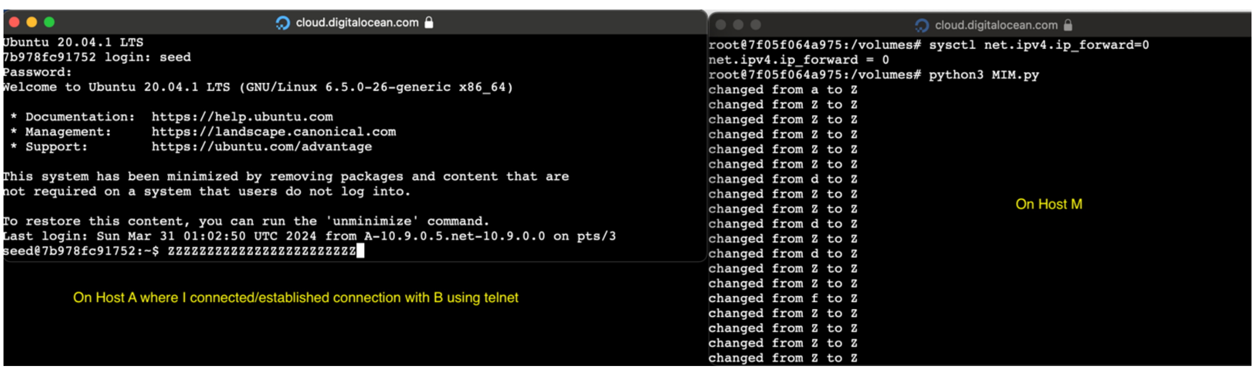
- It first checks if the packet contains IP and TCP layers.
- If the packet is from Host A to Host B, it creates a new packet (`newpkt`) based on the captured one but with modifications.
- It deletes the checksums in the IP and TCP headers to allow Scapy to recalculate them.
- If the TCP payload exists, it replaces each character in the payload with 'Z' and prints the changes made.
- It then sends the modified packet to the destination host (Host B).
- If the packet is from Host B to Host A, it creates a new packet (`newpkt`) based on the captured one without any modifications and sends it to Host A.

4. Packet Sniffing and Processing:

- The script uses the `sniff` function to capture TCP packets on the 'eth0' interface (`iface='eth0'`) and filters only TCP packets (`filter='tcp'`).

- The `spoof_pkt` function is called for each captured packet (`prn=spoof_pkt`).

Successful Attack execution screenshot:



From the screenshot above we can see that on telnet, when sending packet from host A to host B, it's changing every character to 'Z.'

Task 3: Man-In-The-Middle (MITM) Attack on Netcat Using ARP Cache Poisoning

Task 3's objective was to execute a MITM attack on a Netcat session between Hosts A and B, analogous to Task 2, but utilizing Netcat as the communication medium instead of Telnet. The goal was for Host M to intercept and alter the data being transferred between the two hosts.

Code:

```
from scapy.all import *
import re

# Update these variables with the correct IP and MAC addresses for your lab
VM_A_IP = '10.9.0.5'  # IP address of Host A
VM_B_IP = '10.9.0.6'  # IP address of Host B
VM_A_MAC = '02:42:0a:09:00:05'  # MAC address of Host A
VM_B_MAC = '02:42:0a:09:00:06'  # MAC address of Host B

def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP and pkt[TCP].payload:
        payload = pkt[TCP].payload.load
        print("* %s, length: %d" % (payload.decode('utf-8'), len(payload)))

        # Replace 'Kishan' with 'AAAAAA' in the payload
        new_payload = payload.replace(b'kishan', b'AAAAAA')

        print("* Modified payload: %s, length: %d" % (new_payload.decode('utf-8'), len(new_payload)))

        # Create a new packet with the modified payload
        new_pkt = IP(src=pkt[IP].src, dst=pkt[IP].dst) / \
                  TCP(sport=pkt[TCP].sport, dport=pkt[TCP].dport, seq=pkt[TCP].seq, ack=pkt[TCP].ack, flags=pkt[T
CP].flags) / \
                  new_payload

        # Remove checksums
        del new_pkt[IP].chksum
        del new_pkt[TCP].chksum

        send(new_pkt, verbose=False)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        print("Packet received from B to A; forwarding without modification.")
        send(pkt, verbose=False)

# Start sniffing for TCP packets
sniff(filter='tcp', prn=spoof_pkt)

~
~
~
~
~
~
~
~
"netcat-MIM.py" 36L, 1394B                                                    36,0-1        All
```

Code Explanation:

1. Importing Scapy and re Module:

- The script imports necessary functions and modules from Scapy.

- It also imports the re module for regular expression operations, although it's not currently used in the script.

2. Defining IP and MAC Addresses:

- It defines the IP and MAC addresses for Hosts A and B.

3. Spoofing Function (`spoof_pkt`):

- The `spoof_pkt` function is defined to handle captured packets (`pkt`).

- It checks if the packet is from Host A to Host B (`pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP`) and if the packet has a TCP payload (`pkt[TCP].payload`).
- If the conditions are met, it extracts the payload, prints its content and length, and replaces 'Kishan' with 'AAAAAA' in the payload.
- It then creates a new packet (`new_pkt`) with the modified payload and sends it to the destination host (Host B).
- If the packet is from Host B to Host A (`pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP`), it forwards the packet without modification.

4. Packet Sniffing and Processing:

- The script uses the `sniff` function to capture TCP packets (`filter='tcp'`) and calls the `spoof_pkt` function for each captured packet (`prn=spoof_pkt`).

Procedure and Execution:

Netcat Communication Setup:

Hosts A and B initiated a Netcat session to emulate typical client-server interaction. Host A acted as the client sending messages, while Host B served as the listening server.

ARP Spoofing:

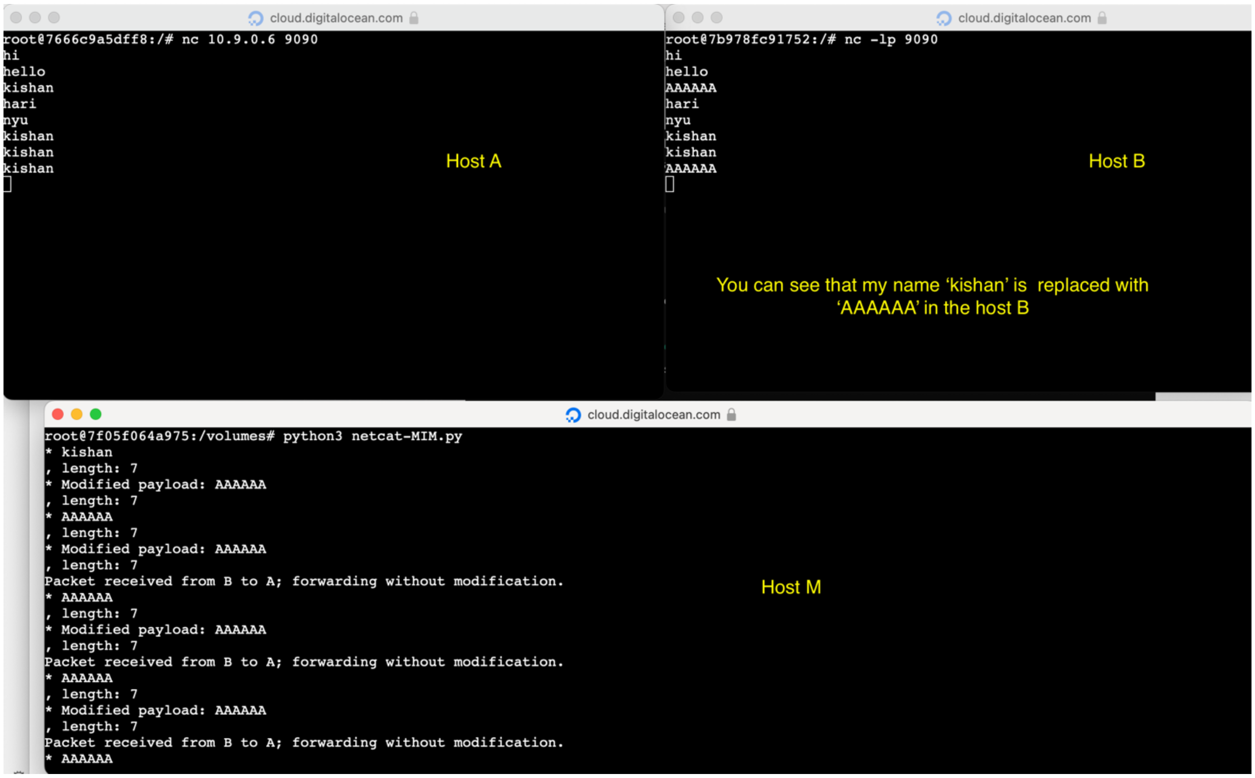
Similar to the previous task, Host M performed ARP cache poisoning to trick both Hosts A and B into thinking that it was the other host in the communication channel. This redirection funneled their Netcat session traffic through Host M.

Packet Sniffing and Modification:

After successfully rerouting the traffic, Host M utilized a Scapy script to intercept TCP packets and search for specific strings for modification. The script's objective was to capture text within the Netcat session, make alterations, and then forward the modified packets to their intended destination.

Verification:

The effectiveness of the interception was validated by transmitting specific strings from Host A and verifying if these strings were received in their modified state on Host B. Additionally, network monitoring tools were utilized to observe the flow of traffic and confirm that Host M was indeed positioned in the middle of the communication pathway.



We can see that my name ‘kishan’ is replaced by ‘AAAAAA’

Note: I reason why Kishan sometimes isn’t replaced is that on Host A, arp cache tables are automatically changed and Ip address of B is mapped to MAC B based on traffic flow. So, I have to again run the script/attack continuously to spoof and make changes to TCP payload.

Task 4: Write-Up on ARP Attacks

In this lab, we explored ARP (Address Resolution Protocol) attacks, focusing on ARP cache poisoning and MITM (Man-in-the-Middle) attacks. Here is an overview of what I learned, along with interesting observations, surprising findings, and challenges faced during the lab.

Key learnings and observations:

1. ***ARP Cache Poisoning:*** This technique involves sending falsified ARP messages to devices on the network, tricking them into associating the attacker's MAC address with a legitimate IP address. This can lead to traffic redirection and interception.
2. ***MITM Attacks:*** ARP cache poisoning is often used in MITM attacks, where an attacker positions themselves between two communicating parties, intercepting, and potentially altering the data exchanged between them.

3. ***Successful ARP Spoofing:*** One interesting observation was how easily ARP cache poisoning could be successful, especially when the target host's ARP cache was not properly secured. It demonstrated the vulnerability of ARP to manipulation.
4. ***Effectiveness of ARP Gratuitous Messages:*** Gratuitous ARP messages proved to be effective in updating ARP caches even when an entry for the target IP address already existed. This was surprising as it showcased the potential for rapid network disruption using ARP attacks.
5. ***Impact on Network Traffic:*** The MITM attacks conducted using ARP cache poisoning significantly impacted network traffic flow. It highlighted the importance of securing ARP tables to prevent unauthorized network access and data interception.

Scenarios and Observations:

Scenario 1: ARP Cache Poisoning using ARP Request (Task 1.A)

Observation: The attack was successful, and the ARP entries of Host A were modified/spoofed to map IP B to MAC M.
Explanation: We crafted an ARP request packet to trick Host A into associating Host M's MAC address with Host B's IP address, effectively redirecting traffic meant for Host B to Host M.

Scenario 2: ARP Cache Poisoning using ARP Reply (Task 1.B)

Observation: When Host B's IP was already present in Host A's ARP cache, the spoofing was immediately effective, and the traffic was rerouted through Host M.
Explanation: Sending ARP reply packets deceived Host A into associating Host B's IP address with the Attacker's MAC address, leading to successful ARP cache poisoning.

Scenario 3: ARP Cache Poisoning using ARP Gratuitous Message (Task 1.C)

Observation: The gratuitous ARP packet successfully updated Host A's ARP cache, even when an entry for Host B already existed, demonstrating its potential for network disruption.
Explanation: The ARP gratuitous message forced an update in Host A's ARP cache, manipulating the ARP entries and potentially redirecting network traffic.

Telnet MITM Attack (Task 2)

Observation: The ARP cache poisoning successfully rerouted Telnet traffic between Hosts A and B through Host M, showcasing the vulnerability of unprotected network communication.
Explanation: ARP cache poisoning in the Telnet scenario demonstrated how attackers can leverage ARP attacks to perform MITM attacks on various protocols, compromising network security.

Netcat MITM Attack (Task 3)

Observation: The ARP cache poisoning allowed Host M to intercept and alter the data being transferred between Hosts A and B in a Netcat session.
Explanation: By performing ARP cache poisoning, Host M was positioned as a Man-in-the-Middle, allowing it to capture and modify network traffic between Hosts A and B.

Challenges Faced:

Detection and Mitigation: One of the challenges faced was detecting and mitigating ARP attacks in real-time. Implementing effective intrusion detection systems (IDS) and ARP spoofing detection mechanisms is crucial but can be complex.

Legitimate vs. Malicious ARP Traffic: Distinguishing between legitimate ARP traffic and malicious ARP packets used in attacks required careful analysis and monitoring. This challenge underscores the need for network monitoring tools and security protocols.

ARP Cache Integrity: Maintaining the integrity of ARP caches posed challenges, especially when hosts did not validate ARP updates or relied solely on ARP cache entries without verification.

Conclusion:

ARP attacks pose significant threats to network security, allowing attackers to manipulate network traffic and compromise data integrity. Understanding the mechanisms of ARP cache poisoning, MITM attacks, and the impact on network communication is essential for network administrators and security professionals to implement robust security measures. Effective detection, mitigation strategies, and regular security audits are crucial in safeguarding against ARP-related vulnerabilities and protecting network infrastructure from unauthorized access and data interception.

The lab provided valuable insights into ARP attacks, their impact on network security, and the effectiveness of ARP cache poisoning in facilitating MITM attacks. Understanding these concepts is crucial for network administrators and security professionals to implement robust security measures and protect against ARP-related vulnerabilities in various scenarios.