第二次实验

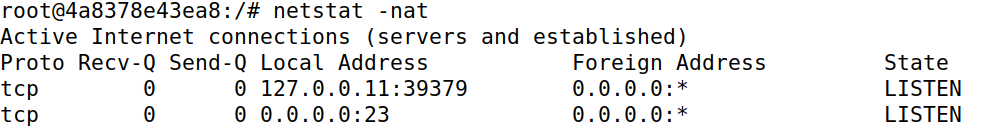
57118122 李泽厚

**Lab 3.1**

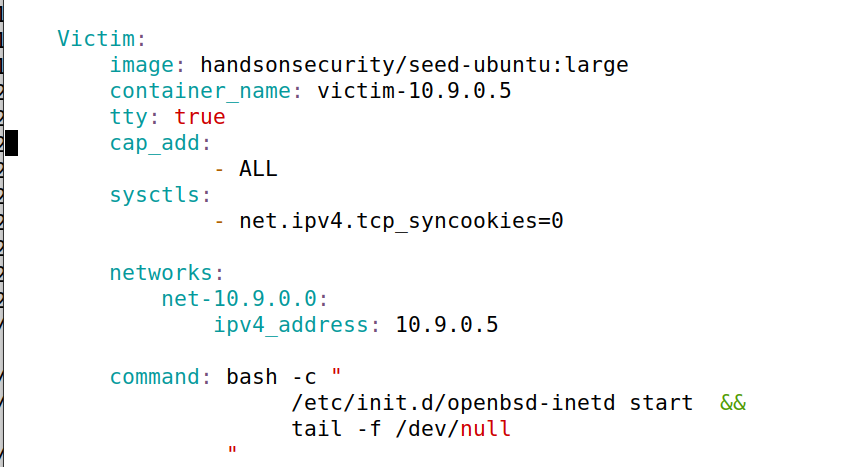
进入受害者主机，输入命令 sysctl -w net.ipv4.tcp\_syncookies=1，改变队列大小。

IMG_256

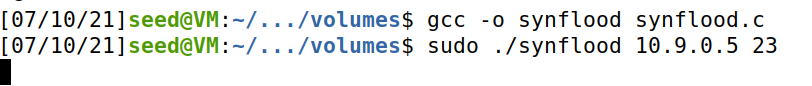
此时在受害者主机上输入命令 nststat -nat，可看见此时队列中没有与其他主机的通信。



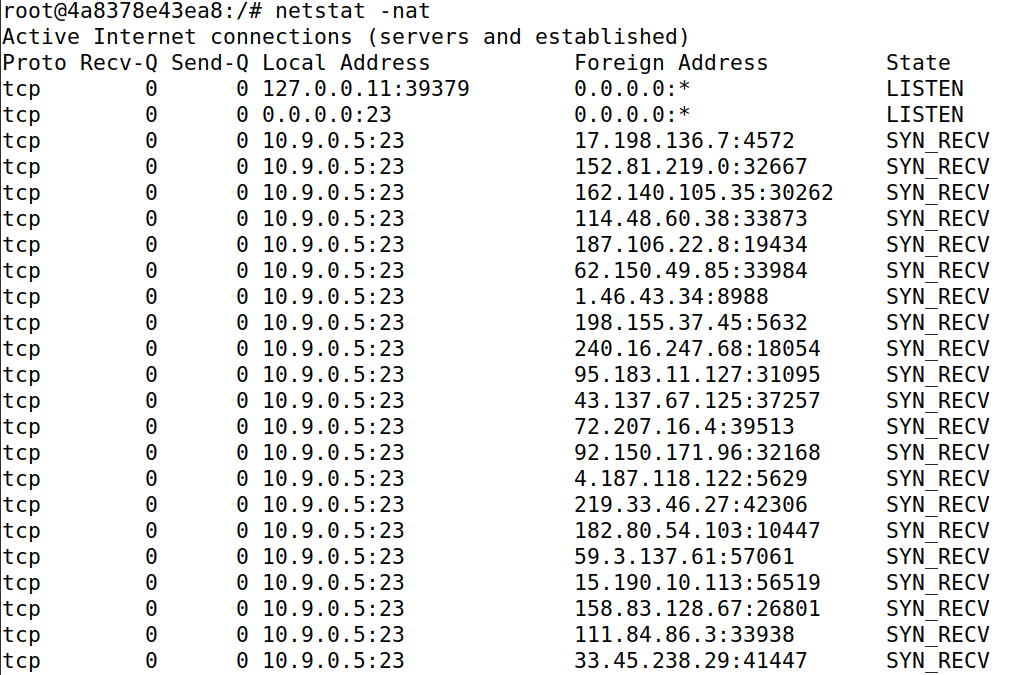
然后回到VM，打开文件 docker-compose.yml ，发现其中受害者主机的SYN cookie已经被禁用。



在VM上编译文件synflood.c 然后输入命令 sudo synflood 10.9.0.5 23。



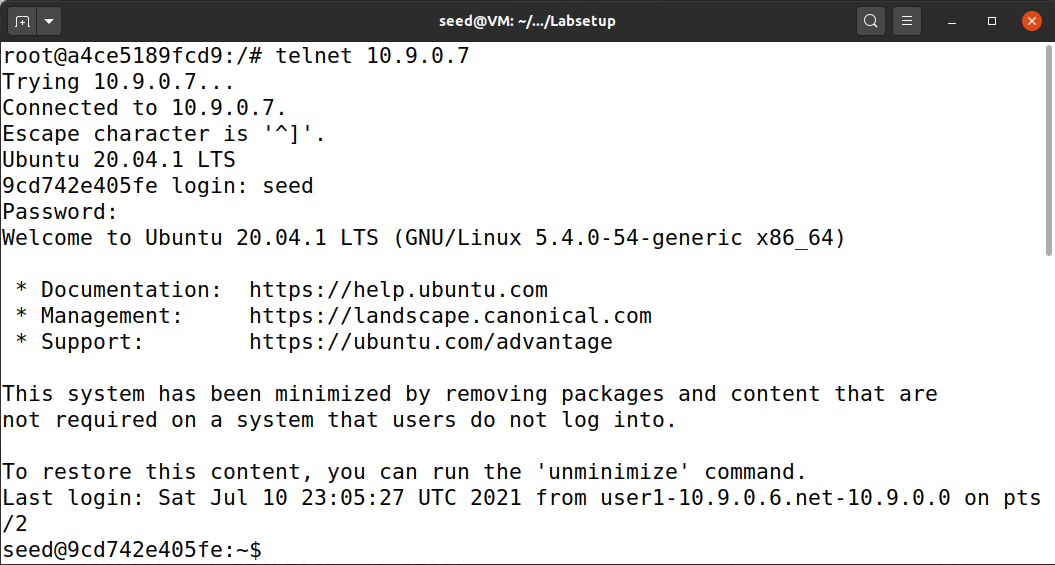
进入受害者主机，先输入命令ip\_tcp metrics flush刷新，再输入 nststat -nat，可看见此时已经受到了SYN泛洪攻击。



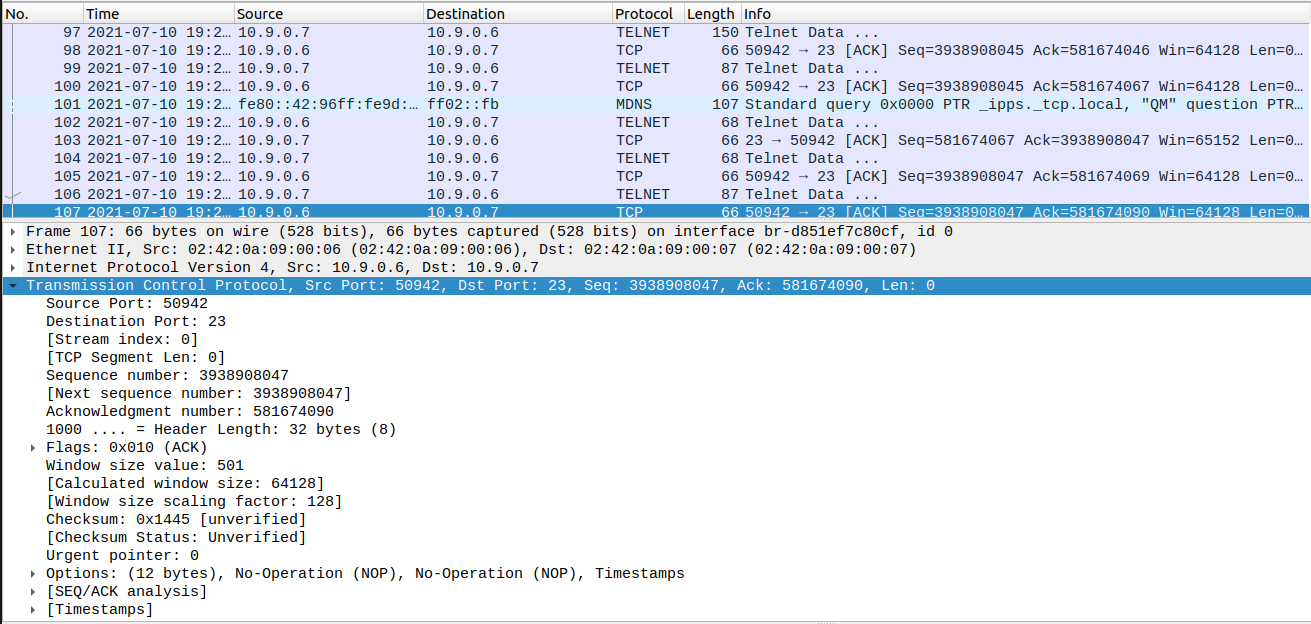
**Lab 3.2**

打开wireshark，开始监听。

在docker中，我们让user1(10.9.0.6)和user2(10.9.0.7)之间进行telnet（由user1发起）。

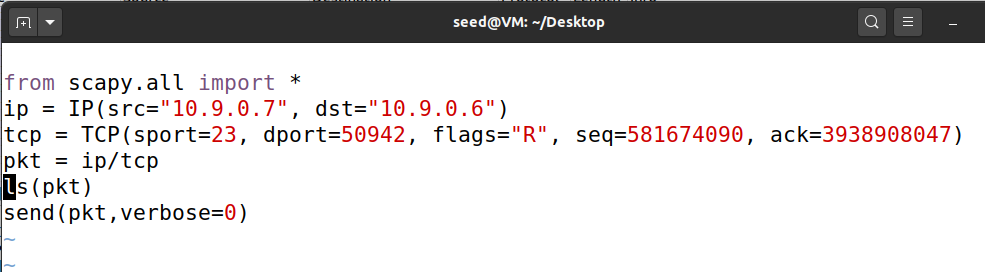


此时用wireshark抓到如下结果：



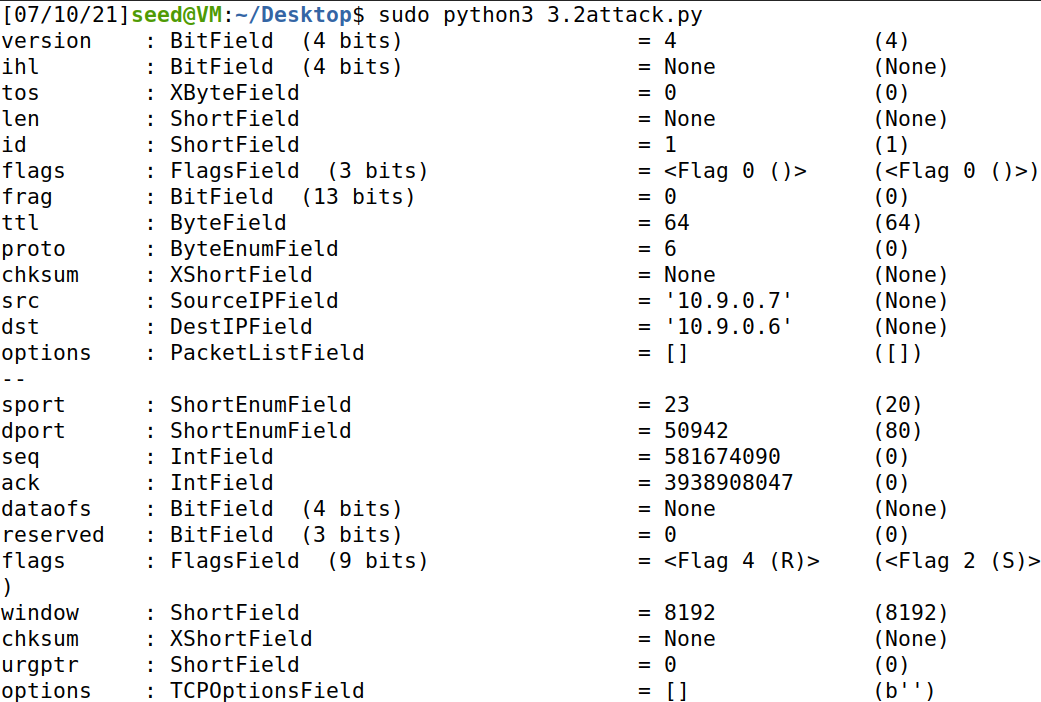
可以看出，最后一次通信是从user1（IP:10.9.0.6，端口号：50942）发给user2（IP:10.9.0.7，端口号：23）。ack=581674090，seq=3938908047，len=0。

在VM上运行如下脚本：



在该脚本中，我们伪造自己的身份为user2，向user1发出结束通信的报文。由于最后一次通信的len=0，故伪造的报文的ack仍为3938908047。

运行该脚本，结果如下：



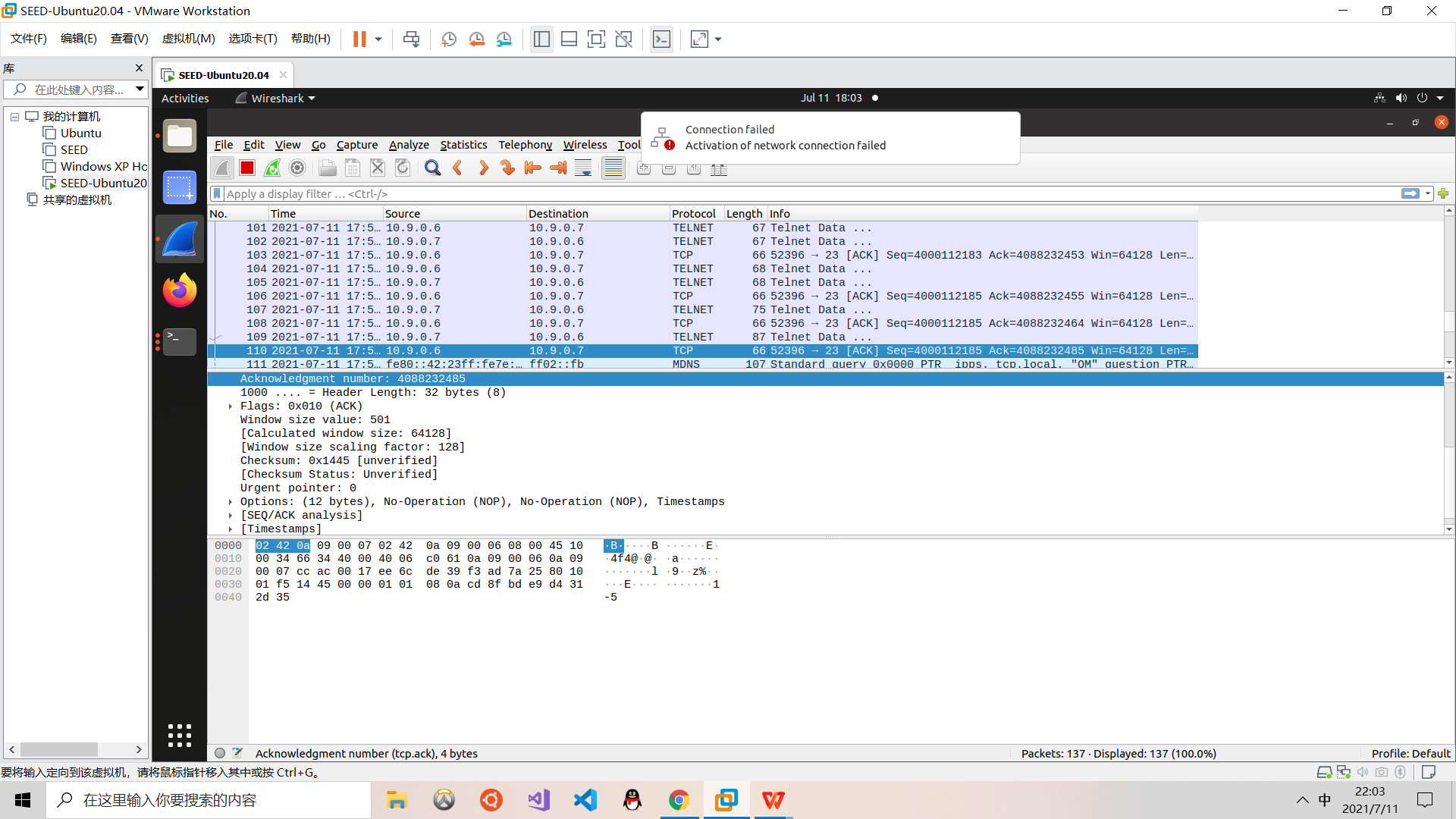
此时查看受害者的终端，显示本次连接已被对方断开：

IMG_256

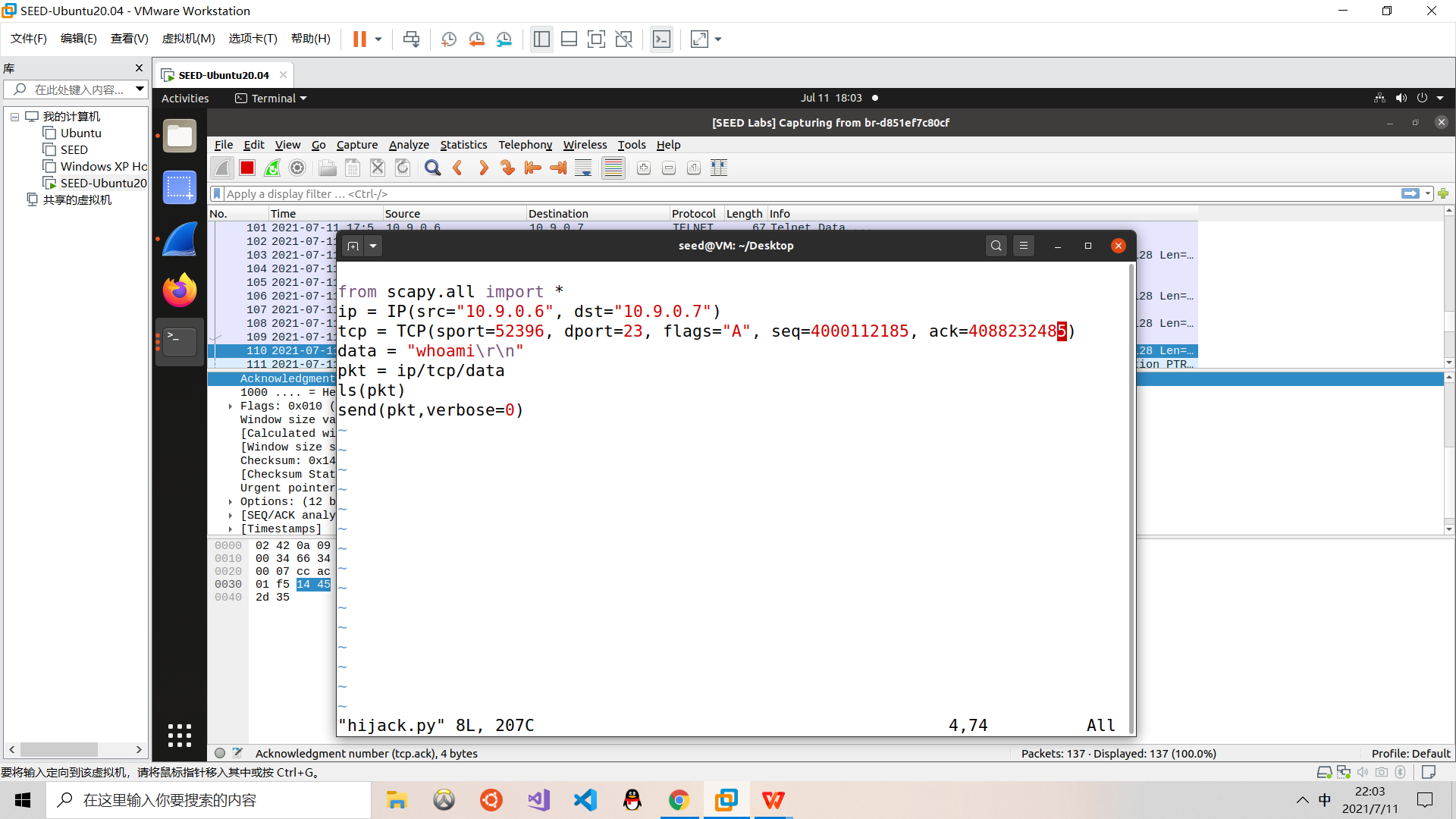
**Lab 3.3**

在docker中，我们让user1(10.9.0.6)和user2(10.9.0.7)之间进行telnet（由user1发起）。

用wireshark抓到如下结果：

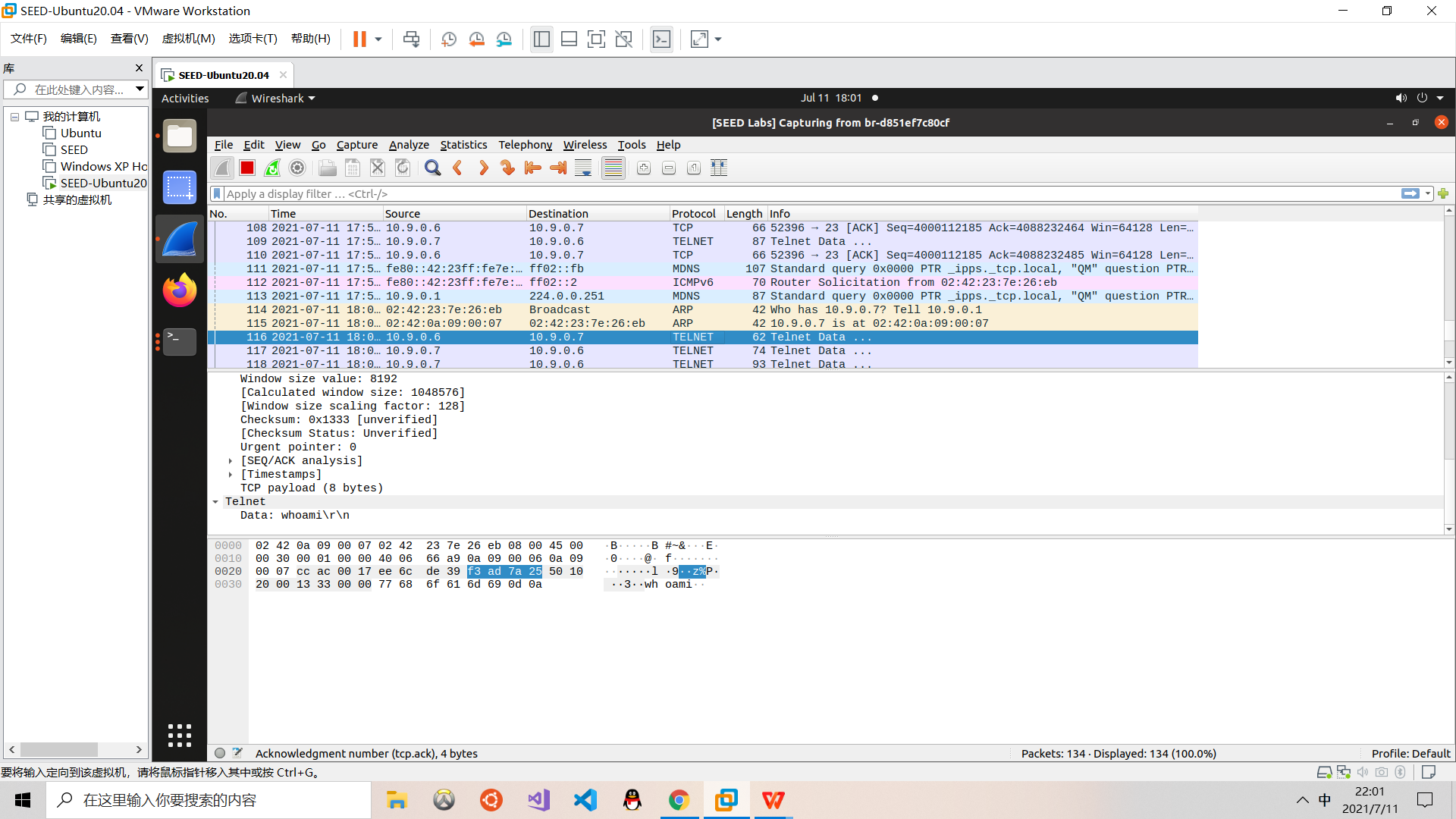


在VM上运行如下脚本：

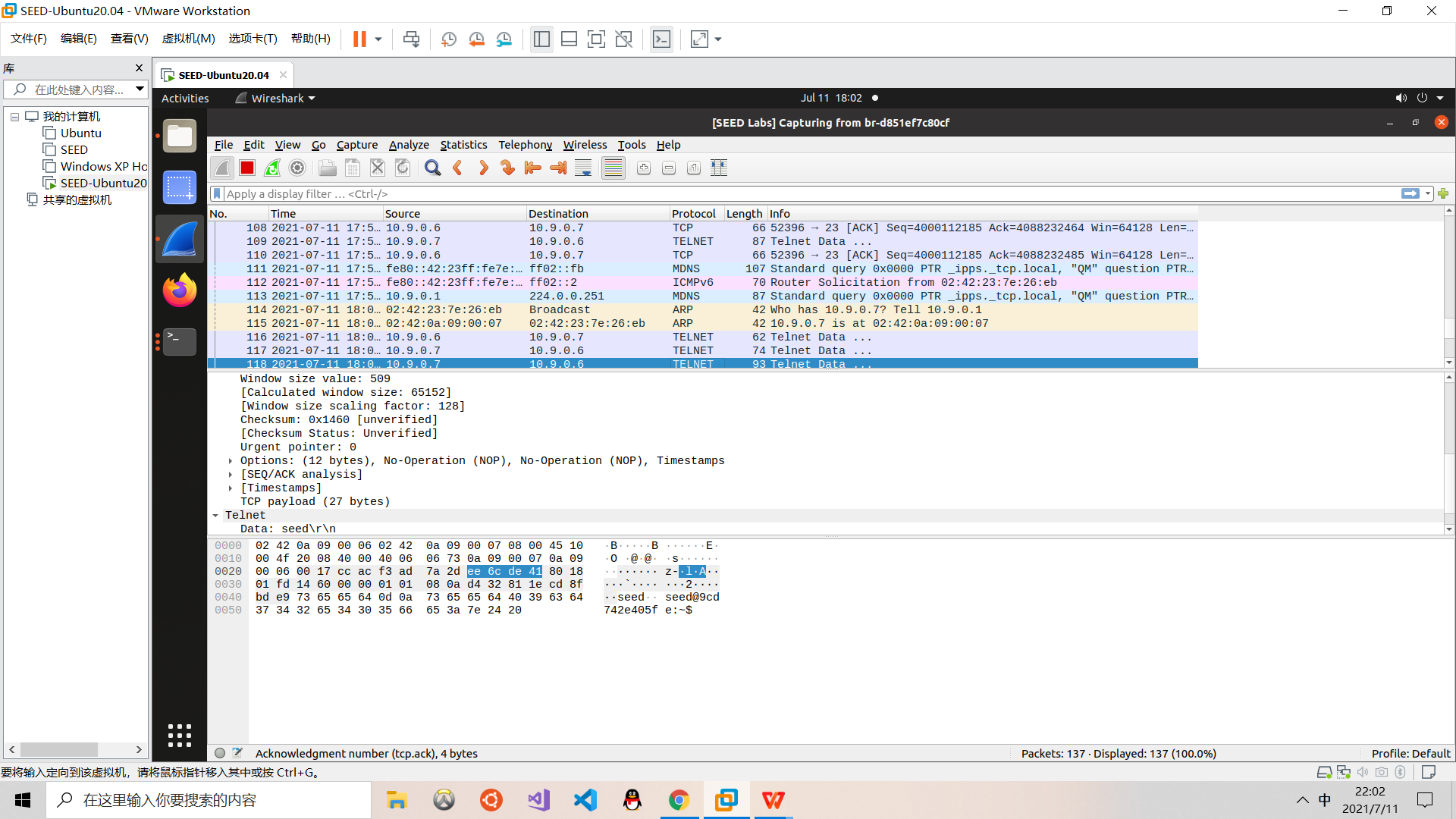


在这个脚本中，我们根据抓取的信息，确定目的和源的端口号和ACK,aeq值，这次劫持对话的过程中，伪造自己的身份为user1(10.9.0.6)，向user2(10.9.0.6)发出“ whoami ”的命令。

结果如下：



可看出我们成功以user1(10.9.0.6)的身份向user2(10.9.0.7)发送了“whami”命令。

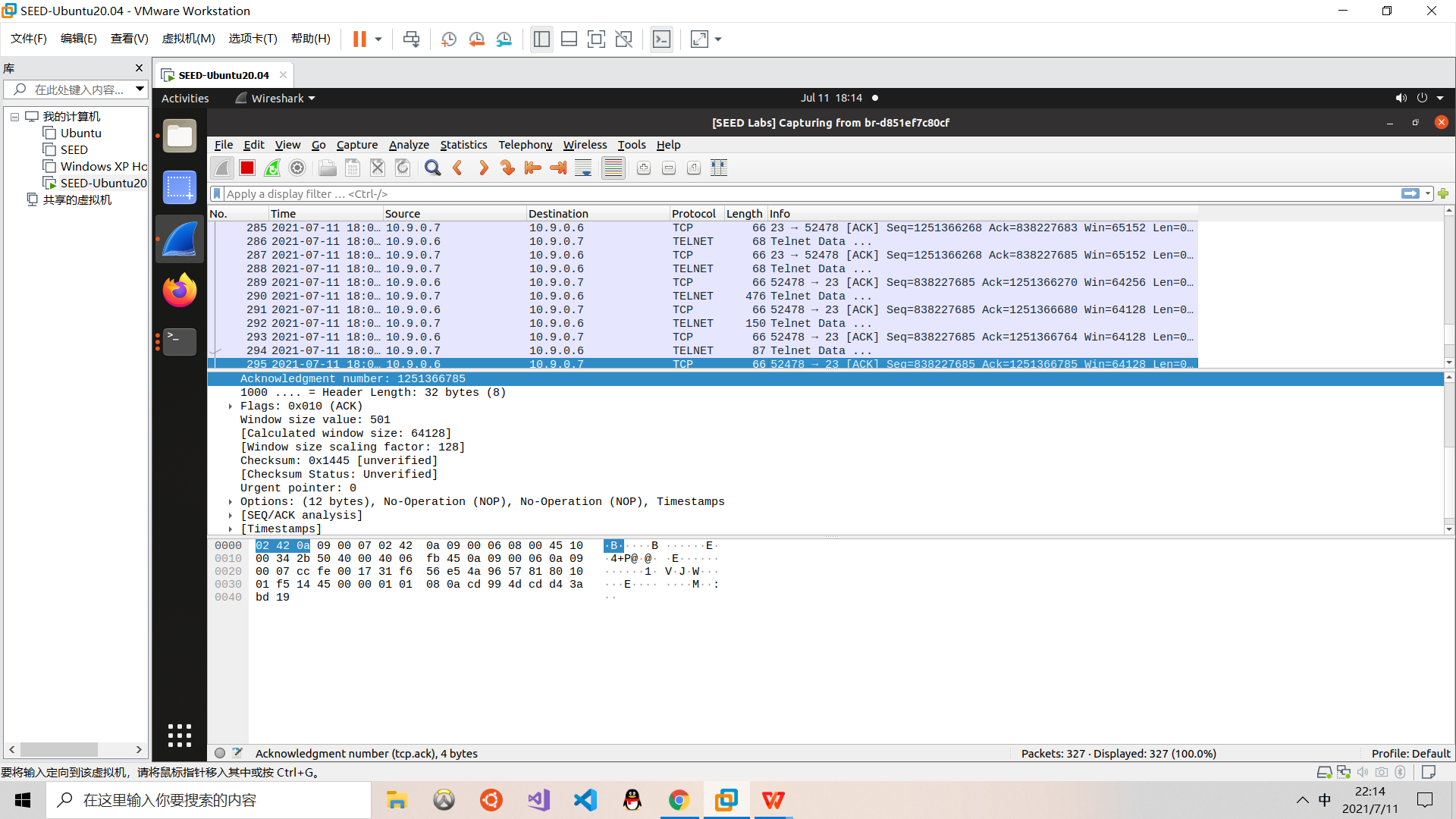


可看出，由于我们之前的“whiami”命令，user2(10.9.0.7)已经向user1(10.9.0.6)发回了“seed”的命令

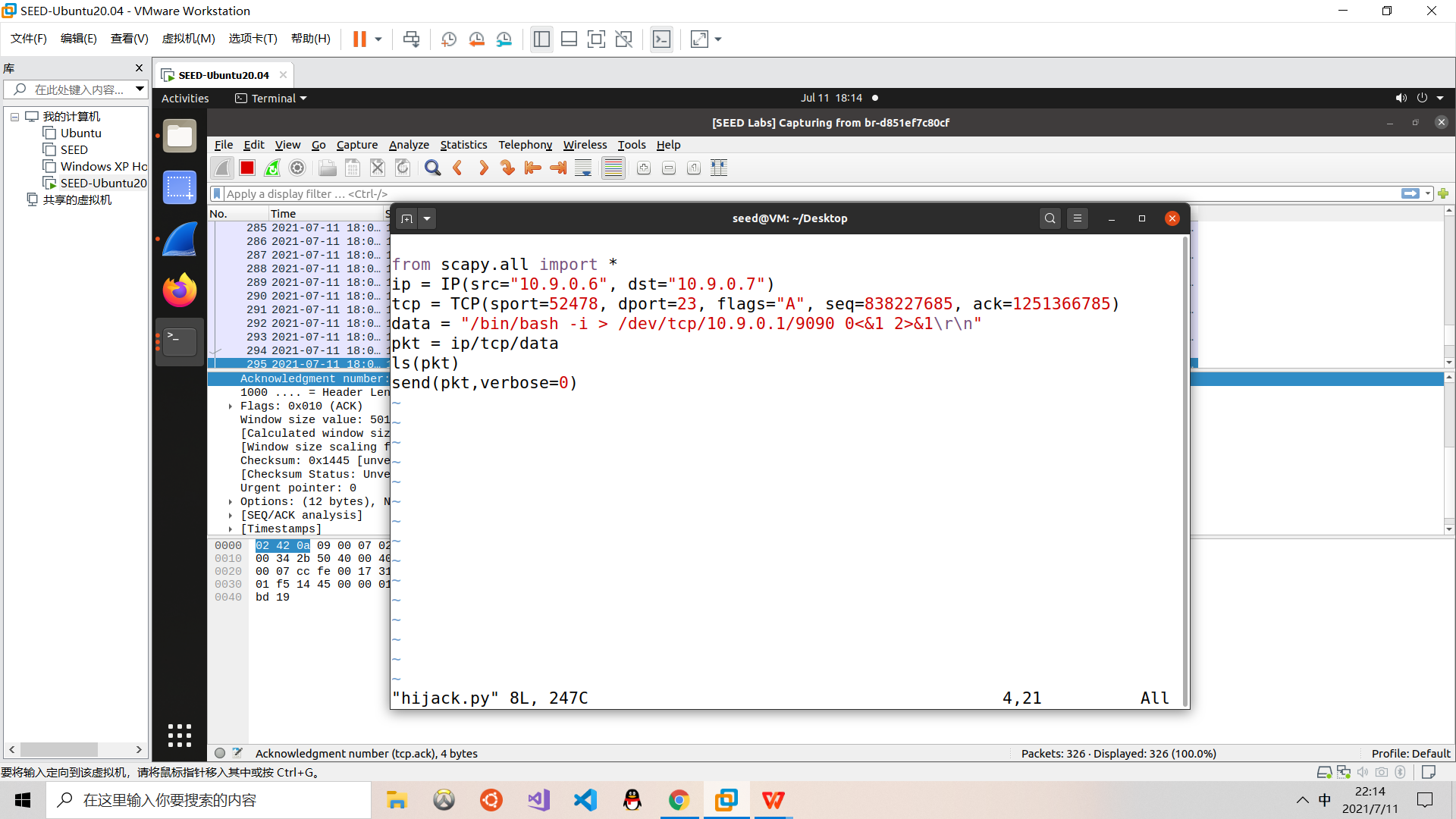
**Lab 3.4**

在docker中，我们让user1(10.9.0.6)和user2(10.9.0.7)之间进行telnet（由user1发起）。

用wireshark抓到如下结果：

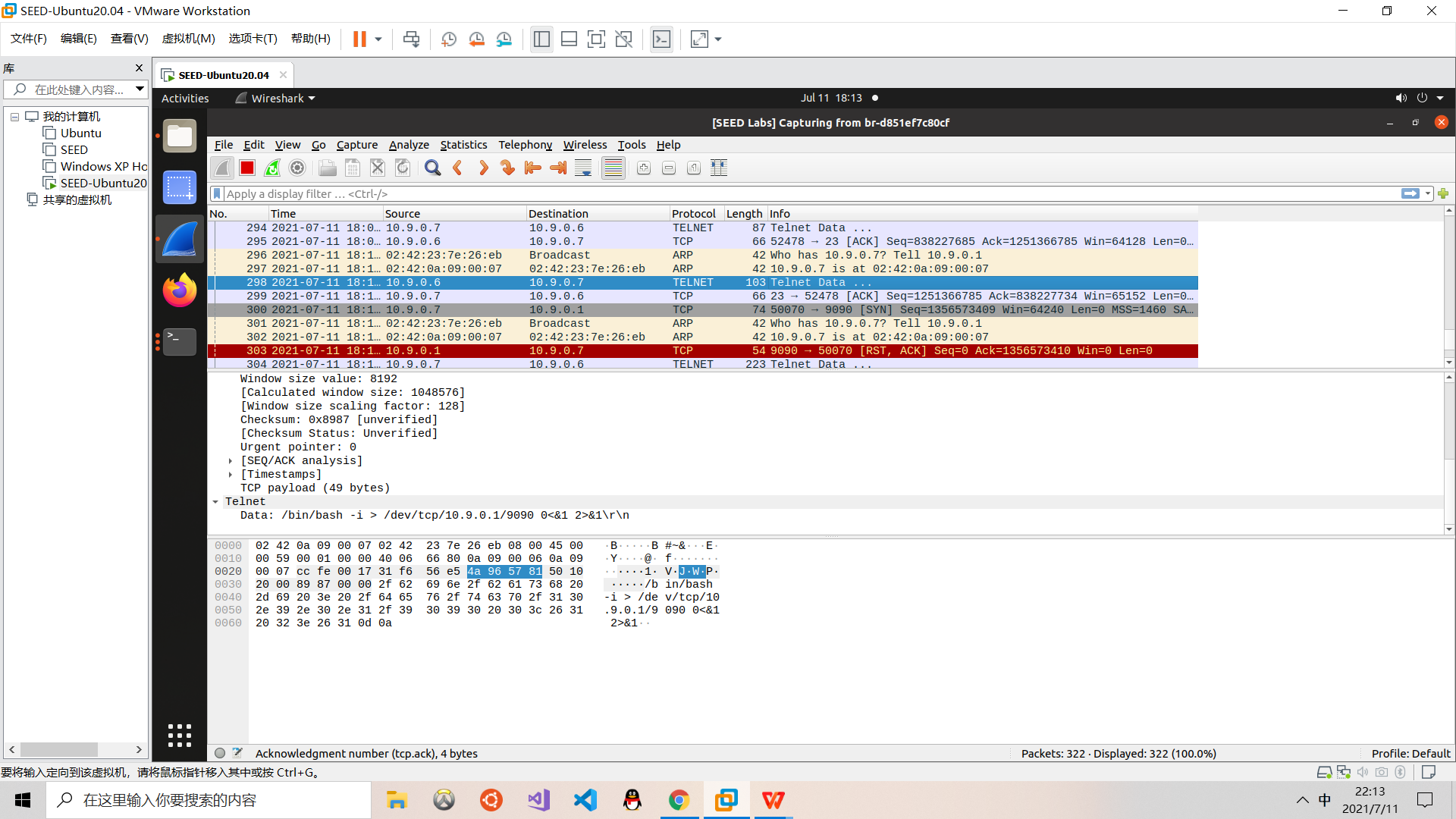


在VM上运行如下脚本：

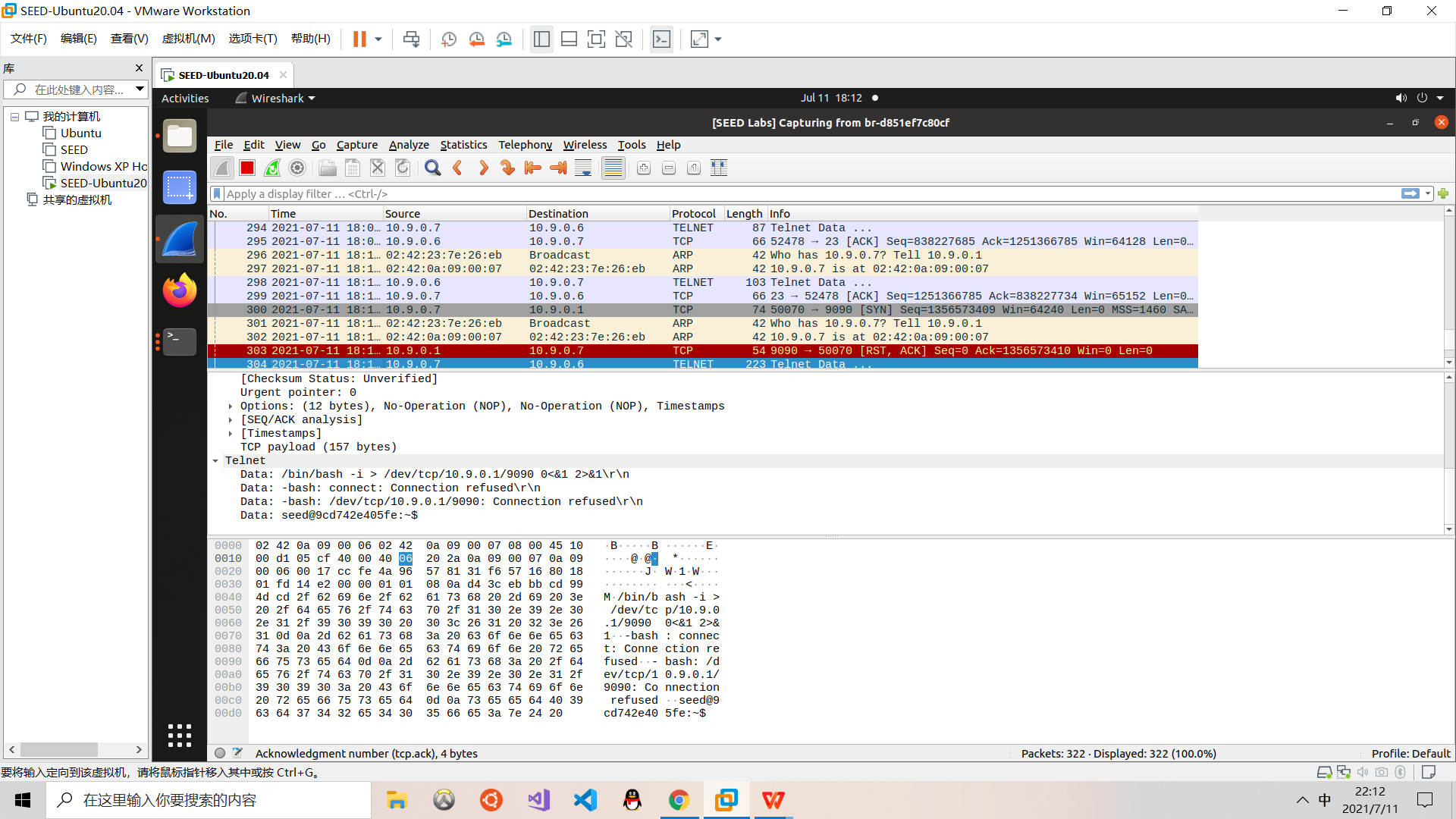


在这个脚本中，我们根据抓取的信息，确定目的和源的端口号和ACK,aeq值，这次劫持对话的过程中，伪造自己的身份为user1(10.9.0.6)，向user2(10.9.0.7)发出反向shell的命令。

结果如下：



可看出我们成功以user1(10.9.0.6)的身份向user2(10.9.0.7)发送了反向shell命令。



可以看出，由于我们之前的反向shell命令，user2(10.9.0.7)已经向user1(10.9.0.6)发回了相应的答复。

在3.3和3.4实验中，由于脚本攻击因为伪造出去的报文会破坏两边序列号的一致性，所以打完之后两边序列号对不上会引起一系列的错误重传，简单表现为终端卡死