Puja Shah

Professor Daeyoung

CSIT-460-03

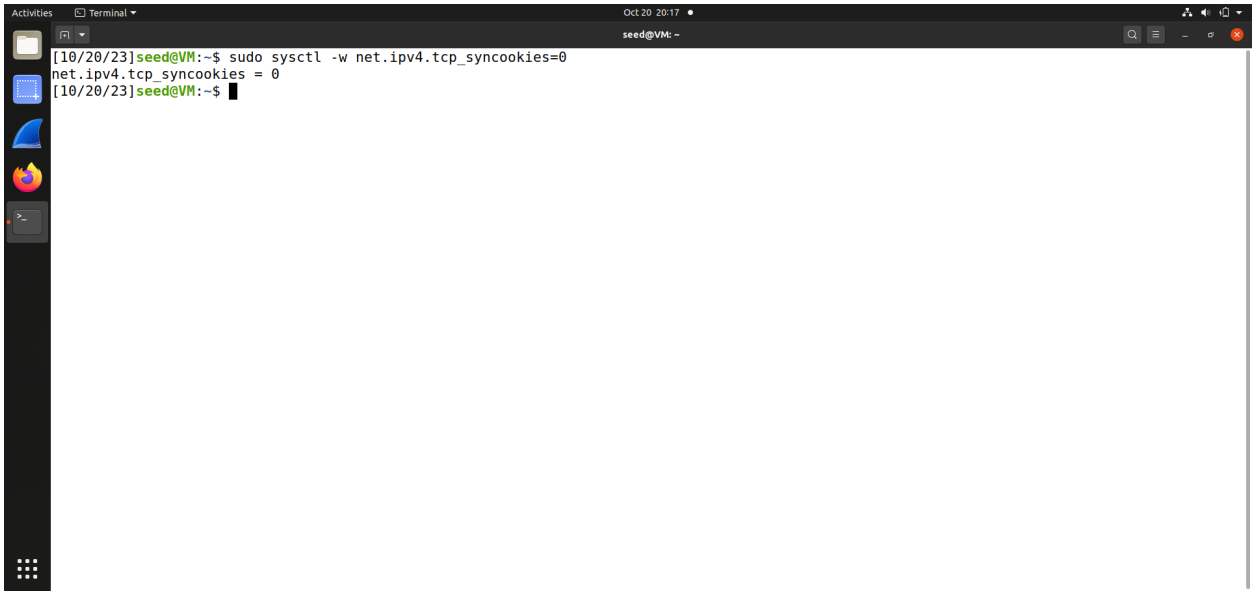
20 October 2023

Lab 2 TCP Attack

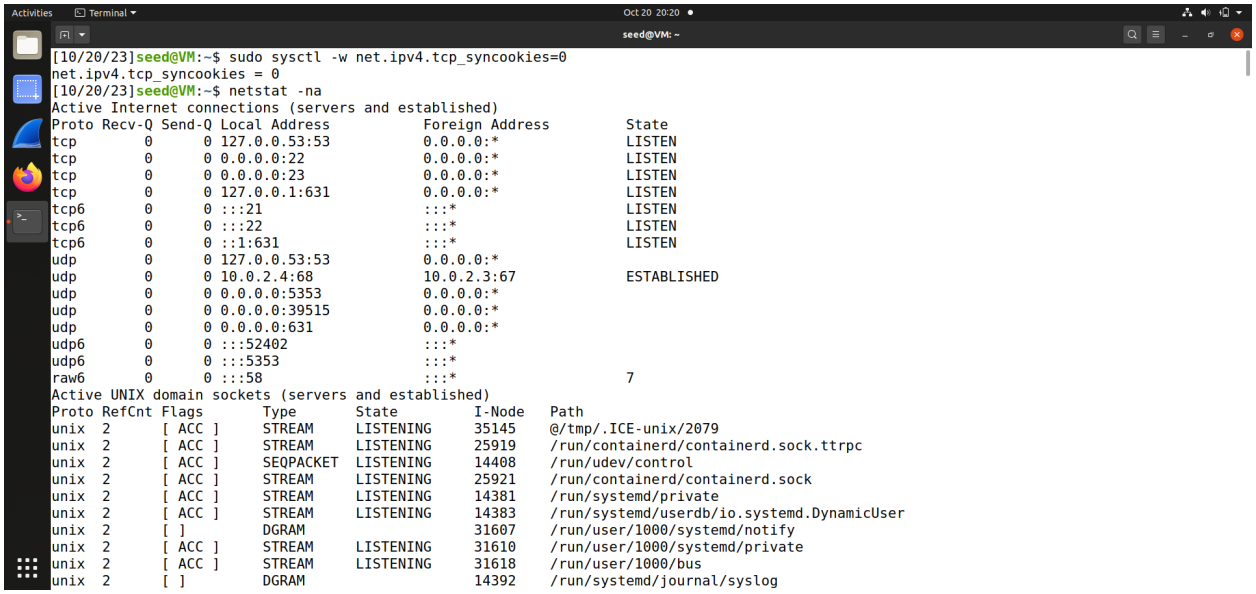
Task 1: SYN Flooding Attack

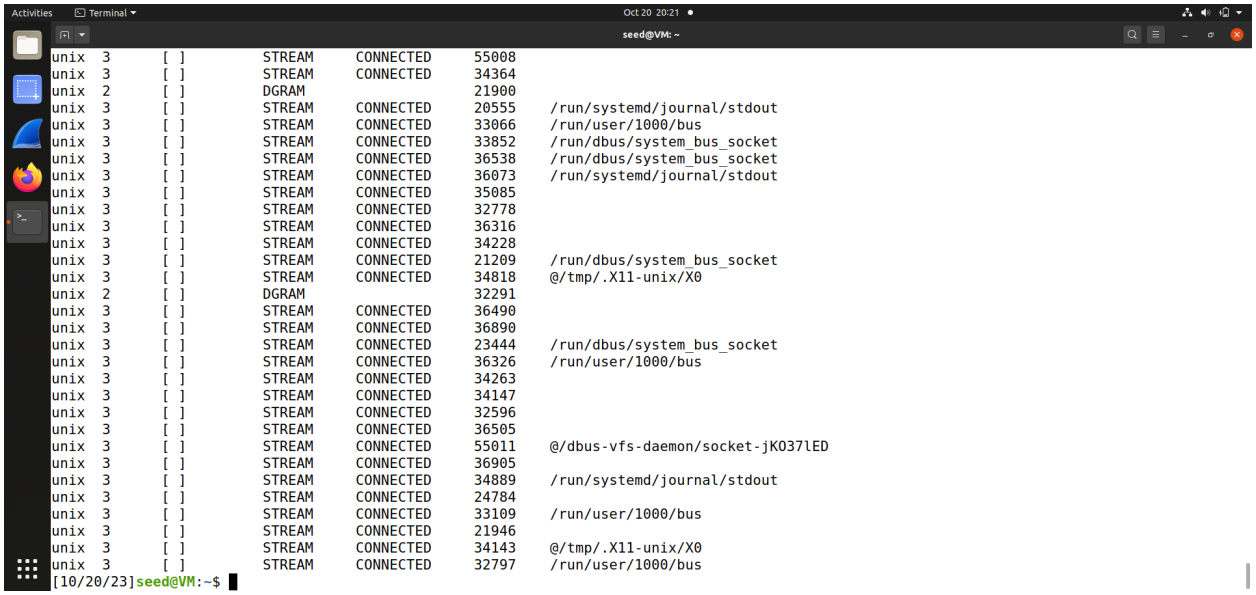
Performed a SYN flooding attack using Netwox and Wireshark on two VMs. Before starting the attack, I ensure that both my victim and attacker VMs are on the same network and install Netwox on both VMs.

To ensure that SYN cookies are turned off, I run the following command on the victim machine:



On the victim VM, I used ‘netstat’ to monitor the number of half-opened connection before the attack:

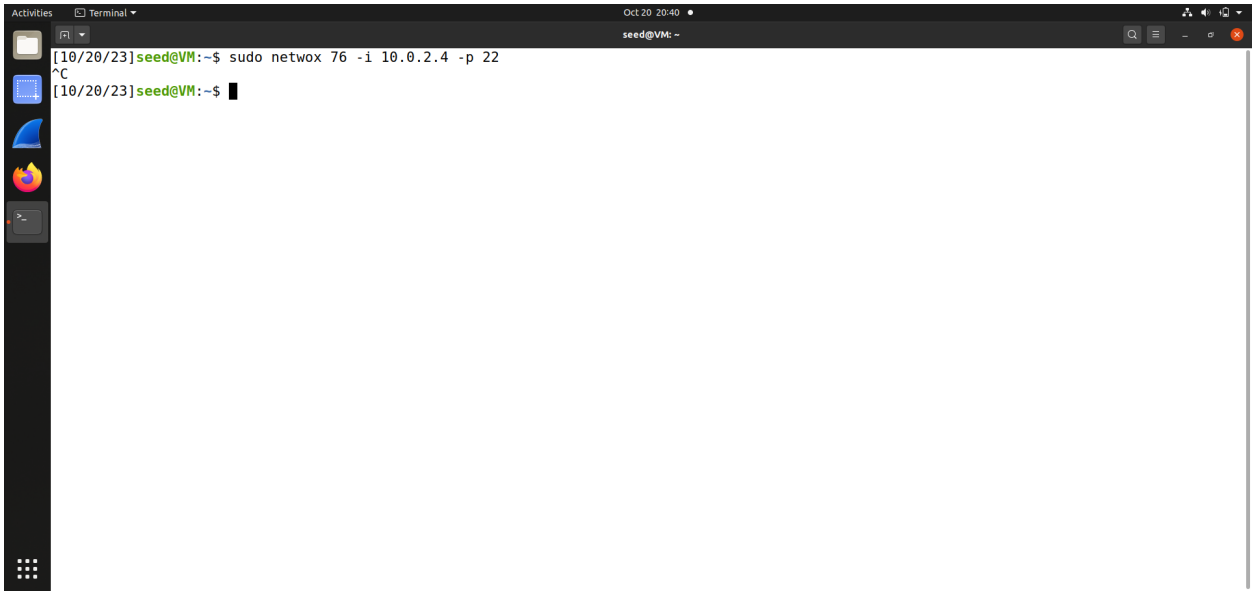




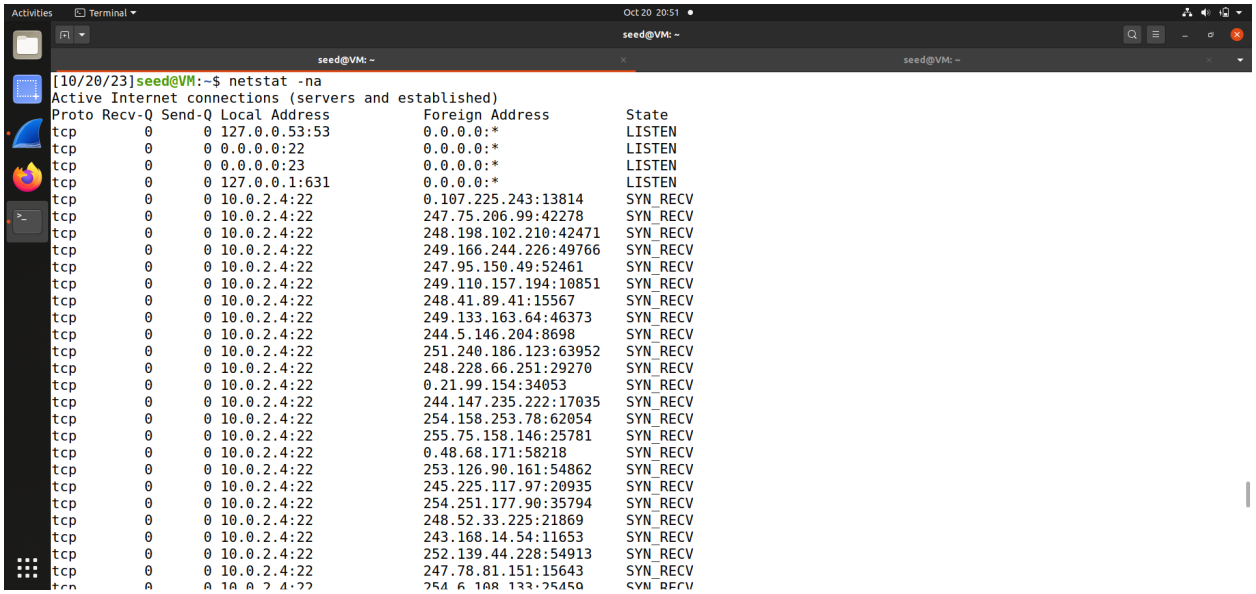
The ‘netstat’ output shows the state of network connections on the victim’s VM before the attack. It doesn’t show any signs of half-opened connections or SYN flooding. Here’s what I observe: SSH service port 22 is in the LISTEN state, indicating that it’s ready to accept incoming connections. Other services like telnet and local DNS service are also in the LISTEN state. There’s an established UDP connection between the victim's VM and other machines on ports 68

and 67, which appears to be related to DHCP or network configuration. There are also some UDP services in the LISTEN state. At this point the victim's VM is in a normal state with services ready to accept connections.

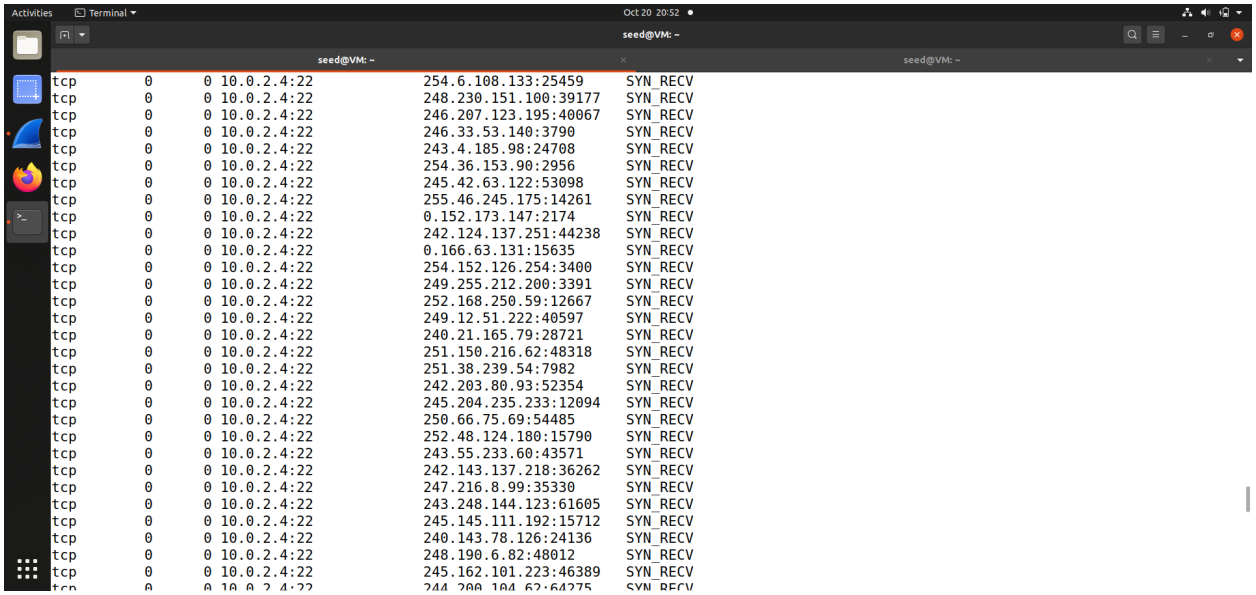
On the attacker VM, I used Netwox tool number 76 to launch the SYN flooding attack. Executed the following command on the attacker's VM:



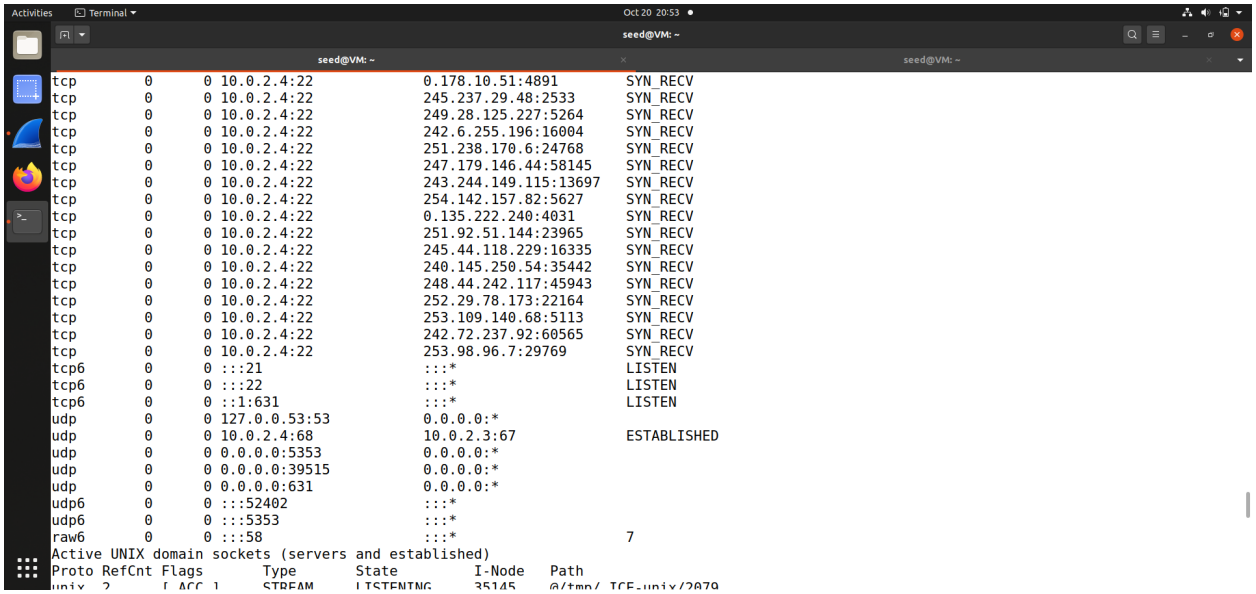
On the victim VM, I re-run the 'netstat' command to monitor the number of half-opened connections while the attack is ongoing:




```
[10/20/23]seed@VM:~$ netstat -na
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.4:22             0.107.225.243:13814     SYN_RECV
tcp        0      0 10.0.2.4:22             247.75.206.99:42278     SYN_RECV
tcp        0      0 10.0.2.4:22             248.198.102.210:42471   SYN_RECV
tcp        0      0 10.0.2.4:22             249.166.244.226:49766   SYN_RECV
tcp        0      0 10.0.2.4:22             247.95.150.49:52461     SYN_RECV
tcp        0      0 10.0.2.4:22             249.110.157.194:10851   SYN_RECV
tcp        0      0 10.0.2.4:22             248.41.89.41:15567      SYN_RECV
tcp        0      0 10.0.2.4:22             249.133.163.64:46373    SYN_RECV
tcp        0      0 10.0.2.4:22             244.5.146.204:8698      SYN_RECV
tcp        0      0 10.0.2.4:22             251.240.186.123:63952   SYN_RECV
tcp        0      0 10.0.2.4:22             248.228.66.251:29270    SYN_RECV
tcp        0      0 10.0.2.4:22             0.21.99.154:34053       SYN_RECV
tcp        0      0 10.0.2.4:22             244.147.235.222:17035   SYN_RECV
tcp        0      0 10.0.2.4:22             254.158.253.78:62054    SYN_RECV
tcp        0      0 10.0.2.4:22             255.75.158.146:25781    SYN_RECV
tcp        0      0 10.0.2.4:22             0.48.68.171:58218       SYN_RECV
tcp        0      0 10.0.2.4:22             253.126.90.161:54862    SYN_RECV
tcp        0      0 10.0.2.4:22             245.225.117.97:20935    SYN_RECV
tcp        0      0 10.0.2.4:22             254.251.177.90:35794    SYN_RECV
tcp        0      0 10.0.2.4:22             248.52.33.225:21869     SYN_RECV
tcp        0      0 10.0.2.4:22             243.168.14.54:11653     SYN_RECV
tcp        0      0 10.0.2.4:22             252.139.44.228:54913    SYN_RECV
tcp        0      0 10.0.2.4:22             247.78.81.151:15643     SYN_RECV
tcp        0      0 10.0.2.4:22             254.6.108.133:25459     SYN_RECV
```

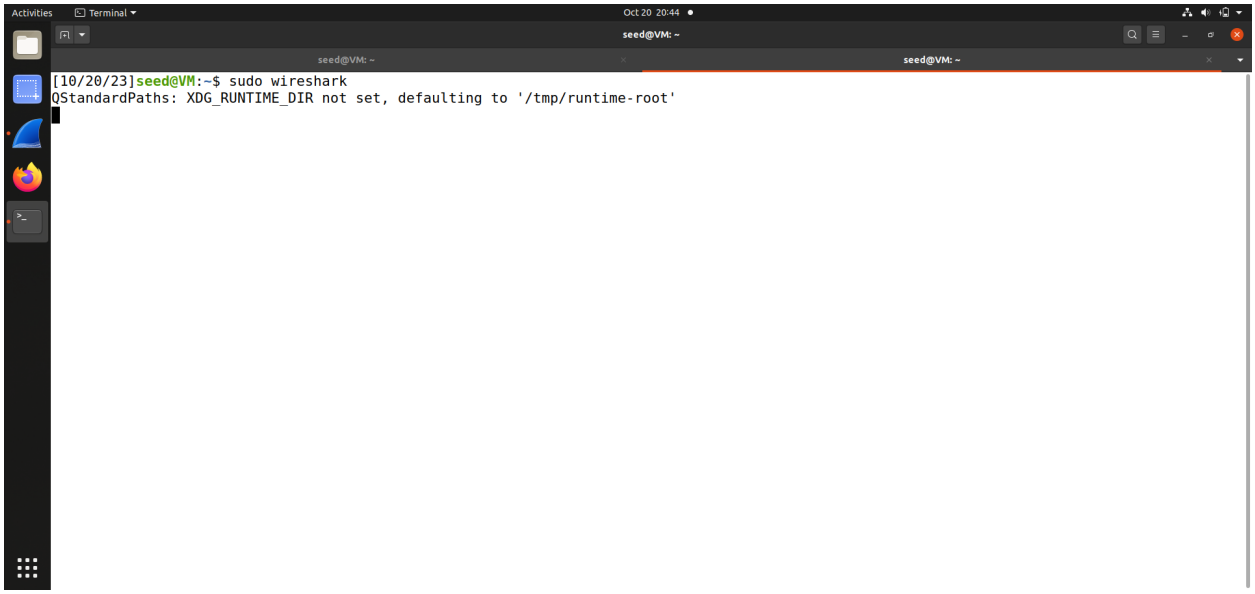



```
tcp        0      0 10.0.2.4:22             254.6.108.133:25459     SYN_RECV
tcp        0      0 10.0.2.4:22             248.230.151.100:39177   SYN_RECV
tcp        0      0 10.0.2.4:22             246.207.123.195:40067   SYN_RECV
tcp        0      0 10.0.2.4:22             246.33.53.140:3790      SYN_RECV
tcp        0      0 10.0.2.4:22             243.4.185.98:24708      SYN_RECV
tcp        0      0 10.0.2.4:22             254.36.153.90:2956      SYN_RECV
tcp        0      0 10.0.2.4:22             245.42.63.122:53098     SYN_RECV
tcp        0      0 10.0.2.4:22             255.46.245.175:14261    SYN_RECV
tcp        0      0 10.0.2.4:22             0.152.173.147:2174      SYN_RECV
tcp        0      0 10.0.2.4:22             242.124.137.251:44238   SYN_RECV
tcp        0      0 10.0.2.4:22             0.166.63.131:15635      SYN_RECV
tcp        0      0 10.0.2.4:22             254.152.126.254:3400    SYN_RECV
tcp        0      0 10.0.2.4:22             249.255.212.200:3391    SYN_RECV
tcp        0      0 10.0.2.4:22             252.168.250.59:12667    SYN_RECV
tcp        0      0 10.0.2.4:22             249.12.51.222:40597     SYN_RECV
tcp        0      0 10.0.2.4:22             240.21.165.79:28721     SYN_RECV
tcp        0      0 10.0.2.4:22             251.150.216.62:48318    SYN_RECV
tcp        0      0 10.0.2.4:22             251.38.239.54:7982      SYN_RECV
tcp        0      0 10.0.2.4:22             242.203.80.93:52354     SYN_RECV
tcp        0      0 10.0.2.4:22             245.204.235.233:12094   SYN_RECV
tcp        0      0 10.0.2.4:22             250.66.75.69:54485      SYN_RECV
tcp        0      0 10.0.2.4:22             252.48.124.180:15790    SYN_RECV
tcp        0      0 10.0.2.4:22             243.55.233.60:43571     SYN_RECV
tcp        0      0 10.0.2.4:22             242.143.137.218:36262   SYN_RECV
tcp        0      0 10.0.2.4:22             247.216.8.99:35330      SYN_RECV
tcp        0      0 10.0.2.4:22             243.248.144.123:61605   SYN_RECV
tcp        0      0 10.0.2.4:22             245.145.111.192:15712   SYN_RECV
tcp        0      0 10.0.2.4:22             240.143.78.126:24136    SYN_RECV
tcp        0      0 10.0.2.4:22             248.190.6.82:48012      SYN_RECV
tcp        0      0 10.0.2.4:22             245.162.101.223:46389   SYN_RECV
tcp        0      0 10.0.2.4:22             244.200.104.62:64275    SYN_RECV
```

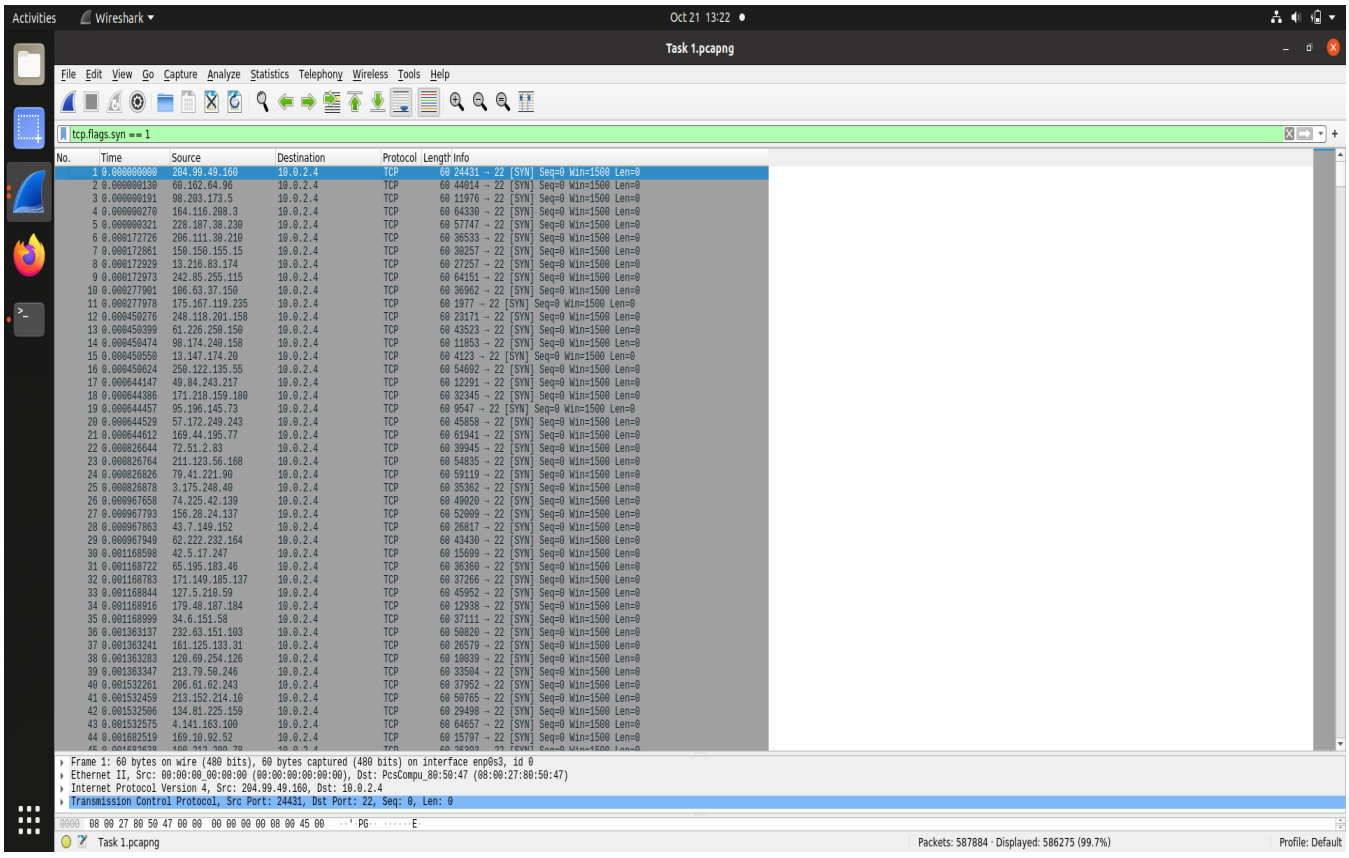


The above output was when SYN cookies were turned off. In this scenario, without SYN cookies enabled, my server was indeed experiencing a SYN flood attack, which resulted in a large number of half-open connections, as indicated by the numerous SYN_RECV states for port 22 in the ‘netstat’ output. This is expected behavior when SYN cookies are not active, and it signifies that the server is experiencing a flood of incoming connection requests that it is attempting to establish but hasn’t yet completed due to the attack.

While the SYN flooding attack was ongoing, I ran Wireshark on the victim VM to capture network packets. Launched Wireshark with root privileges to capture network traffic. Following command opens the Wireshark graphical interface with elevated privileges, allowing it to capture network traffic.



Analyzing network traffic captured in Wireshark when SYN cookies are disabled during a SYN flooding attack:

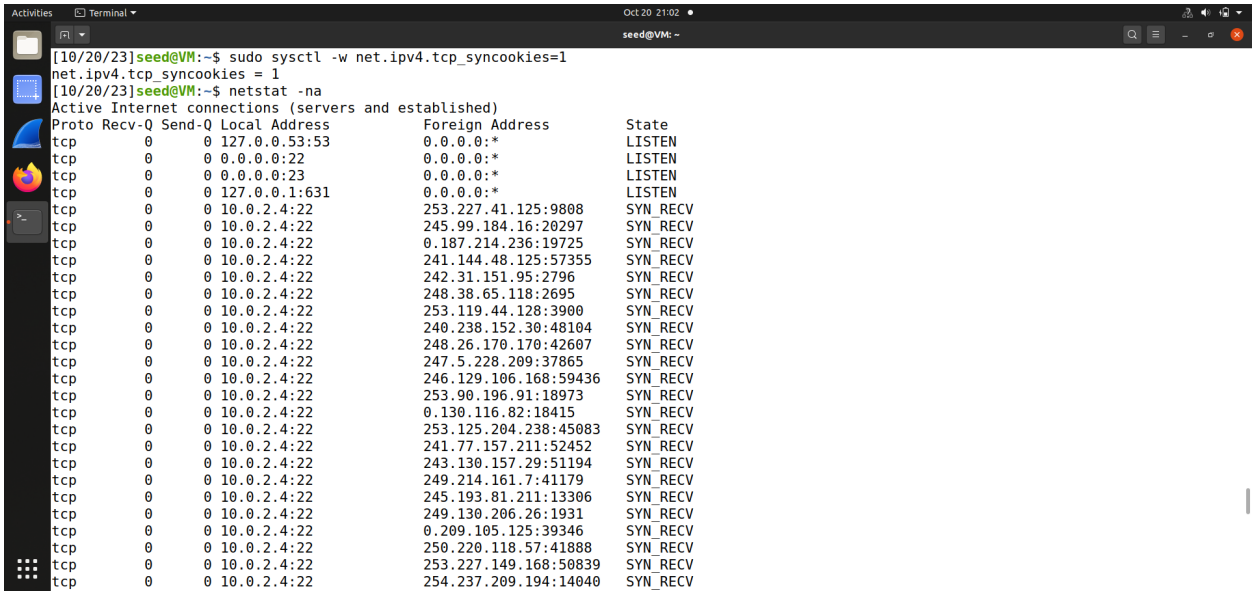


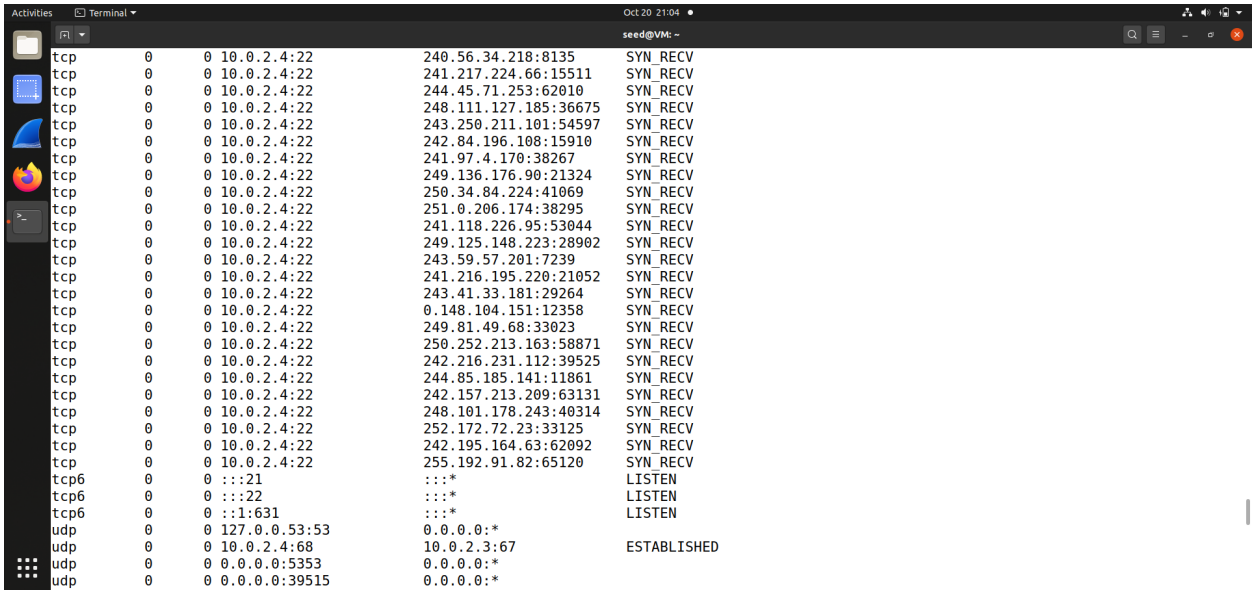
To narrow down analysis to the relevant packets. For a SYN flood attack, I filter SYN packets in

wireshark's filter bar 'tcp.flags.syn == 1' ; this filter displays only the packets with the SYN flag set, which are the initial synchronization requests sent during the attack.

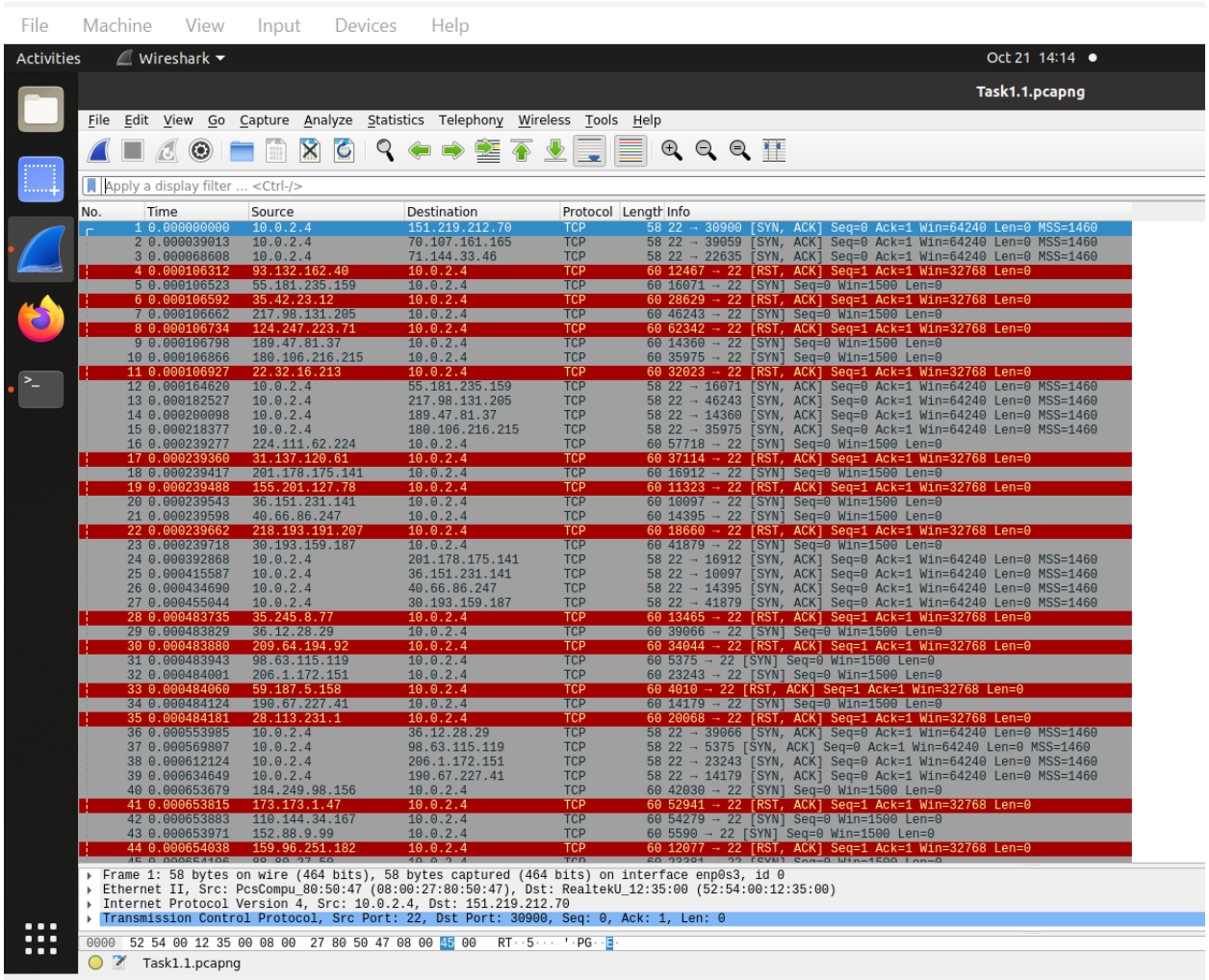
There is a significantly high number of incoming SYN packets, indicating the presence of a SYN flooding attack. Multiple IP addresses are sending SYN packets to the victim server. Various source IP addresses are involved in the attack. These addresses can be observed as the origin of the malicious connection requests. The SYN packets from different source IP addresses exhibit a pattern of similarity, as they all have the SYN flag set without completing the three-way handshake. The absence of SYN cookies means that the victim server is not implementing SYN cookies as a defense mechanism. This allows the attack to have a more direct impact on the server's resources.

Enable SYN Cookies: To test the effectiveness of SYN cookies, I turn on SYN cookies on the victim VM. After enabling SYN cookies, I repeated the attack and observed the results:

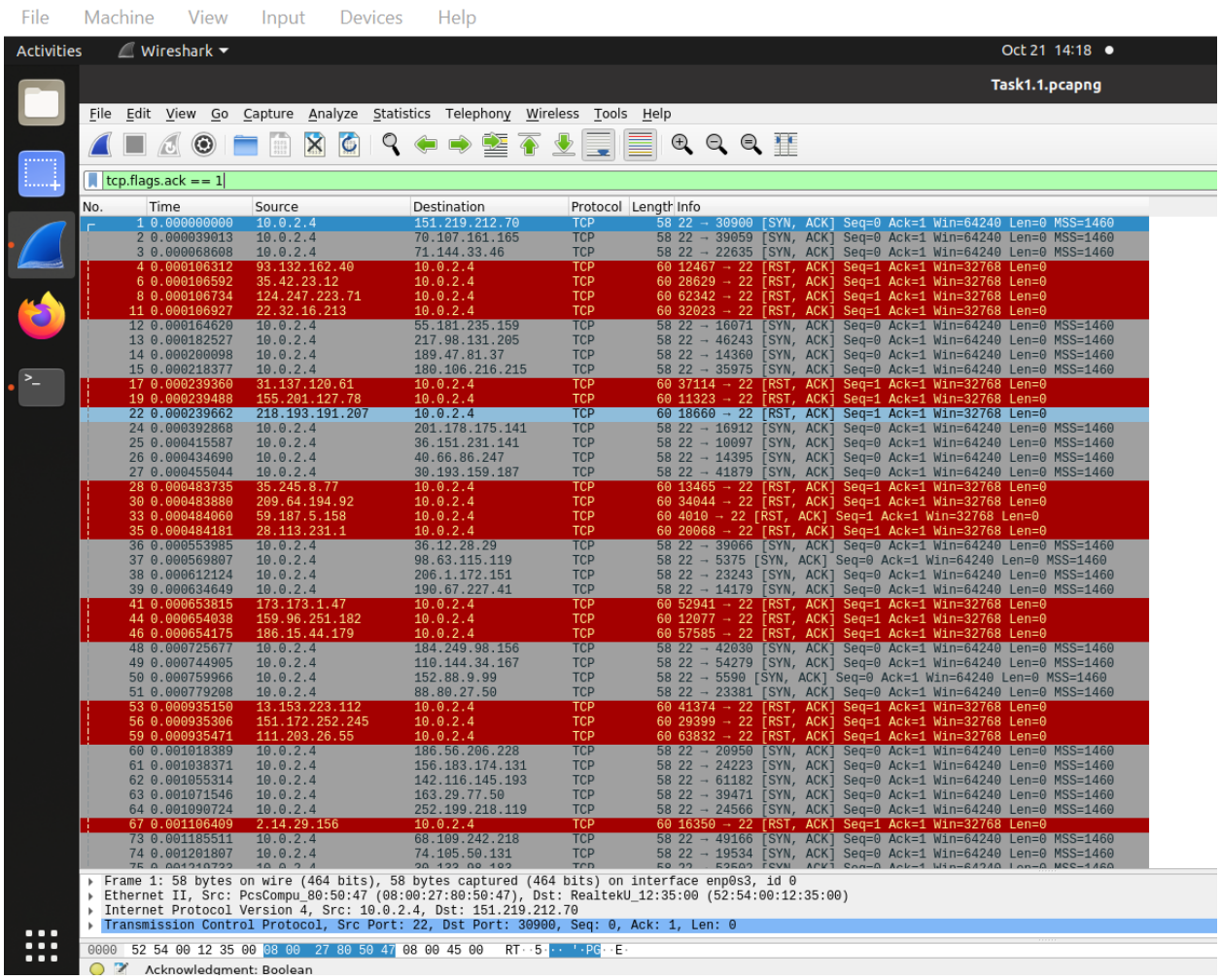




The output with ‘net.ipv4.tcp_syncookies=1’ appears to be similar to the output I provided earlier when ‘syncookies’ were disabled. This is because the ‘netstat’ command output primarily shows established and listening connections and doesn’t directly reflect the use of SYN cookies. To analyze the impact of SYN cookies, I examine network packet captures in wireshark as this is where I observe the behavior and effectiveness of SYN cookies in mitigating SYN flooding attacks.



I filter ‘tcp.flag.ack == 1’ which filters for packets with the ACK flag set, which indicate that the server is using SYN cookies to generate an ACK in response to a SYN:



The network traffic shows numerous incoming TCP packets with the SYN flag set. These packets represent connection initiation attempts from various client IP addresses. In response to the incoming SYN packets, the server generates packets with both the SYN and ACK flags set. These packets serve as SYN-ACK responses. The SYN-ACK packets typically have a TCP length of 0, indicating that no payload data is included in these packets. This is a characteristic of SYN cookies as they are designed to keep connection state information minimal. The absence of a full three-way handshake and the presence of SYN-ACK packets with TCP length 0 indicate that SYN cookies are successfully employed by the server to mitigate the SYN flooding attack. The SYN cookie mechanism, when enabled, effectively protects the server from SYN flooding

attacks by mitigating resource exhaustion. It does so by avoiding a full handshake, directly responding with SYN-ACK packets, and generating cookies for verification. This method ensures that even under a high volume of incoming SYN packets, the server remains available for legitimate connections.

In summary, SYN cookies are an efficient and resource-friendly method for protecting against SYN flooding attacks. They allow the server to conserve resources, maintain stability, and ensure service availability during such attacks, making them a valuable security feature.

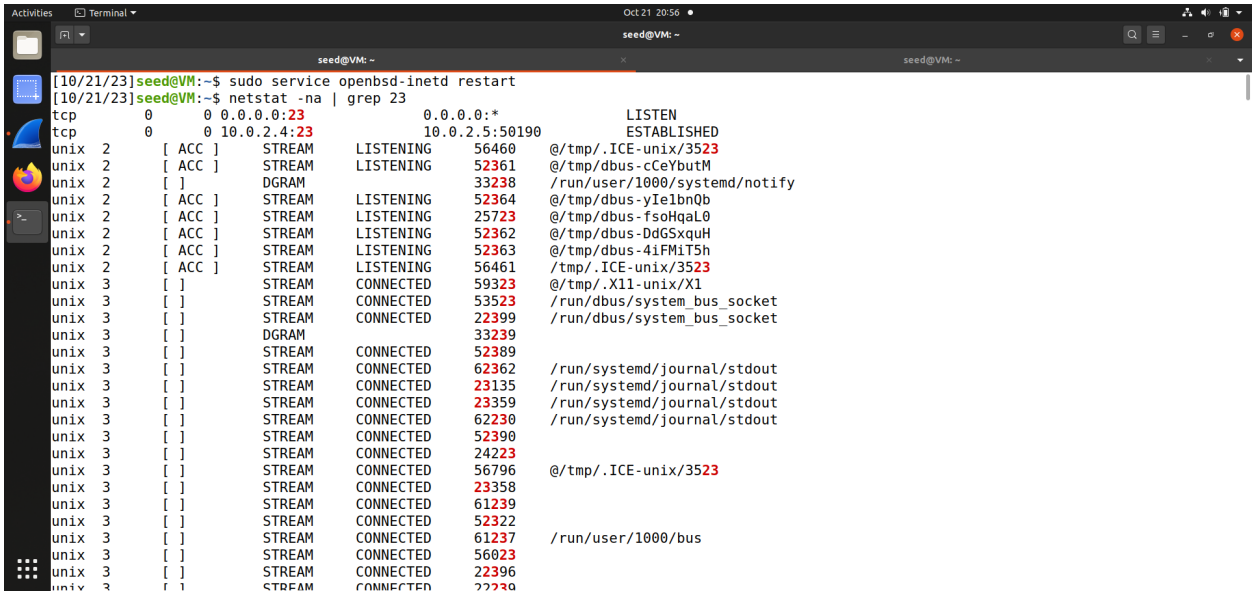
Task 2: TCP RST Attacks on telnet and ssh Connections

Telnet: Using Netwox:

On victim's start a telnet server and on attacker's establish a telnet connection to the remote host at IP address 10.0.2.4 on port 23, which is the default telnet port. Then I launched a TCP RST attack using Netwox with the source IP address as "10.0.2.4" and network device "enp0s3".

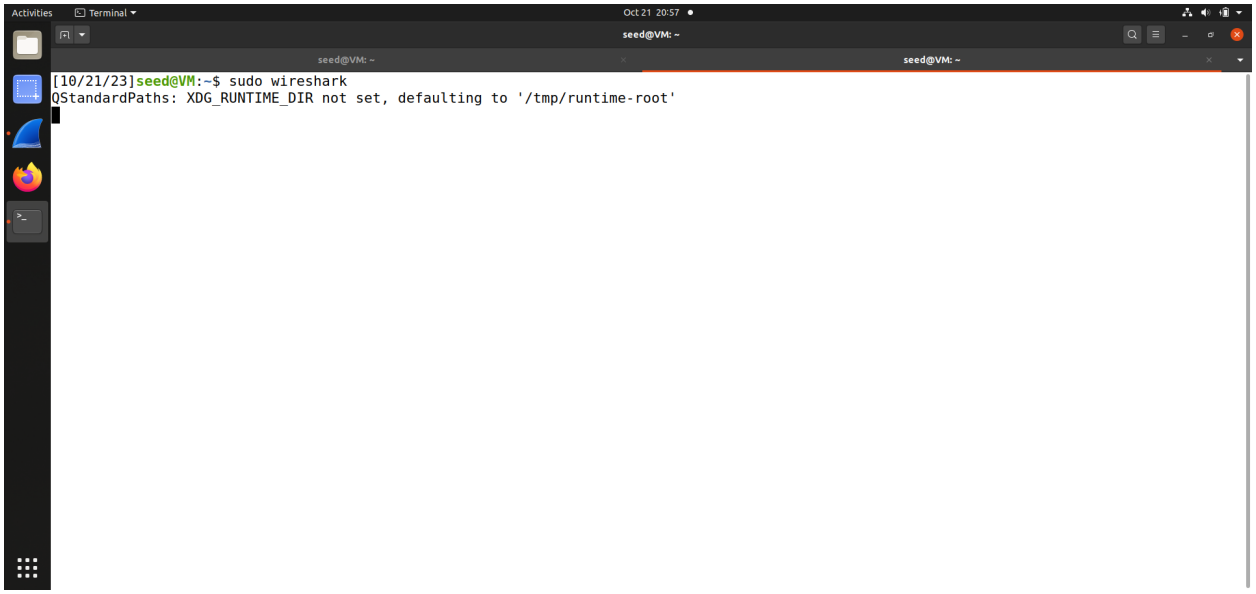
'sudo netwox 78 -d enp0s3 -s 10.0.2.4' this command sends RST packets from network device 'enp0s3' to reset all TCP connections to the target IP address 10.0.2.4, including any Telnet

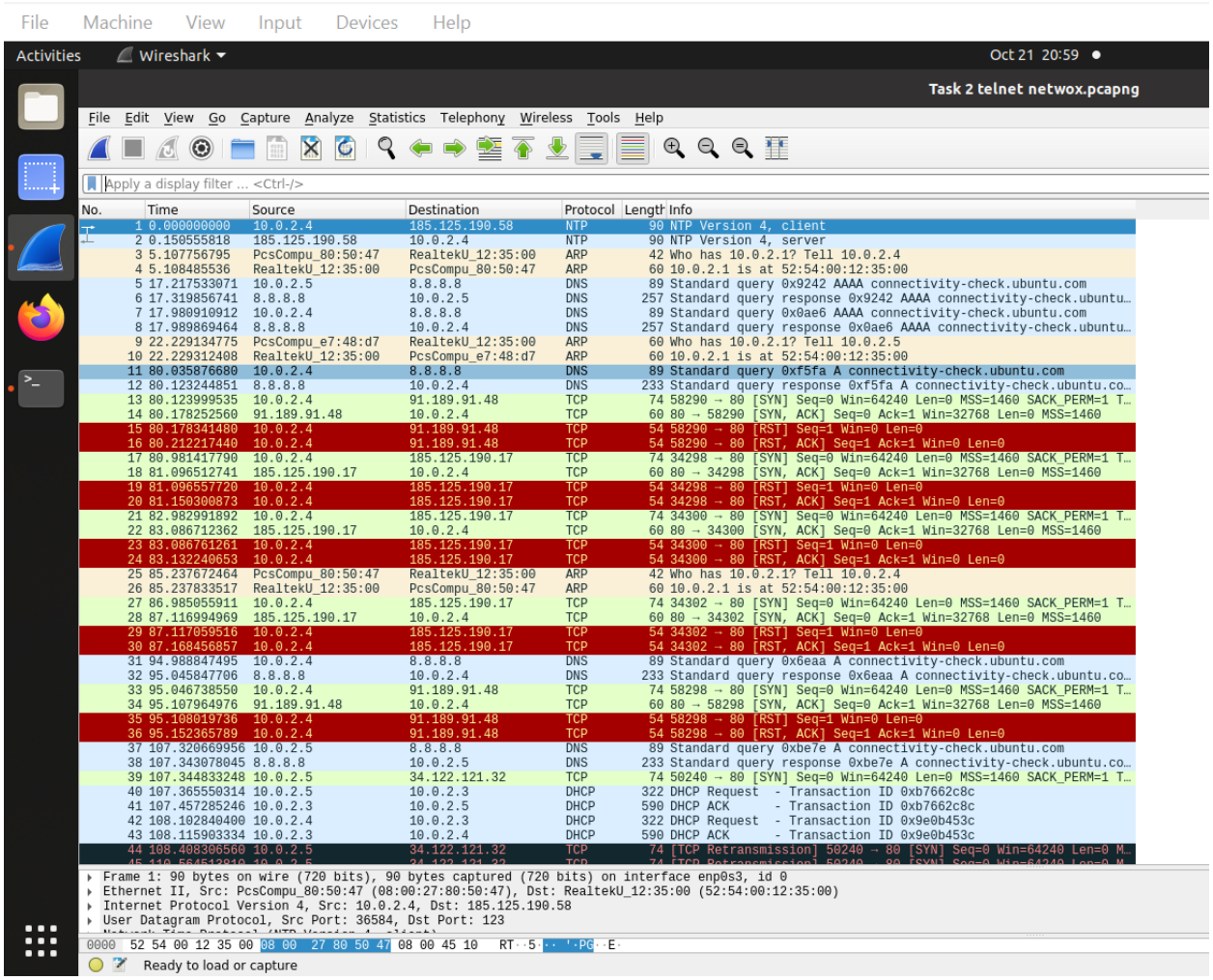
connection if they are active.



The output from ‘netstat -na | grep 23’ shows the current network connections on the victim. It appears that there is a telnet connection on port 23, as indicated by the line with “ESTABLISHED”. However, the ‘netstat’ command did not display the termination of a connection immediately after a TCP RST attack. It took some time for the connection state to change.

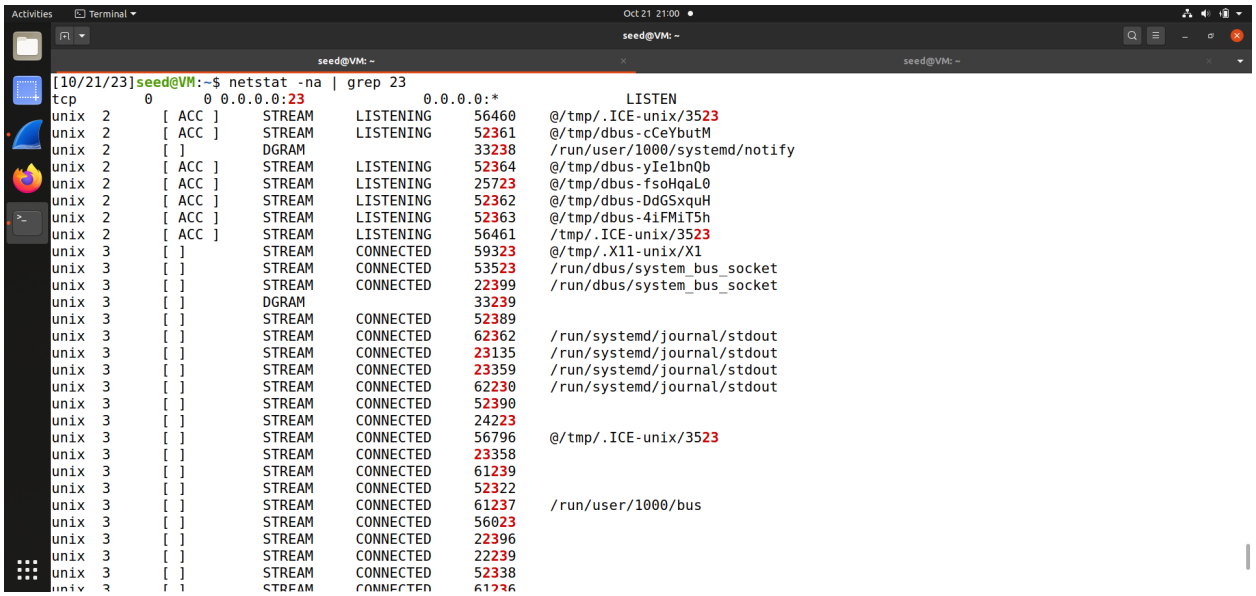
While on other side I ran wireshark to see the TCP RST attack:





Here I observed RST packets in wireshark, it indicates that the TCP RST attack was indeed sent

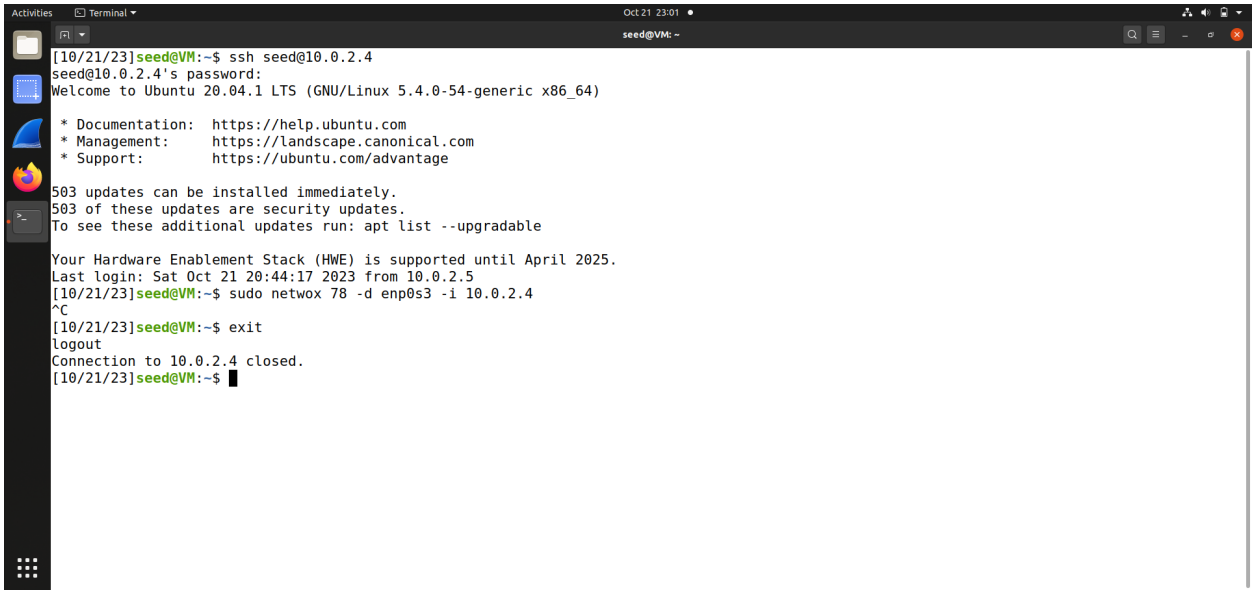
and received by the target (victim) in the telnet connection. The RST packet is used to forcibly terminate an established TCP connection. So, from a packet analysis perspective, the RST packet was sent, which should terminate the connection.



The above output indicates that there is no longer an established telnet connection on port 23. This suggests that the connection has been successfully terminated. In the ‘netstat’ output, the only mention of port 23 is related to the telnet service listening for incoming connections, and there are no established connections. The successful termination of the telnet connection aligns with the expected behavior when sending a TCP RST attack. The RST packet forces the termination of the established connection.

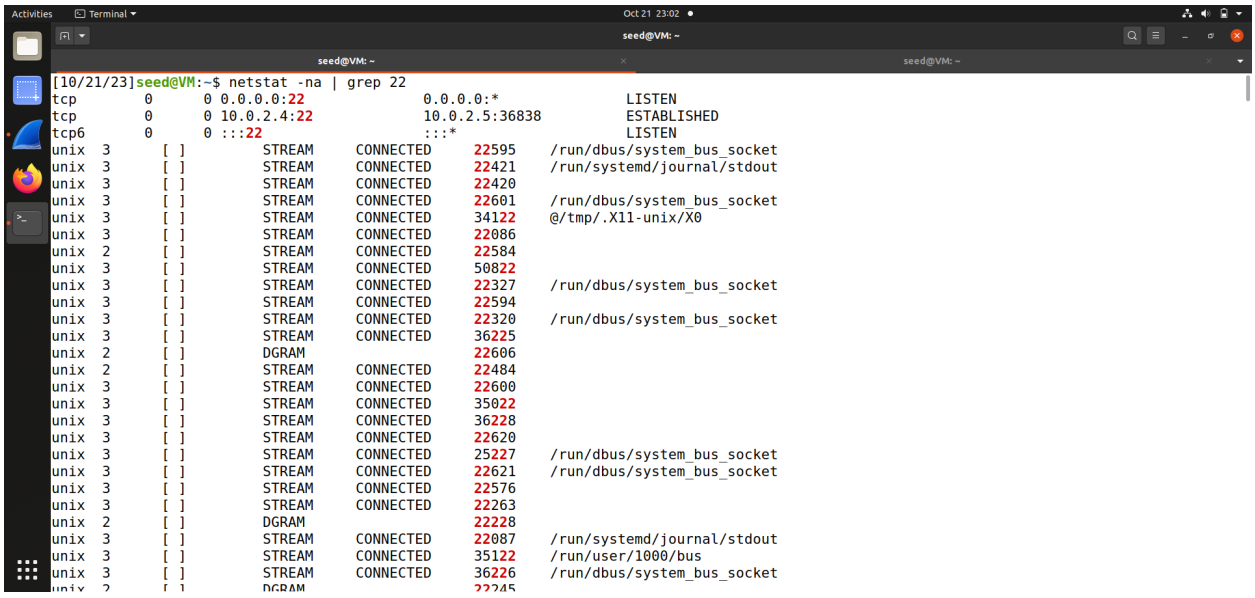
SSH: Using Netwox:

Establish an SSH connection between attacker machine and the victim machine:



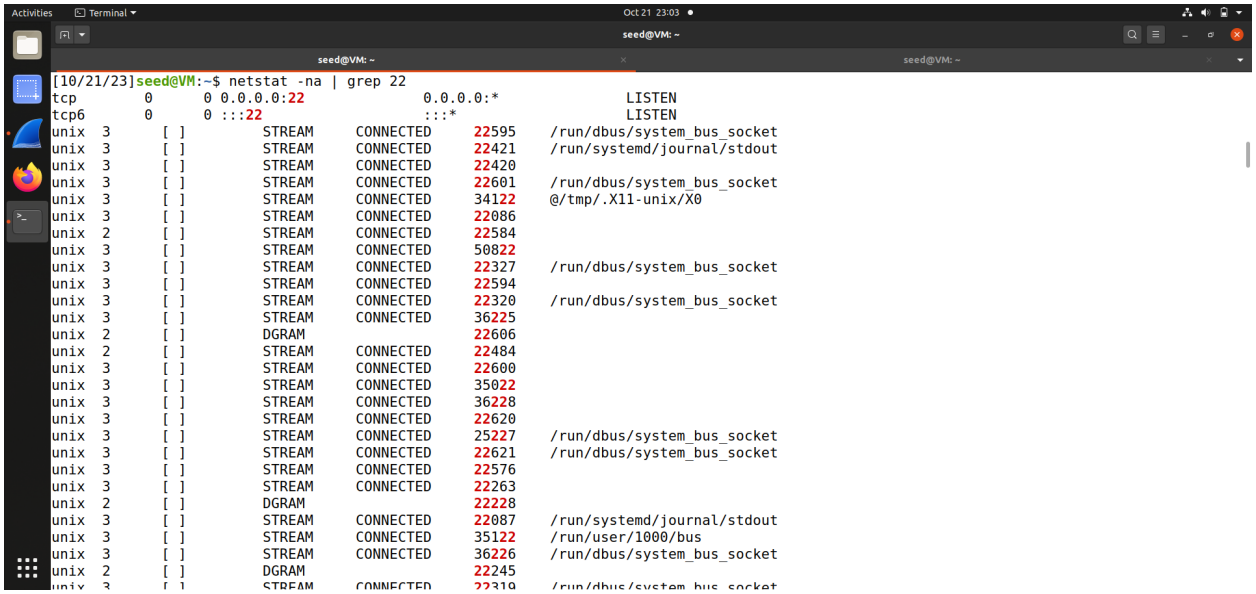
On the attacker machine, I use netwox to send a RST packet to terminate the SSH connection.

Verify the established connection before attack:



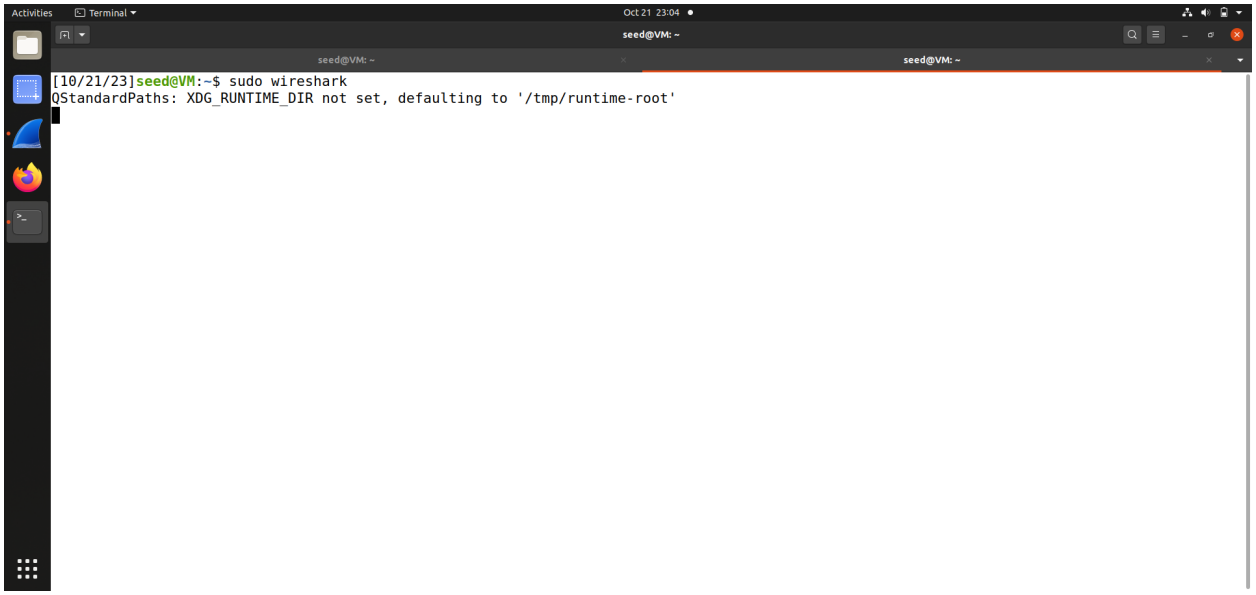
The above output seems that an SSH connection is established on port 22, and the connection is in the “ESTABLISHED” state.

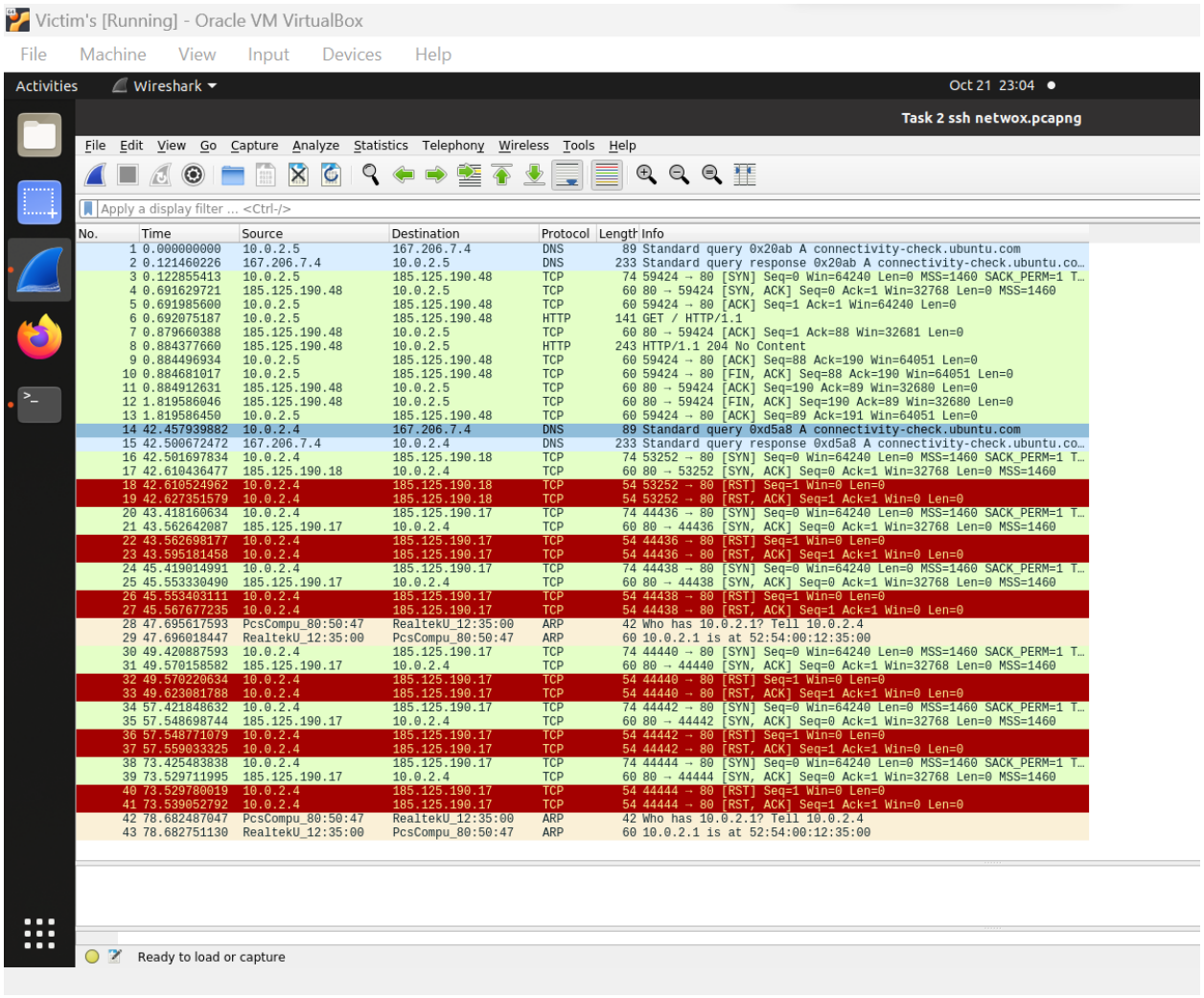
Verify the connection termination after attack:



The ‘netstat’ command does not show any established SSH connections which means it was able to terminate an SSH connection using ‘netwox’. SSH is a more secure and robust protocol compared to telnet. It is designed to be resistant to connection interruptions, and in many cases, an SSH connection will automatically close or be terminated after the session is finished.

Wireshark to see the TCP RST attack:





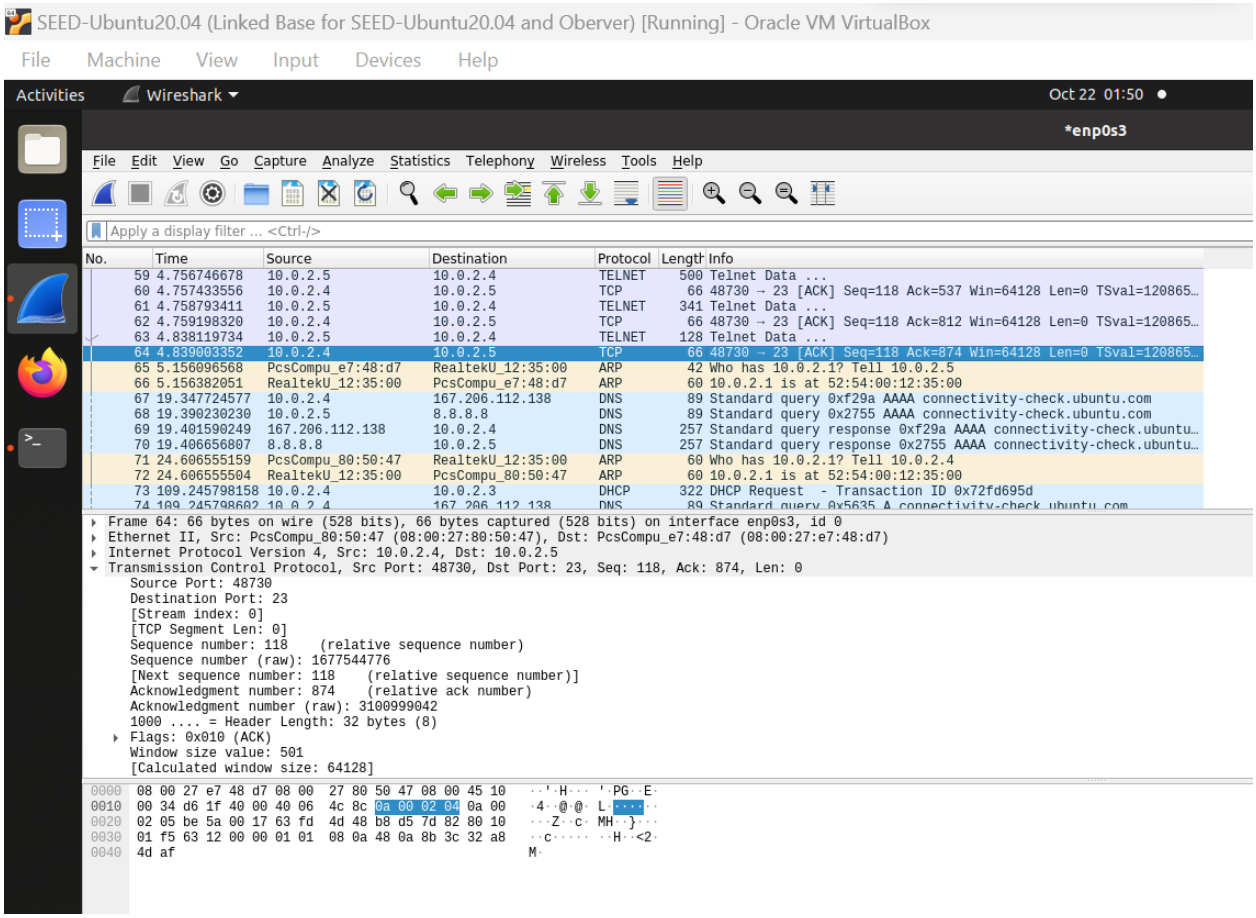
When analyzing network traffic using wireshark, “RST, ACK” packets were observed. These packets represent a TCP connection reset and acknowledgment, confirming that the SSH connection was terminated.

I observed the difference between SSH and Telnet is that SSH generally maintains better security and connection management compared to telnet. When I close an SSH session, the connection is promptly terminated and cleaned up. Telnet, on the other hand, keeps the connection listed for a longer period even after I have exited the session.

Telnet using Scapy:

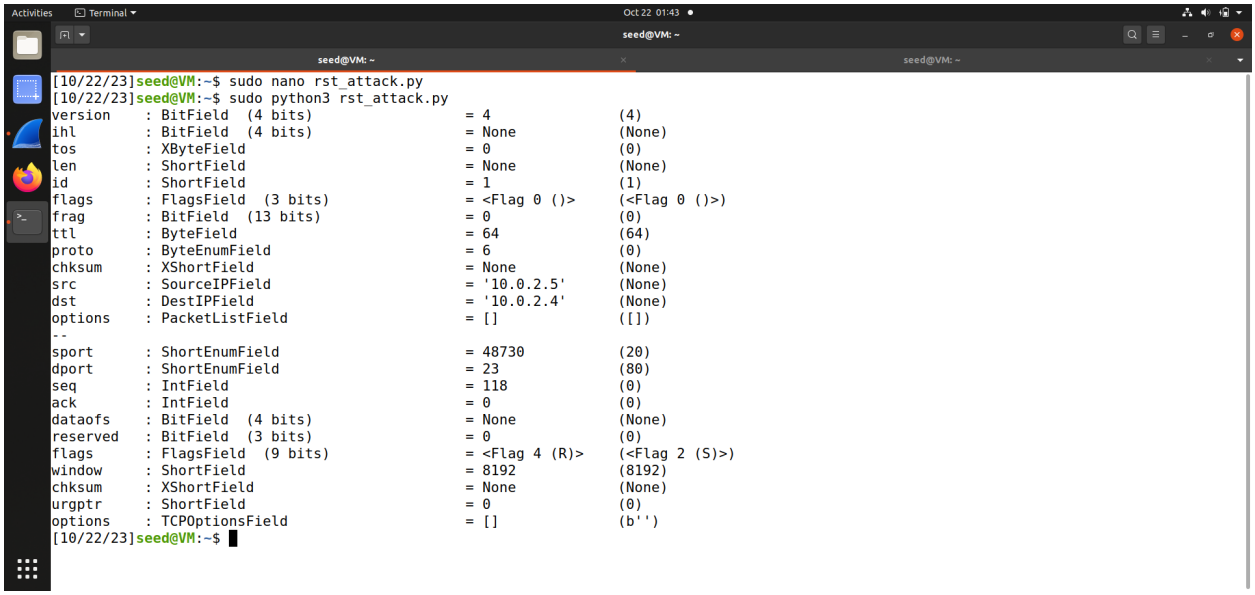
Performing a TCP RST attack on a Telnet connection using Scapy and observing the attack with Wireshark.

Started capturing packets on the network interface used for the Telnet connection:

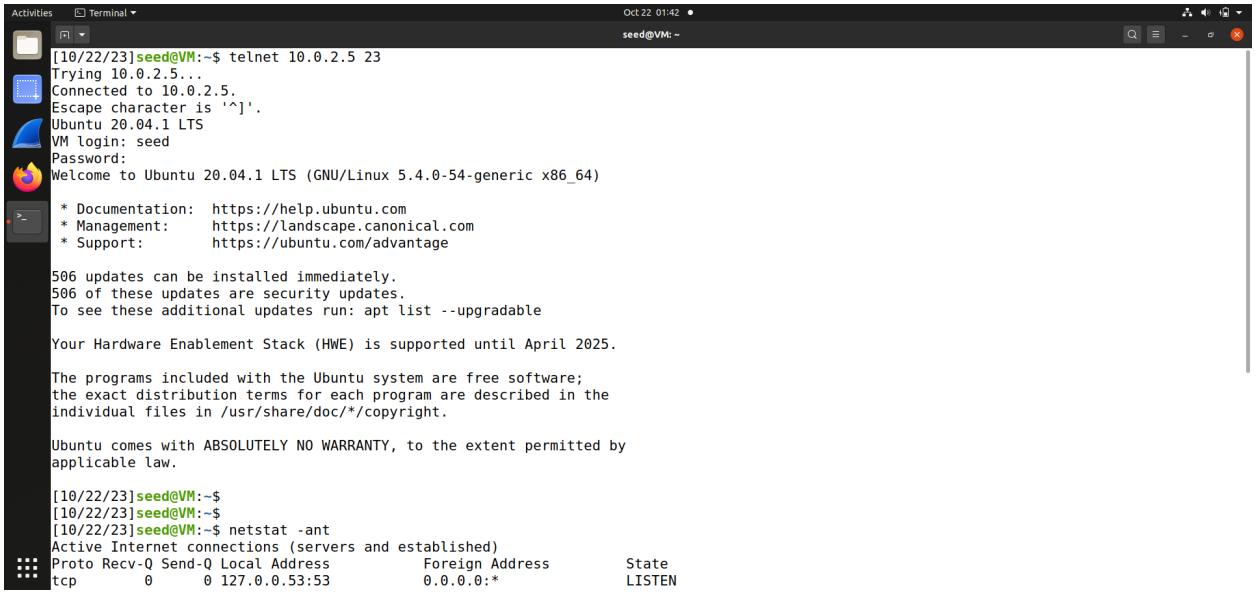


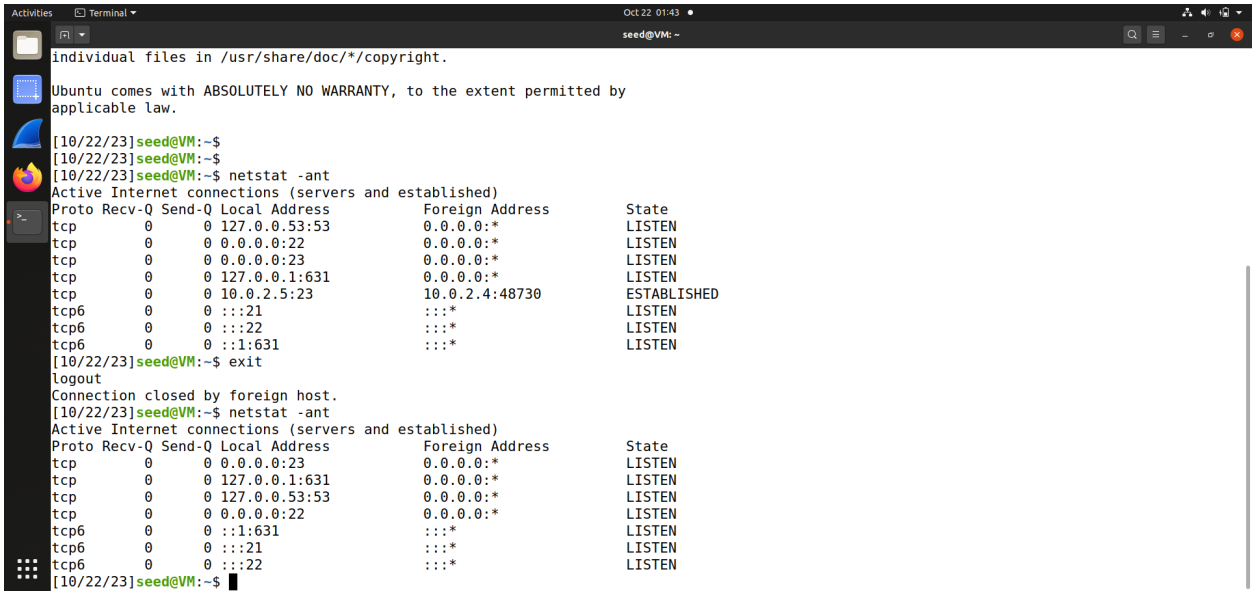
Create a Python script rst_attack.py, on Attack's machine using a text editor. Used Scapy to create a spoofed TCP RST packet targeting the Telnet connection. The script sets the source IP, destination IP, source port, destination port, flags to RST, and appropriate sequence numbers to match the Telnet session.

Run the Python script on attack's machine to send the spoofed RST packet:



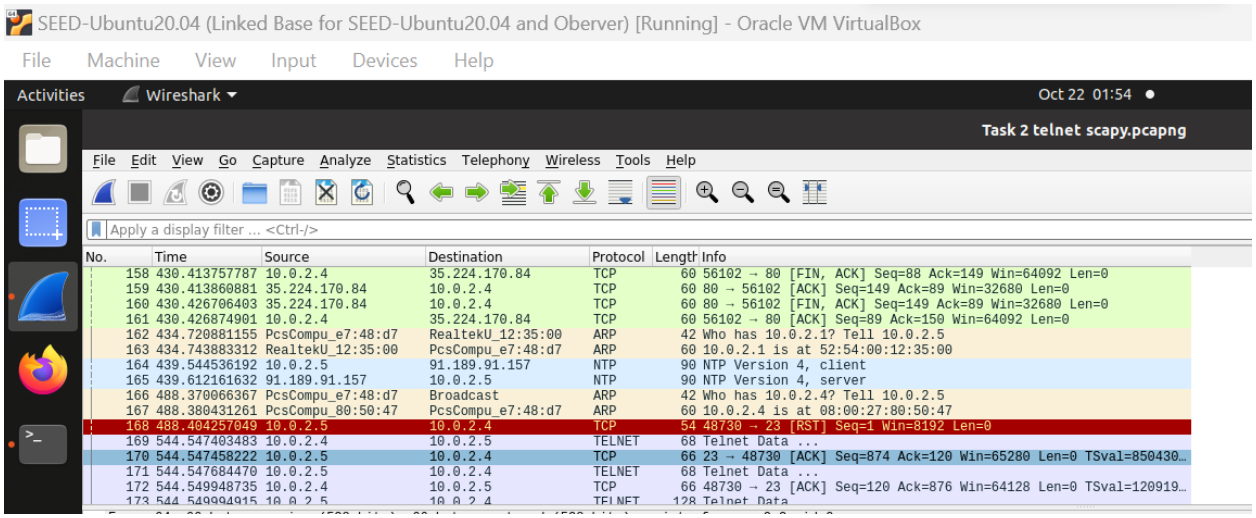
On victim's machine I had an active Telnet connection to another machine using the telnet command:

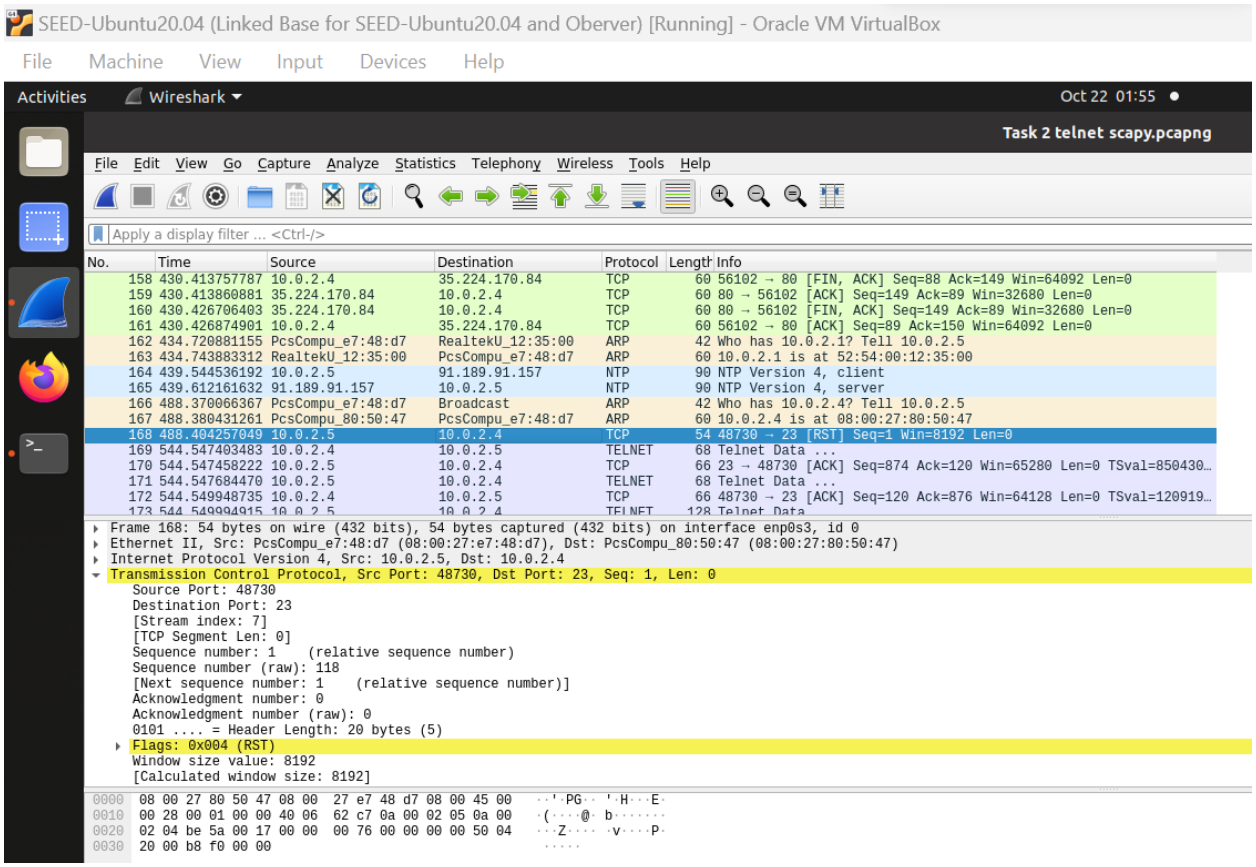




After the attack, the Telnet connection on victim's changed its state or disappeared from the netstat output. The Telnet session has been terminated due to the RST packet.

In Wireshark on Attacker's machine, I see the spoofed RST packet being sent to Victim:





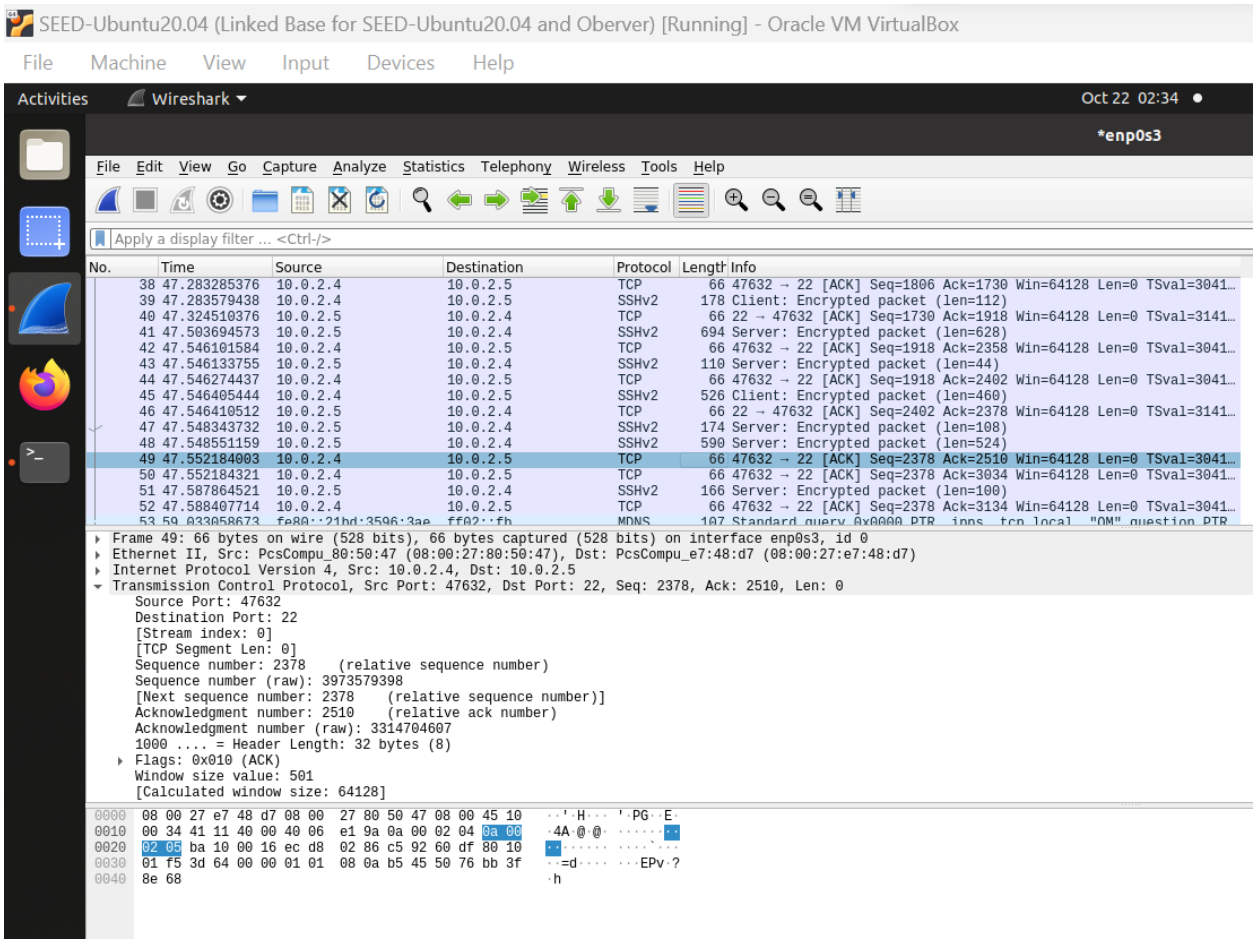
In the packet details, I notice that the spoofed RST packet has the RST flag set in the TCP header. This flag signals the abrupt termination of a connection.

The use of Scapy to conduct a TCP RST attack on a Telnet connection, coupled with Wireshark for network traffic analysis, successfully demonstrated the capability of an attacker to forcibly

terminate an established Telnet connection between two hosts. This emphasizes the importance of network security measures to detect and prevent such malicious attacks and highlights the significance of secure protocols when transmitting sensitive information over the network

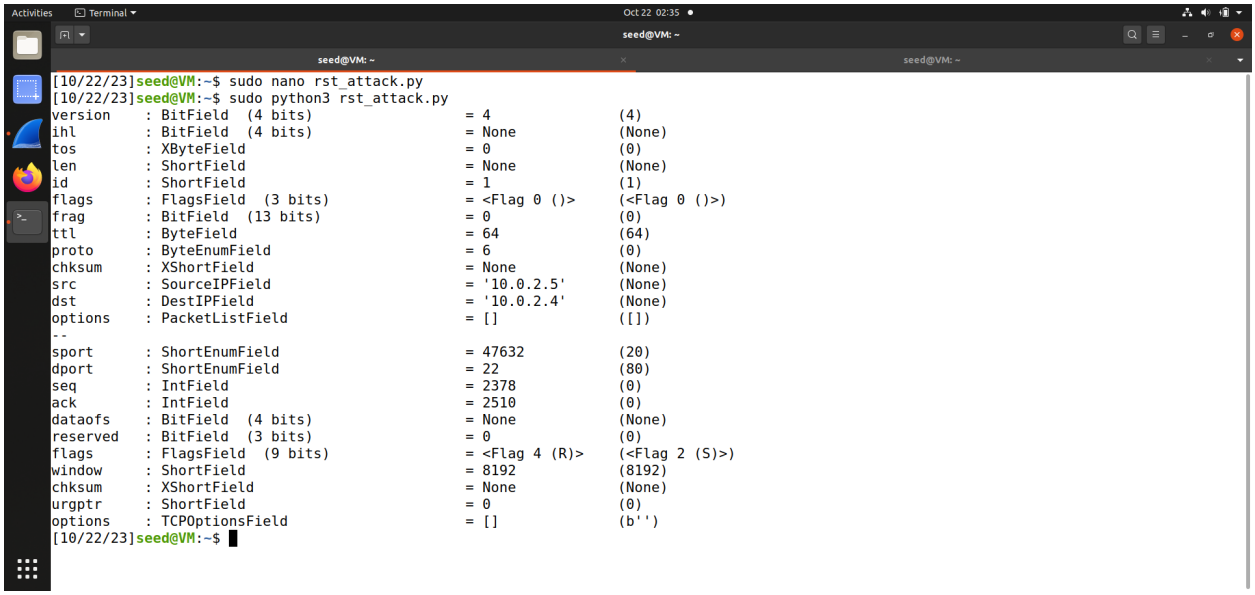
SSH using Scapy:

On Attacker's machine, started Wireshark to capture network traffic:

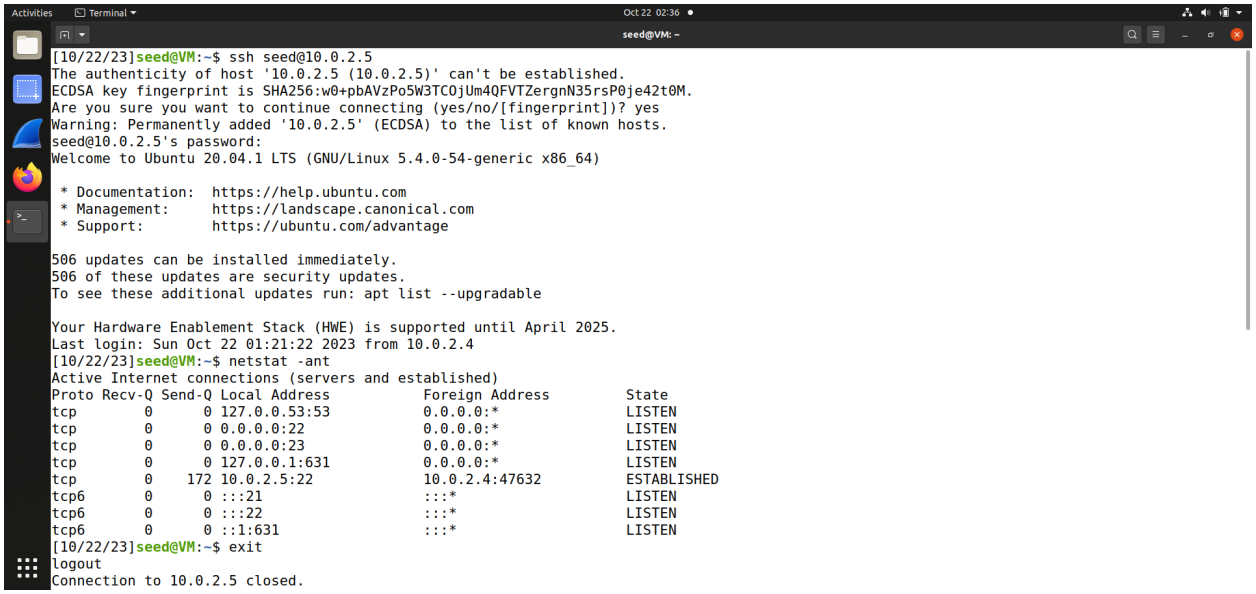


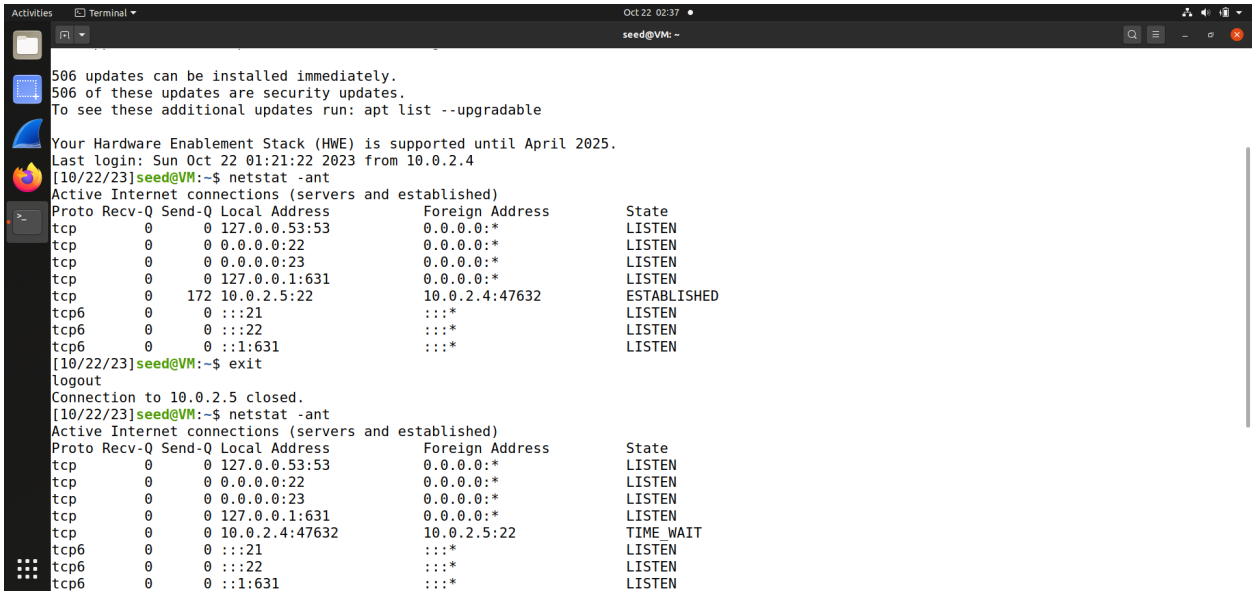
Attacker executes the Python script (rst_attack.py) to send a spoofed RST packet to Victim, targeting the SSH connection:

Run the Python script on Attacker's machine to send the spoofed RST packet:



On victim's machine I had an active SSH connection to another machine using the SSH command:

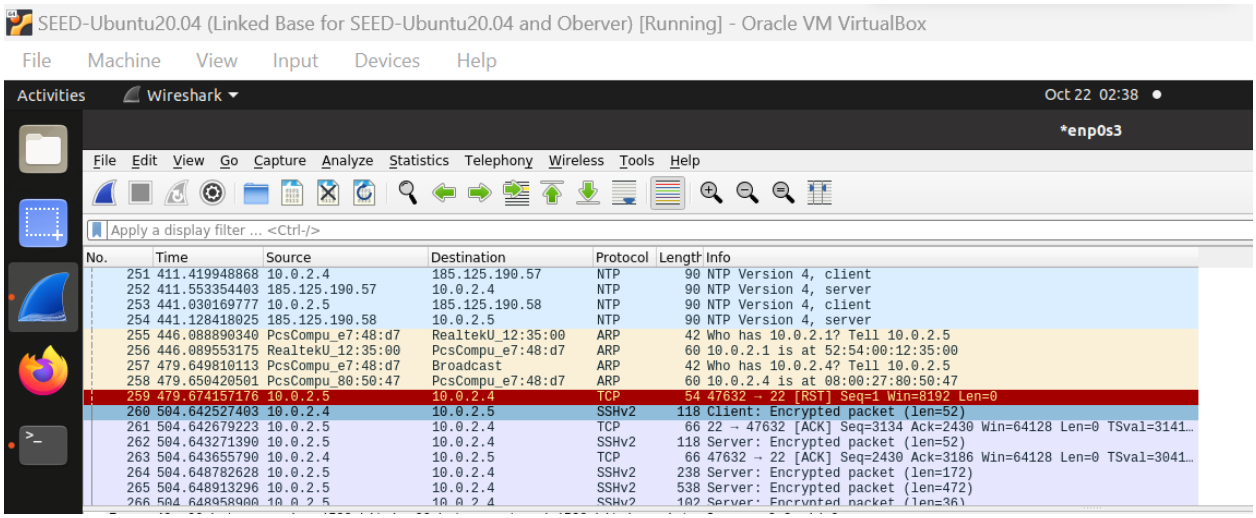


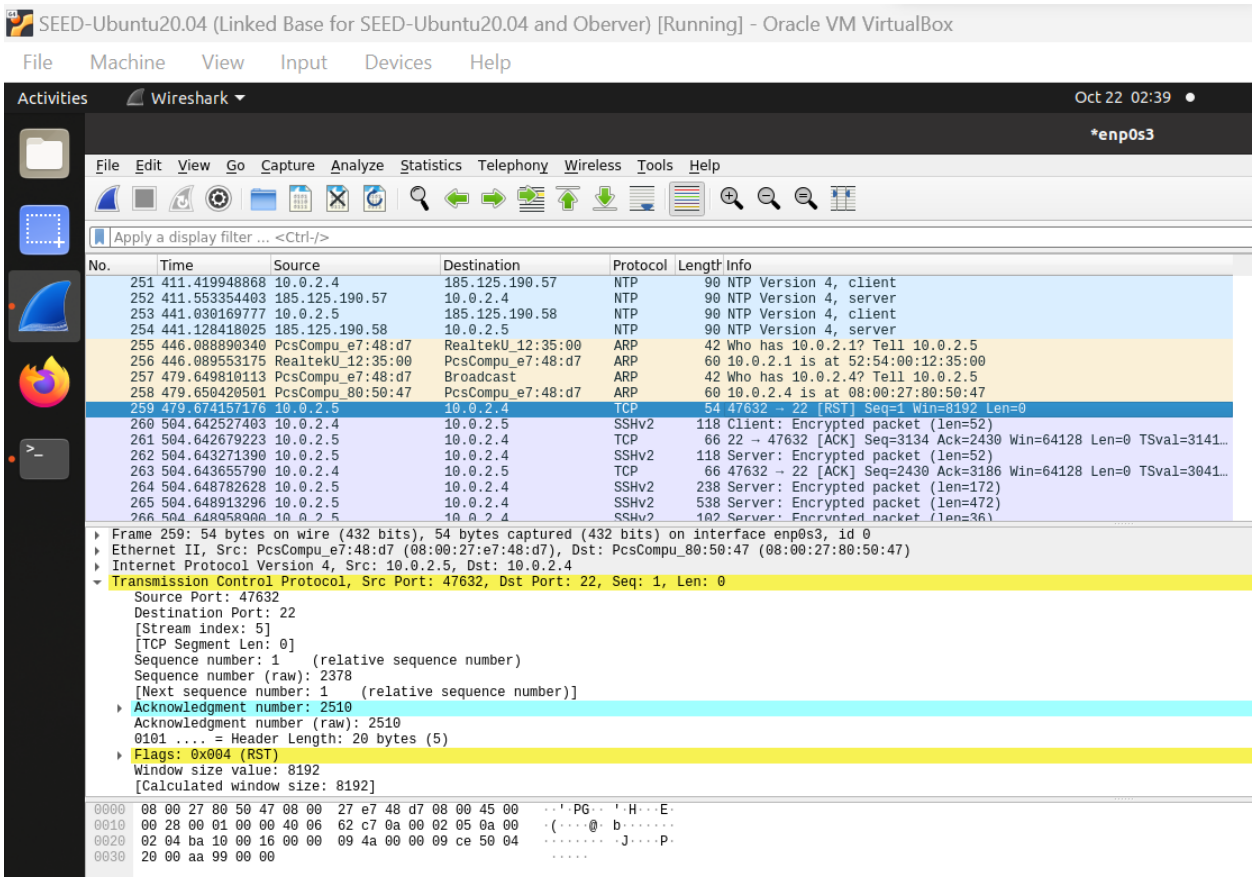


After the attack, the SSH connection on the victim changed to time wait from the netstat output.

The SSH session has been terminated due to the RST packet.

The spoofed RST packet being sent to Victim on Wireshark:



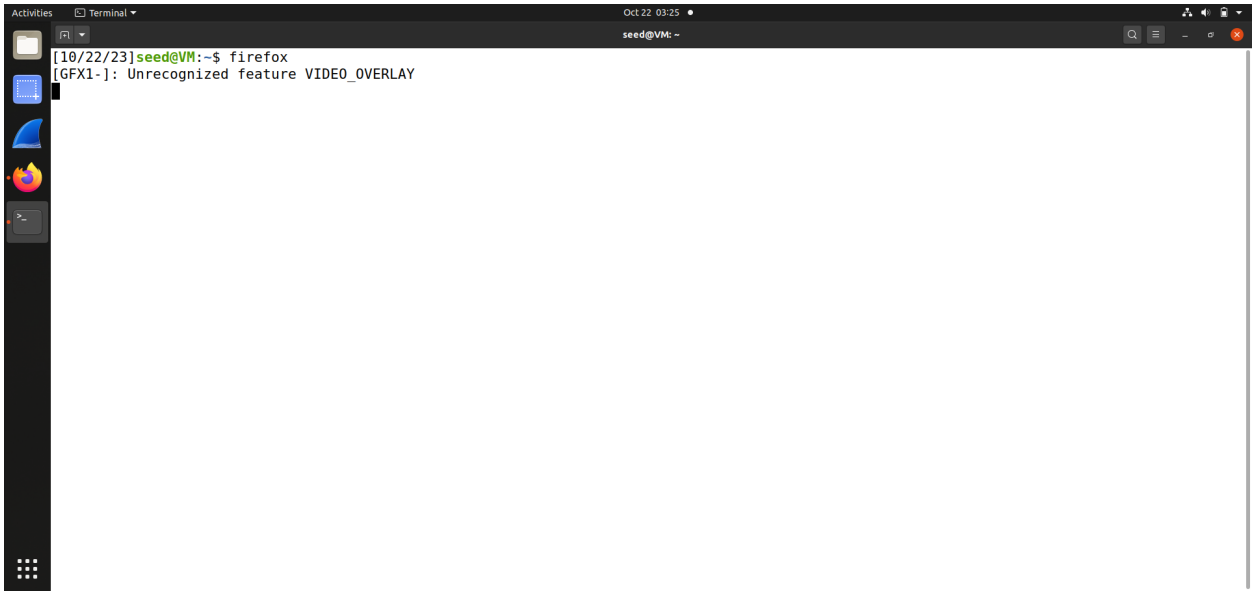


In the packet details, I notice that the spoofed RST packet has the RST flag set in the TCP header. This flag signals the abrupt termination of a connection.

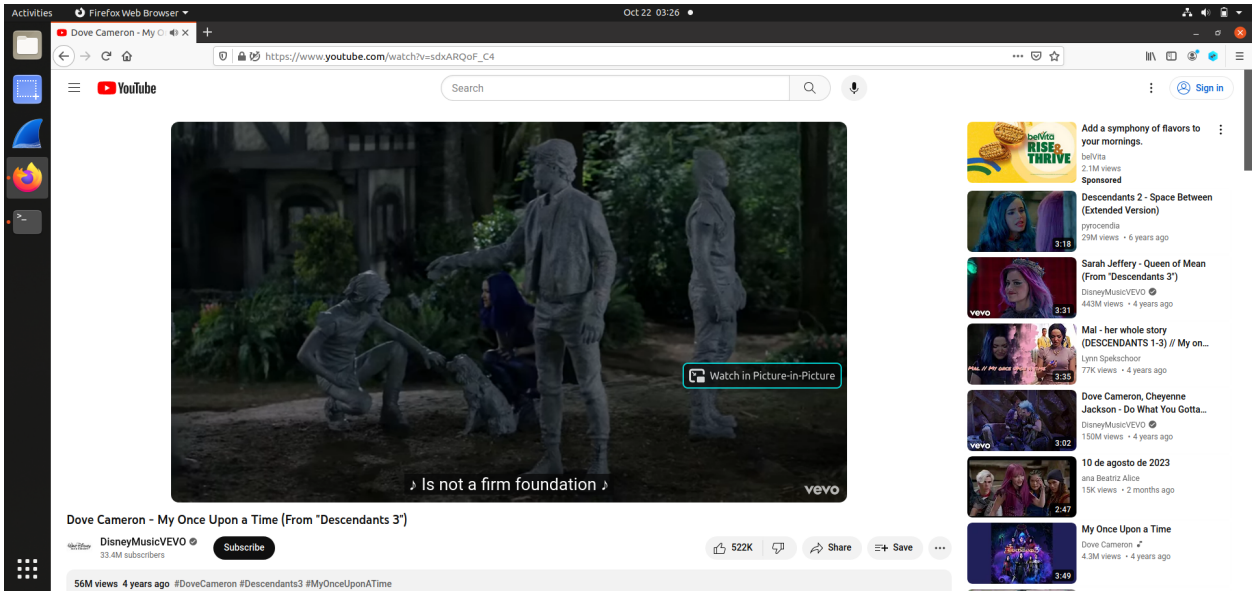
Task 3: TCP RST Attacks on Video Streaming Applications

To disrupt the video streaming on YouTube by injecting a TCP RST packet using the Netwox tool.

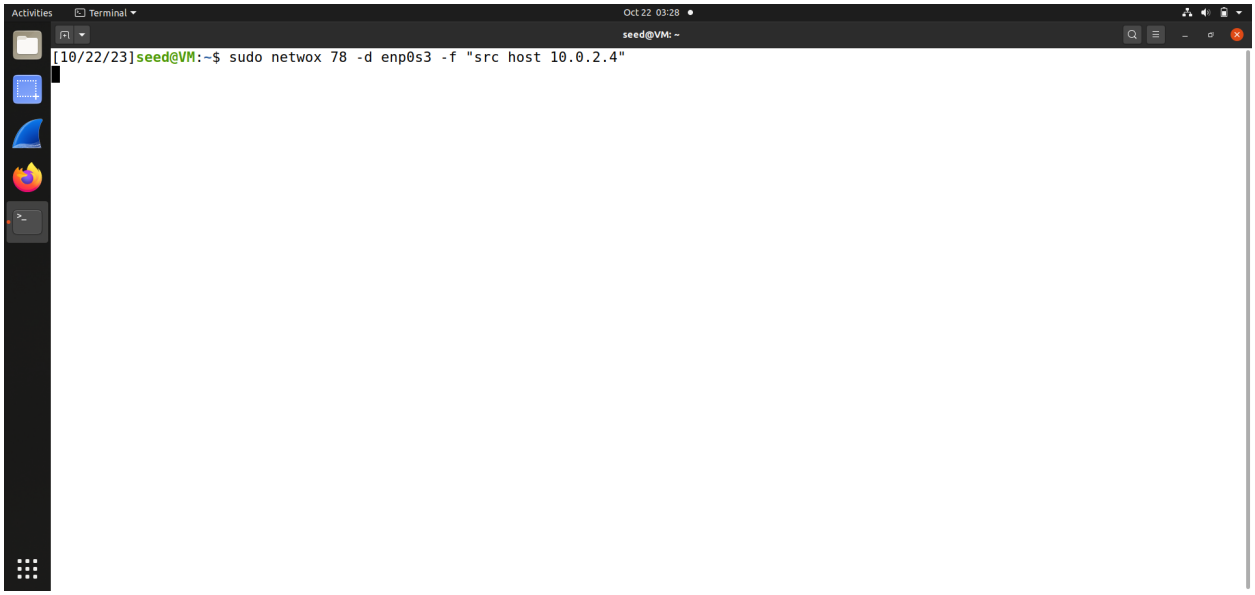
On victim's opened the Firefox web browser and typed www.youtube.com into the address bar:



On victim select and play video:

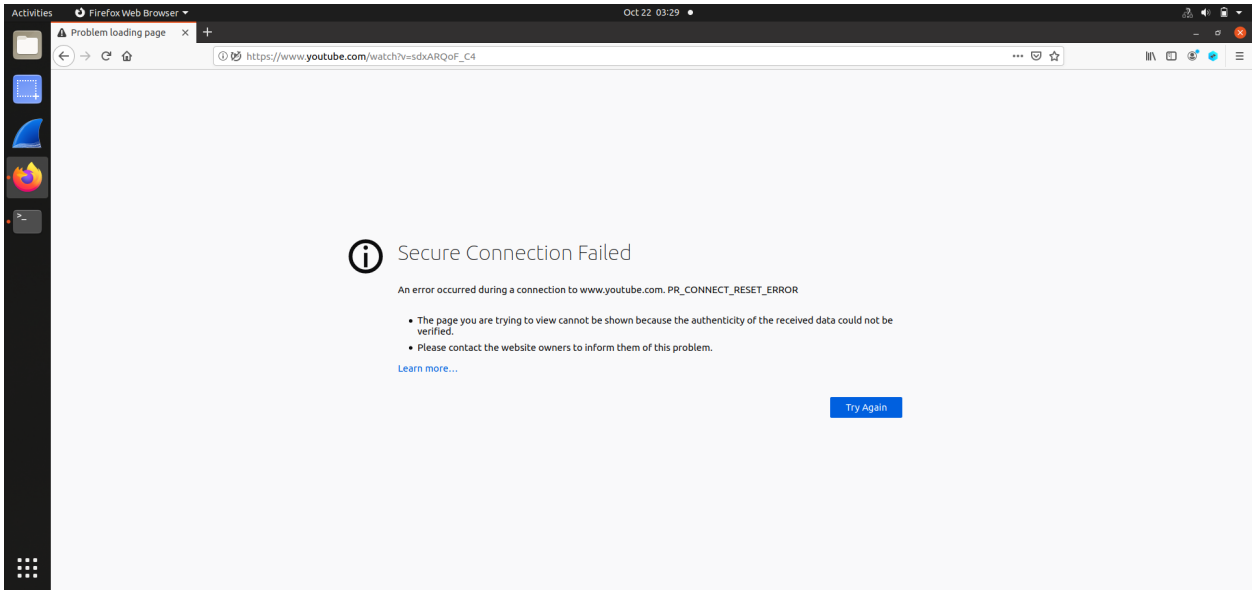


On the attacker's machine, I ran the following netwox command:



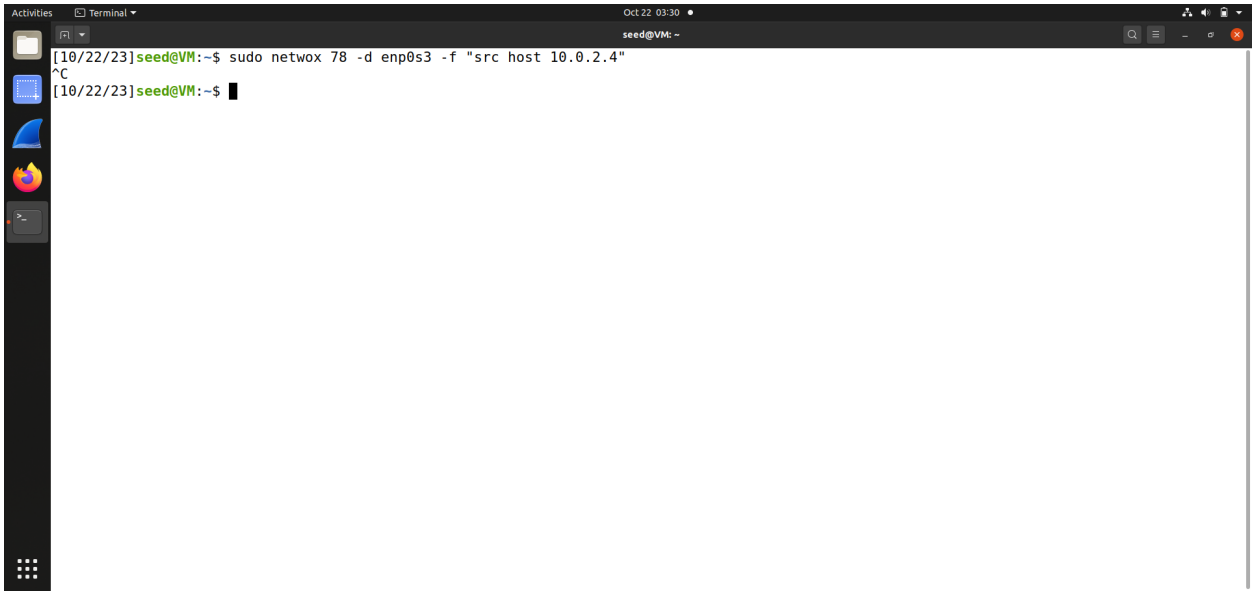
'Sudo' is used to run the command as a superuser to manipulate network traffic. 'netwox 78' is the command for sending a TCP RST packet. '-d enp0s3' specifies the network interface on which to send the packet. '-f "src host 10.0.2.4"' sets the filter to match packets with a source IP address of the victim's IP address.

As soon as I executed the command, a TCP RST packet with a forged source IP address was sent from the attacker's machine to the victim's machine.

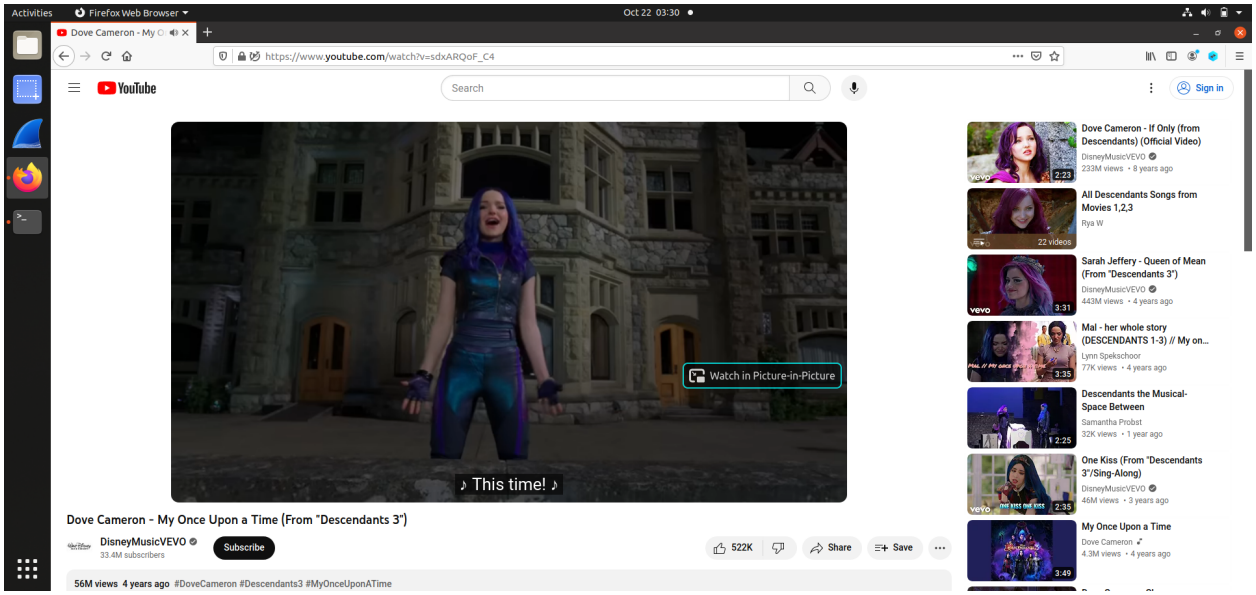


Upon receiving the forged RST packet, the victim's browser interprets this as a connection reset request and aborts the existing TCP connection to YouTube. This leads to the interruption of the video streaming. The victim's browser displays an error message, such as "Secure Connection Failed," indicating that the connection with YouTube is no longer secure.

Stopped the attack by pressing 'Ctrl+C' on the attacker's machine, the injection of RST packets ceased:



Once the attack is stopped, the victim's browser attempts to re-establish a connection with YouTube:



Observation: The attack effectively disrupted the video streaming on the victim's machine temporarily. The victim's browser interpreted the injected RST packet as a request to terminate the secure connection. As soon as the attack stopped, the victim's browser attempted to re-establish the connection with YouTube, and the video resumed.