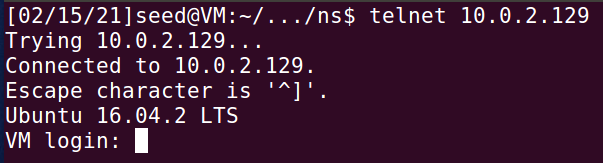
# 50.020 Network Security Lab 2 (TCP/IP Attack)

## Task 1: SYN Flooding Attack

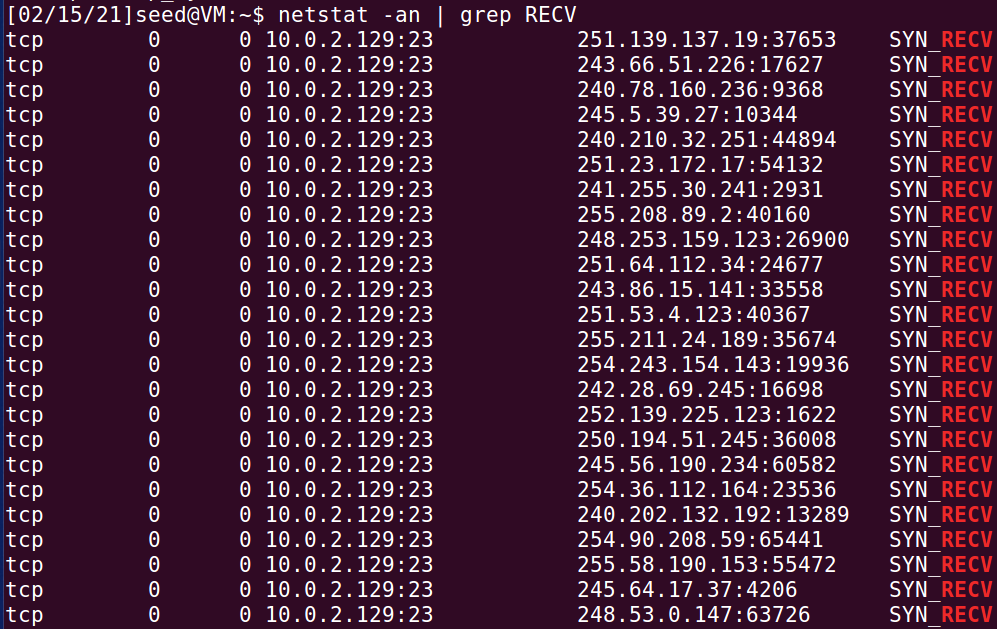
Setting up the Environment for the Experiment

The Client, Server and Attacker VMs are set up with the following respective IP addresses: 10.0.2.128, 10.0.2.129, 10.0.2.130. All three machines are run on the same host.

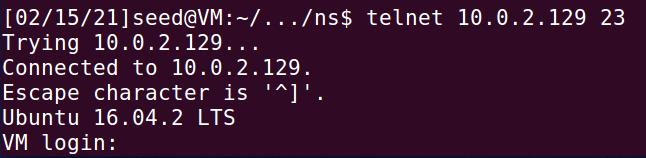
Carrying out SYN Flooding with SYN Cookie Enabled

By default, the Server is listening at port 23 for Telnet connections. To confirm that clients can connect to the Server via Telnet, we run the Telnet command on the Client:  


The Server responds by establishing half-opened connects and this can be seen by running netstat:

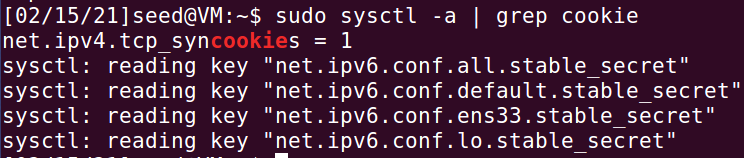


However, the Client can still connect to the Server via Telnet:



This shows that the attack is unsuccessful because if the attack is successful, the Server should not be able to take in any new connections and the Client should not be able to connect to port 23 as a result.

Investigating this, SYN Cookie is shown to be turned on at the Server:



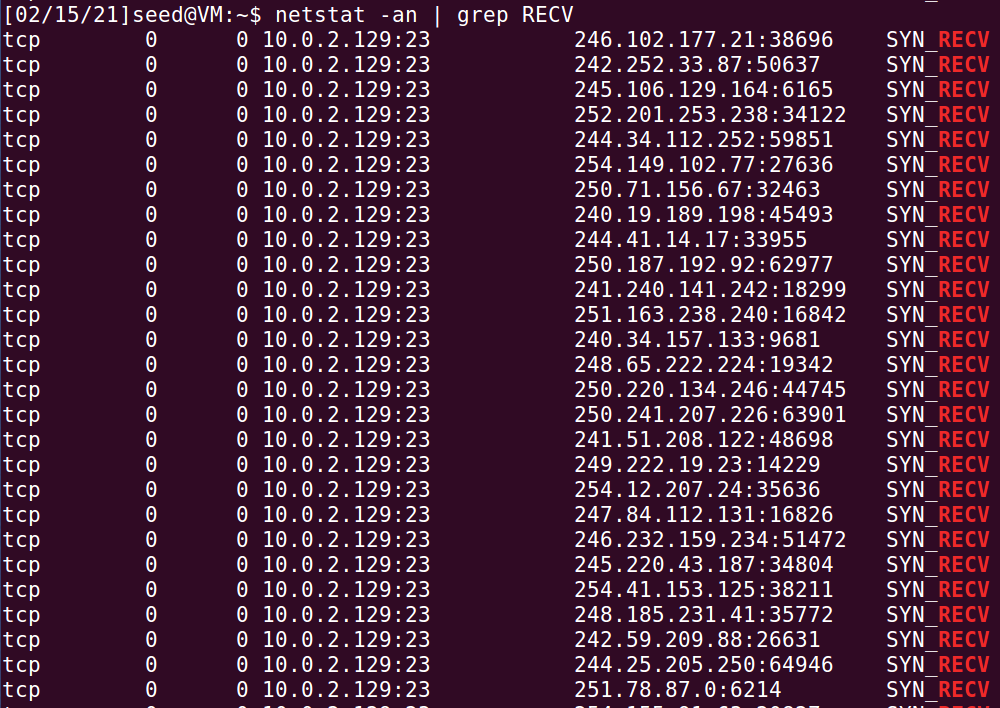
which explains why the attack is unsuccessful.

Carrying out SYN Flooding with SYN Cookie Disabled

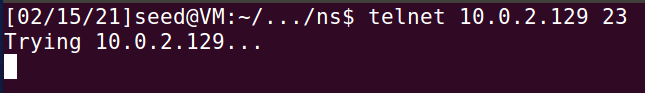
We proceed to turn of the SYN Cookie mechanism on the Server:



And proceed to carry out the SYN flooding experiment one more time. Half-open connections are still established as expected:



However, this time, the Client is unable to connect to the Server:



The attack is successful as this shows that the Server can no longer accept any more connections.

Describe why the SYN cookie can effectively protect the machine against SYN flooding attack

The SYN cookie will alter the mechanism of the way the server handles SYN messages:

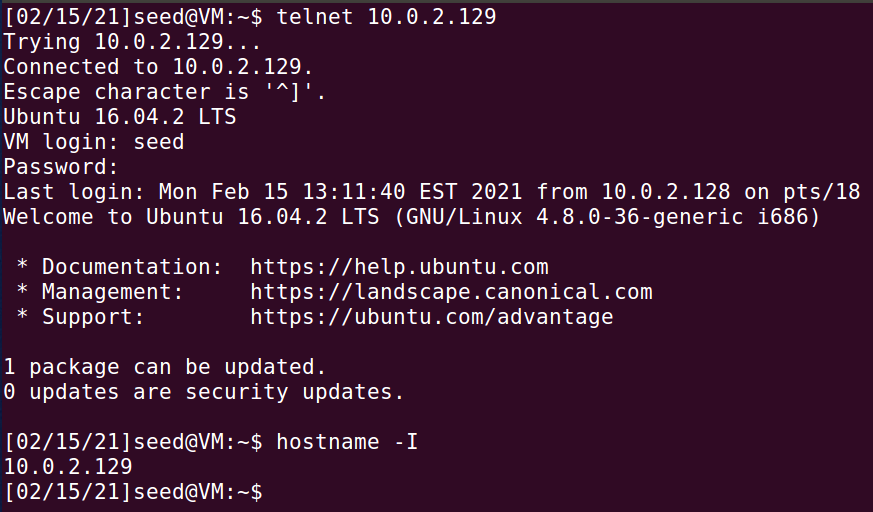
* After a server receives a SYN packet, it calculates a keyed hash (H) from the information in the packet using a secret key that is only known to the server
* This hash (H) is sent to the client as the initial sequence number from the server. H is called SYN cookie
  + The first 5 bits are a timestamp
  + The next 3 bits are an encoded value representing the maximum segment size
  + The final 24 bits are a MAC of the server and client IP addresses, the server and client port numbers, and the previously used timestamp, computed using a secret key
* The server will not store the half-open connection in its queue
* If the client is an attacker, H will not reach the attacker
* If the client is not an attacker, it sends H+1 in the acknowledgement field
* The server checks if the number in the acknowledgement field is valid or not by recalculating the cookie

SYN cookie is just a way for server to not store records in TCB queue, so it would not face the problem of having a full TCB queue which is vulnerable to the SYN flooding attack.

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

TCP RST Attack on Telnet using Netwox

The Telnet connection between Client and Server is established first:

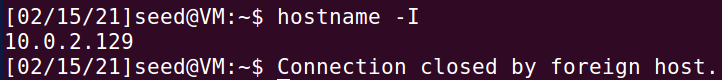


Following this, the Netwox command is run on the Attacker machine to carry out the TCP RST Attack:

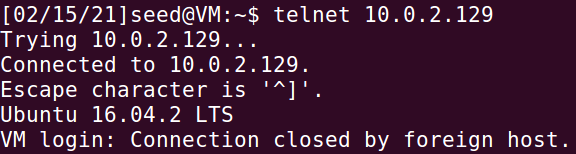


This filter is applied to filter out packets from TCP sessions involving the Server (10.0.2.129).

Shortly after this, the Telnet connection is broken as shown on the Client:



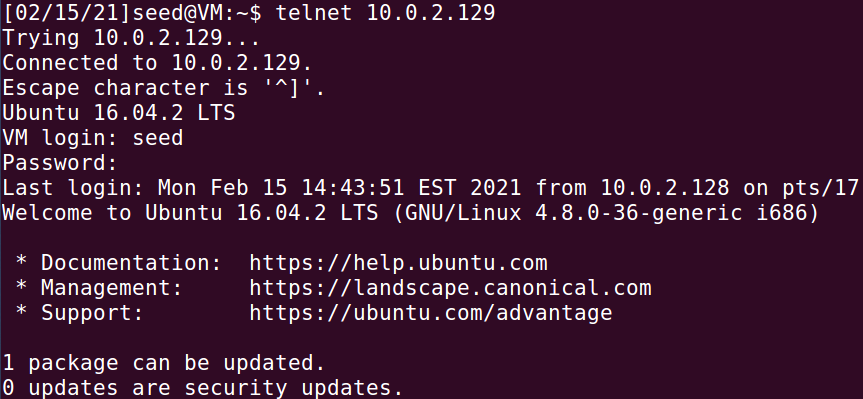
When the Client tries to establish another Telnet connection with the Server, it is unable to do so:



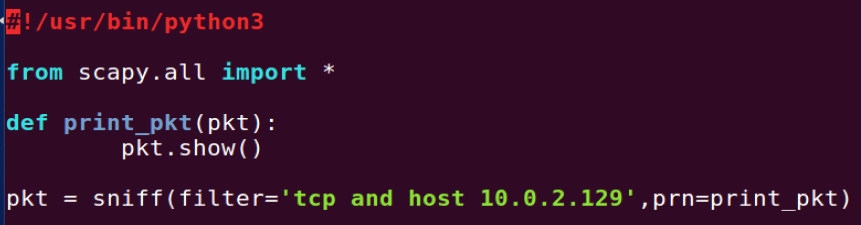
This shows that the TCP RST Attack is successful.

TCP RST Attack on Telnet using Scapy

The Telnet connection between Client and Server is established first:



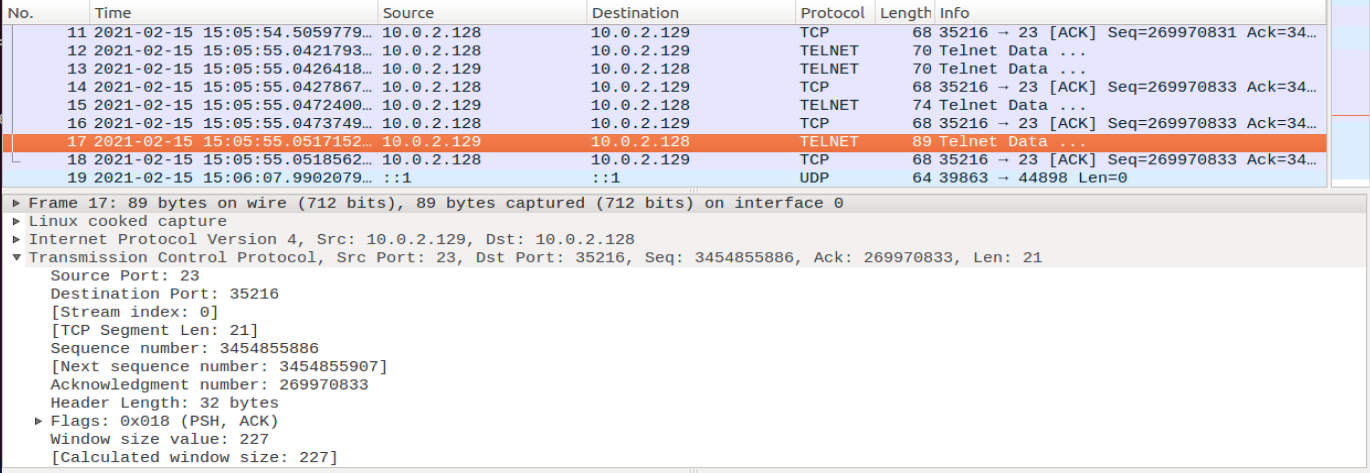
The following sniffer.py code is run on the Attacker machine to sniff all packets in the LAN so that it can sniff the Telnet packets between the Client and the Server:



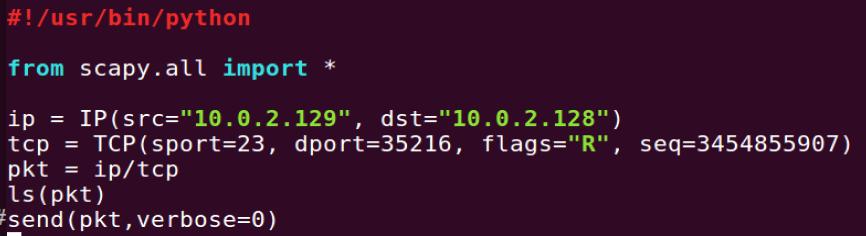
A simple command is run on the Telnet connection to allow some Telnet packets to get captured:



Wireshark is opened to view the details of the packets captured:



From here, we can determine the source port number (23), the destination port number (35216) and the next sequence number (3454855907). We edit the tcp\_rst\_attack\_telnet.py code accordingly:



and run the code to carry out the TCP RST Attack.

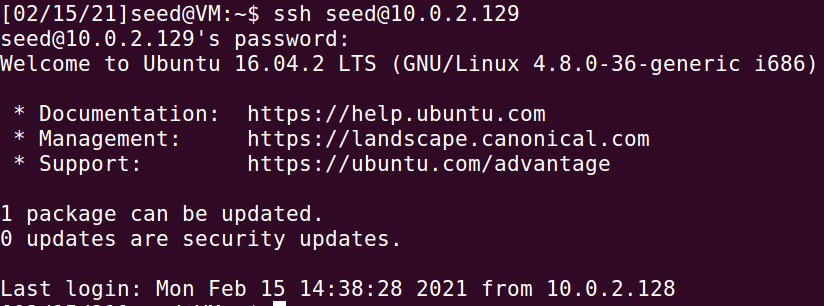
Shortly after this, the Telnet connection is broken as shown on the Client:



This shows that the TCP RST Attack is successful.

TCP RST Attack on SSH using Netwox

The SSH connection between Client and Server is established first:

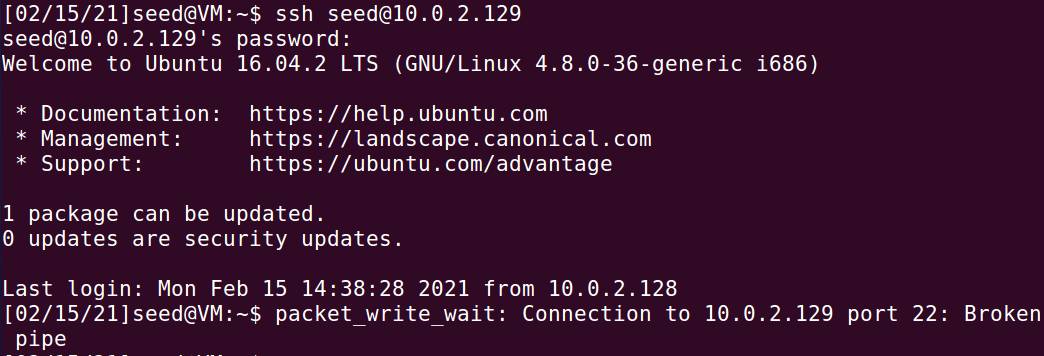


Following this, the Netwox command is run on the Attacker machine to carry out the TCP RST Attack:

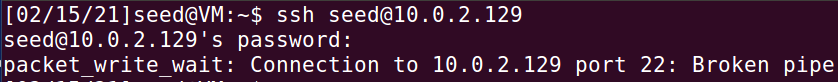


This filter is applied to filter out packets from TCP sessions involving the Server (10.0.2.129).

Shortly after this, the SSH connection is broken as shown on the Client:



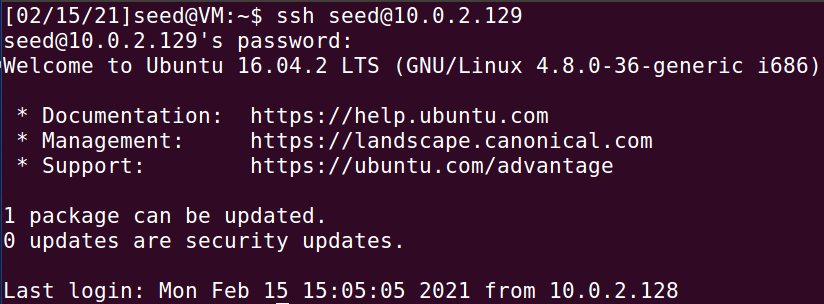
When the Client tries to establish another SSH connection with the Server, it is unable to do so:



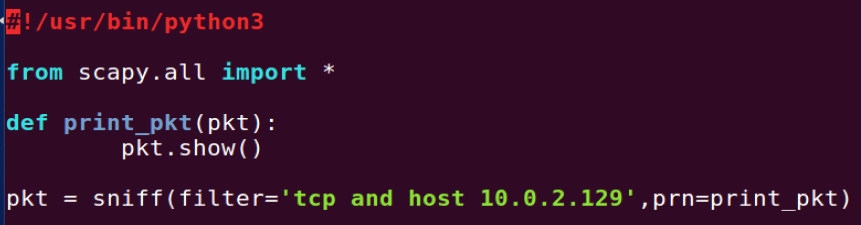
This shows that the TCP RST Attack is successful.

TCP RST Attack on SSH using Scapy

The SSH connection between Client and Server is established first:



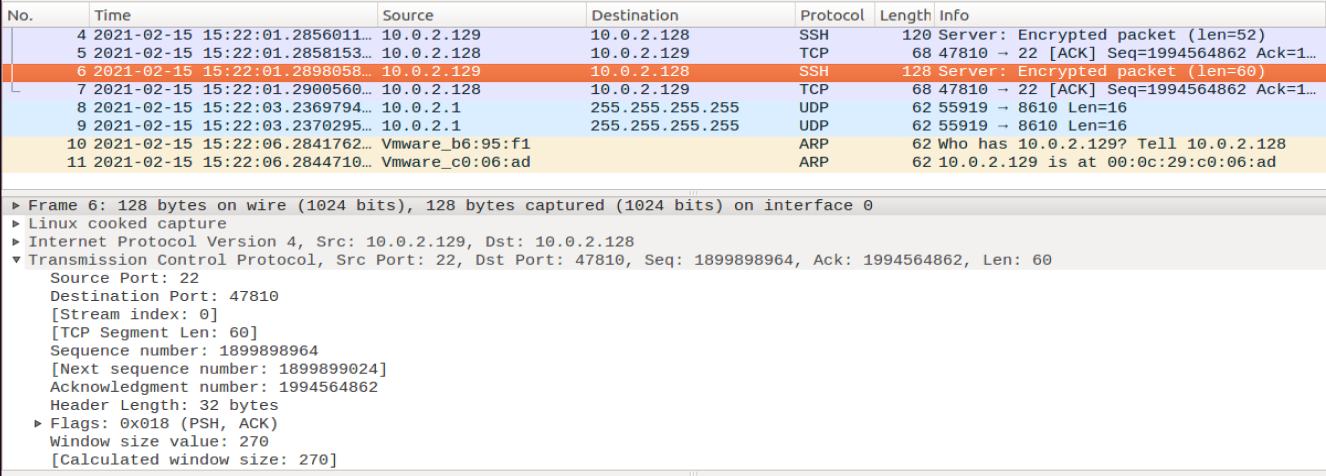
The following sniffer.py code is run on the Attacker machine to sniff all packets in the LAN so that it can sniff the SSH packets between the Client and the Server:



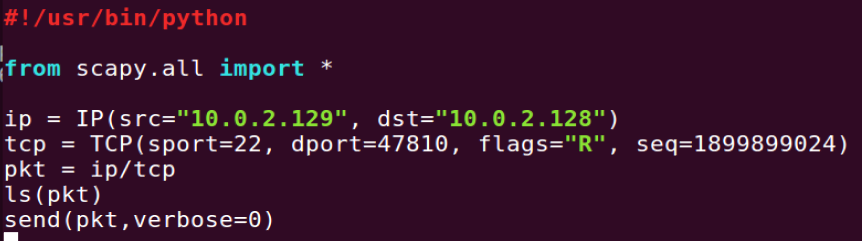
A simple command is run on the Telnet connection to allow some SSH packets to get captured:



Wireshark is opened to view the details of the packets captured:



From here, we can determine the source port number (22), the destination port number (47810) and the next sequence number (1899899024). We edit the tcp\_rst\_attack\_ssh.py code accordingly:



and run the code to carry out the TCP RST Attack.

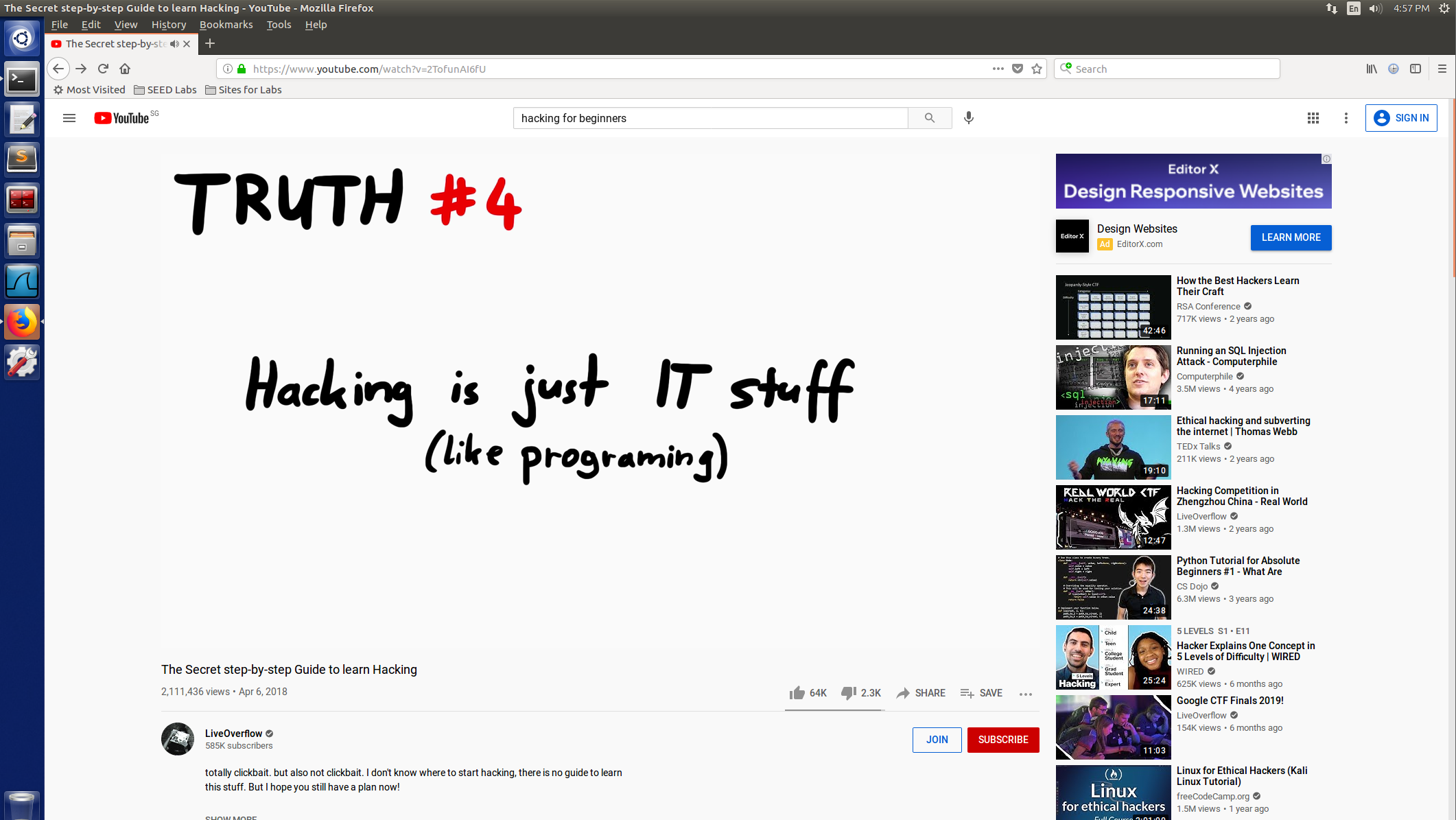
Shortly after this, the Telnet connection is broken as shown on the Client:



This shows that the TCP RST Attack is successful.

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

Using the VM with IP address 10.0.2.129 as the victim machine, we open a browser on the machine and play a random video on youtube:

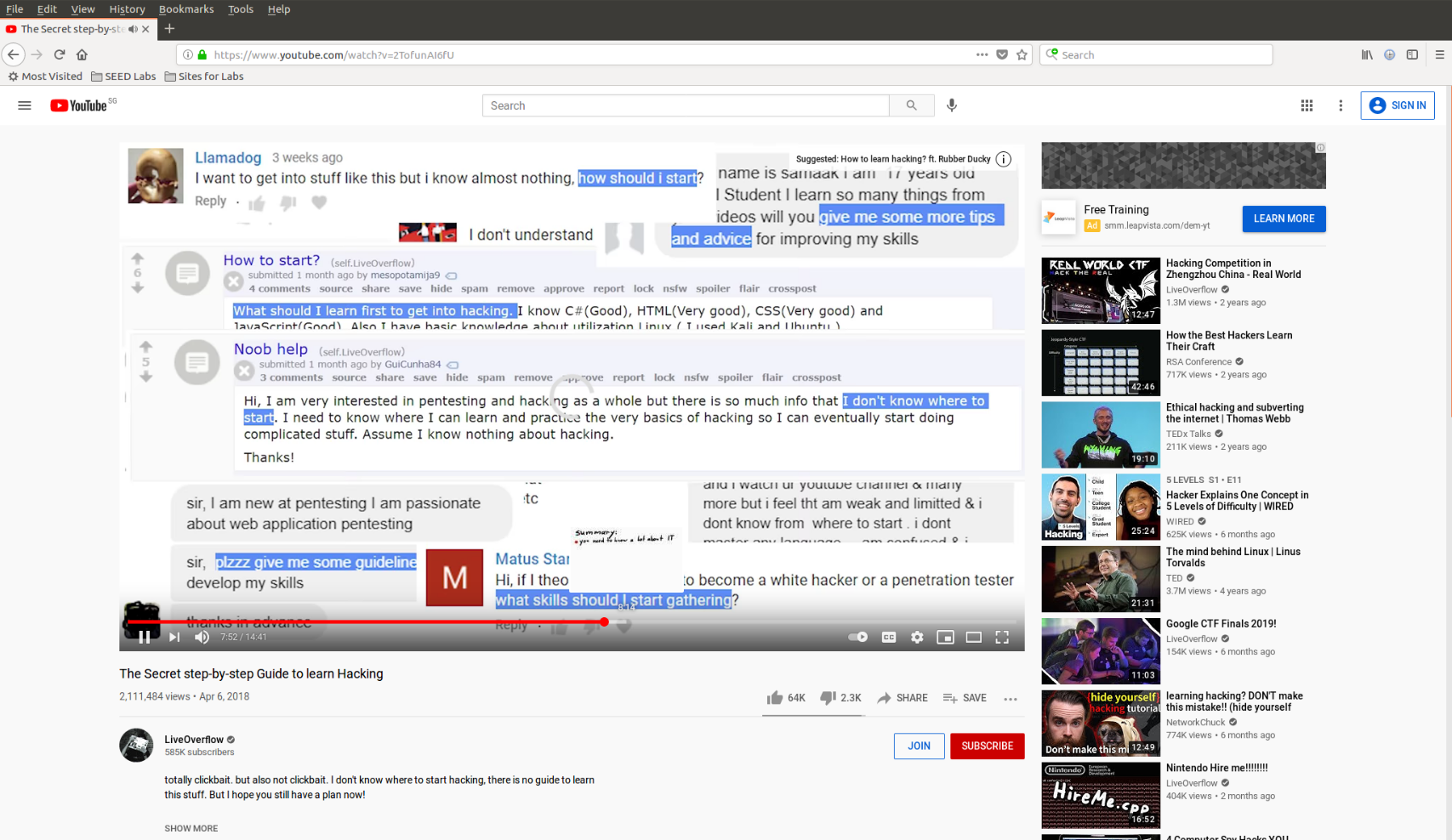


Following this, the Netwox command is run on the Attacker machine to carry out the TCP RST Attack:

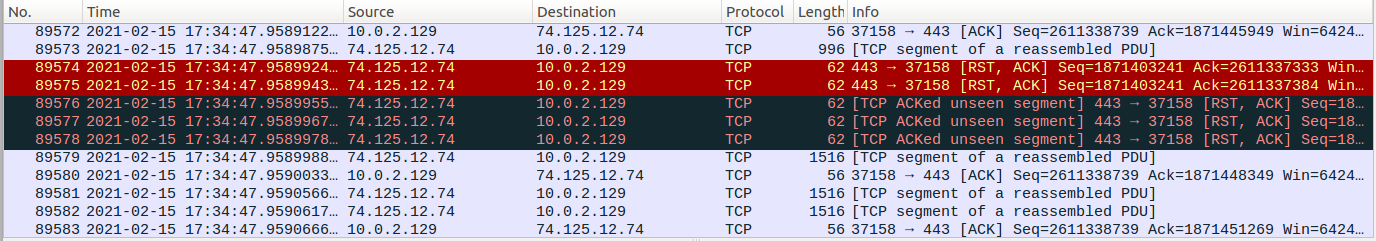


This filter is applied to filter out packets from TCP sessions involving the victim machine (10.0.2.129) as the source.

When this is executed, every time the victim clicks at a timing where the video is not buffered, the video buffers (internet connection has slowed drastically):



The packets captured on Wireshark also shows RST packets being sent to the victim machine to break the TCP connections between the victim machine and the video streaming web site:

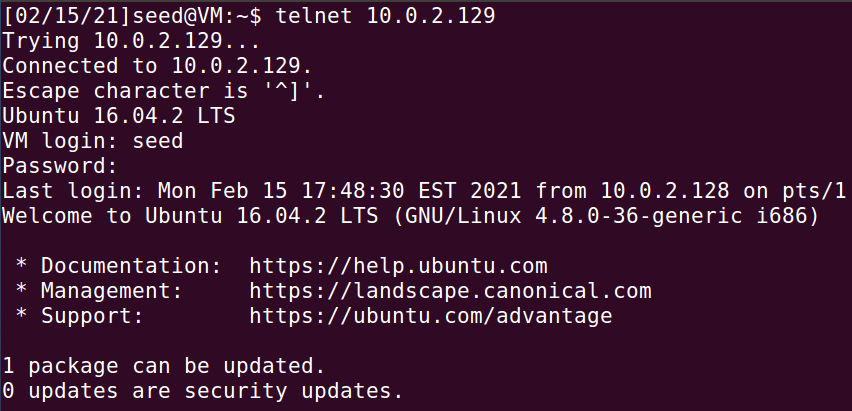


This shows that the TCP RST Attack is successful.

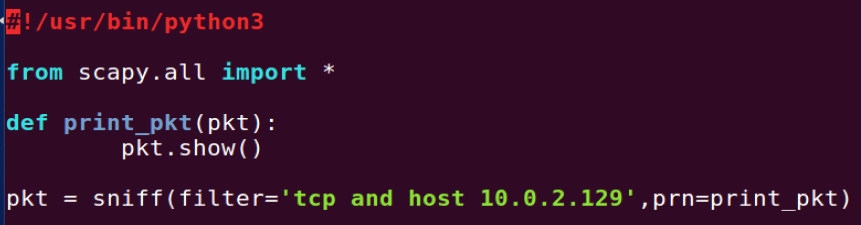
## Task 4: TCP Session Hijacking

TCP Session Hijacking using Netwox

The Telnet connection between Client and Server is established first:



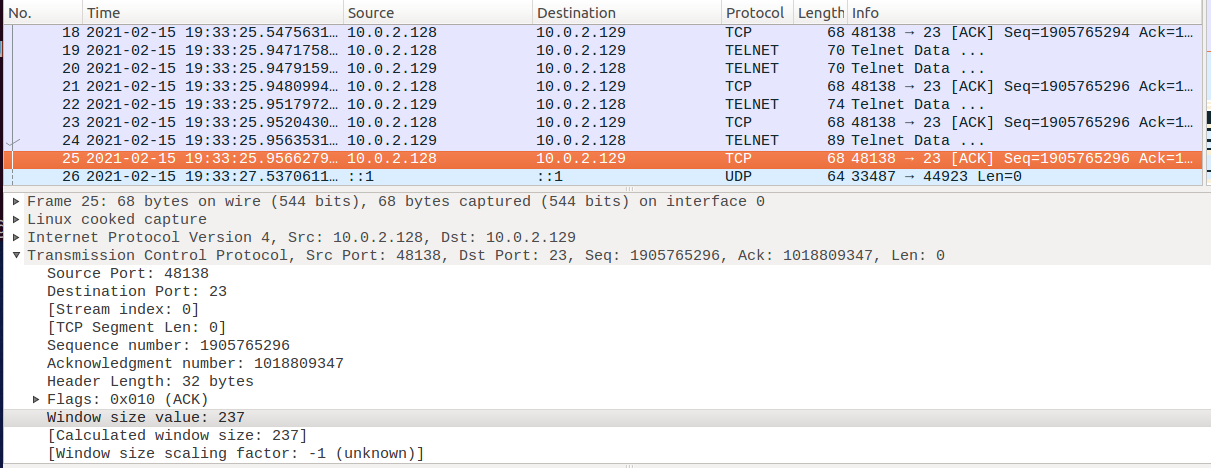
The following sniffer.py code is run on the Attacker machine to sniff all packets in the LAN so that it can sniff the SSH packets between the Client and the Server:



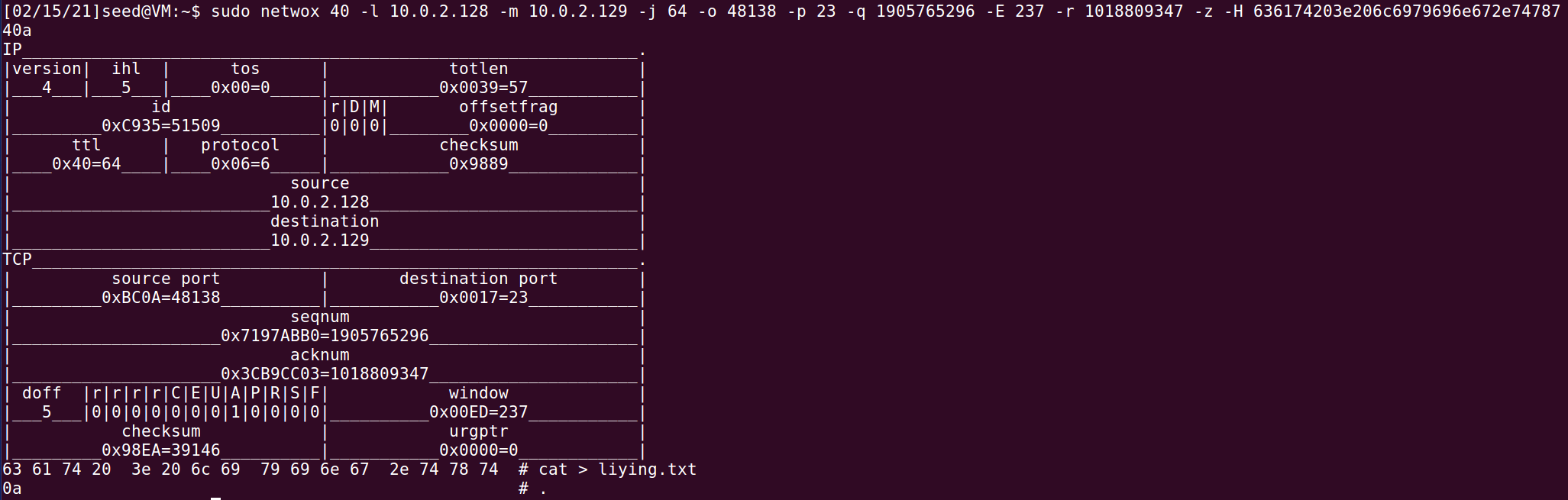
A simple command is run on the Telnet connection to allow some SSH packets to get captured:



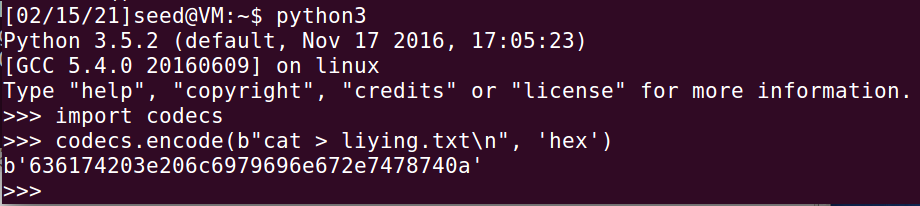
Wireshark is opened to view the details of the packets captured:



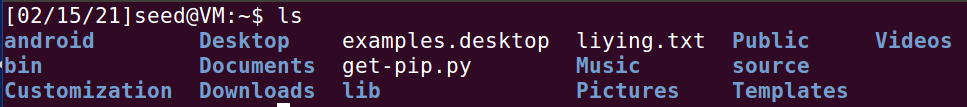
From here, we can determine the source port number (48138), the destination port number (23), the next sequence number (1905765296), the TCP window size (237) and the acknowledgement number (1018809347). We edit the Netwox command accordingly:



The TCP data (‘636174203e206c6979696e672e7478740a’) is the hex version of the command “cat > liying.txt” so that when the Netwox command is run, it creates a file ‘liying.txt’ on the Telnet Server:



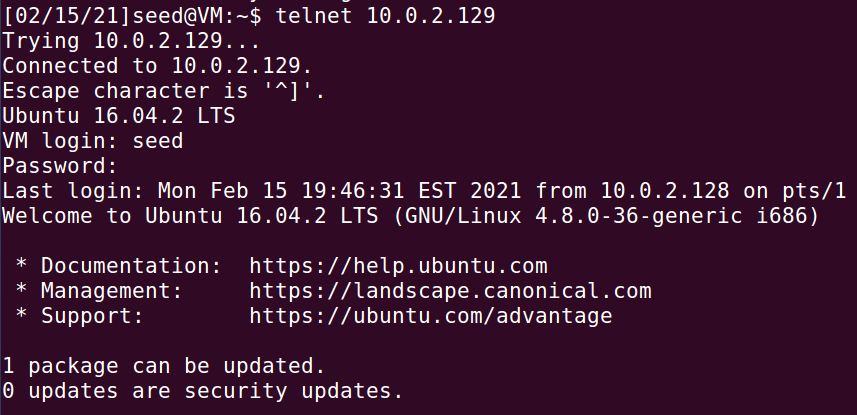
The Netwox command is then run on the Attacker machine to carry out the TCP hijacking attack. This creates the file liying.txt in the home folder of the victim machine:



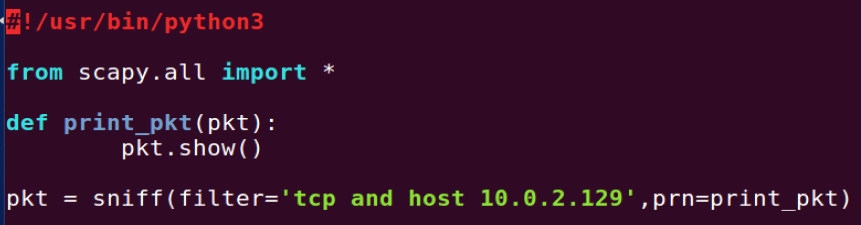
This shows that the TCP Session Hijacking Attack is successful.

TCP Session Hijacking using Scapy

The Telnet connection between Client and Server is established first:



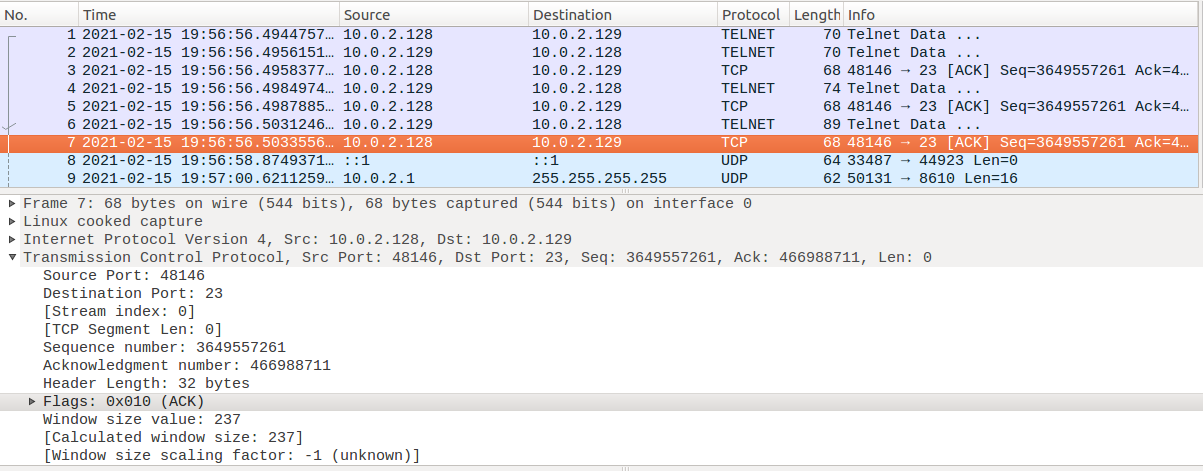
The following sniffer.py code is run on the Attacker machine to sniff all packets in the LAN so that it can sniff the SSH packets between the Client and the Server:



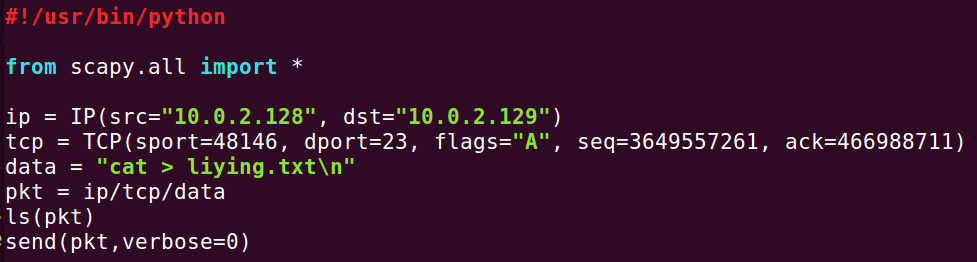
A simple command is run on the Telnet connection to allow some SSH packets to get captured:



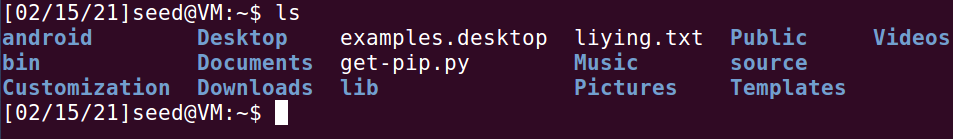
Wireshark is opened to view the details of the packets captured:



From here, we can determine the source port number (48146), the destination port number (23), the next sequence number (3649557261), the TCP window size (237) and the acknowledgement number (466988711). We edit the hijack.py code accordingly:



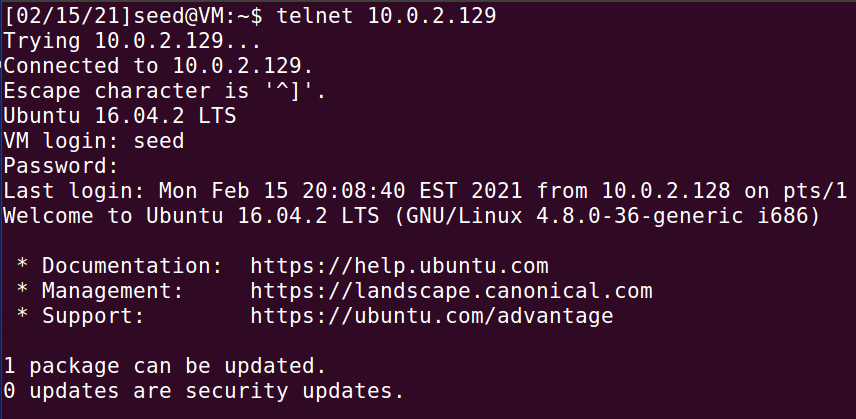
The code is then run on the Attacker machine to carry out the TCP hijacking attack. This creates the file liying.txt in the home folder of the victim machine:



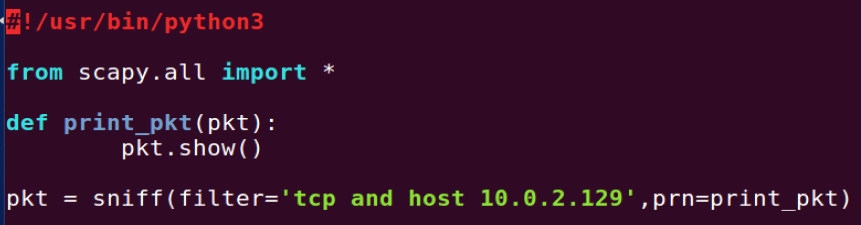
This shows that the TCP Session Hijacking Attack is successful.

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

The Telnet connection between Client and Server is established first:



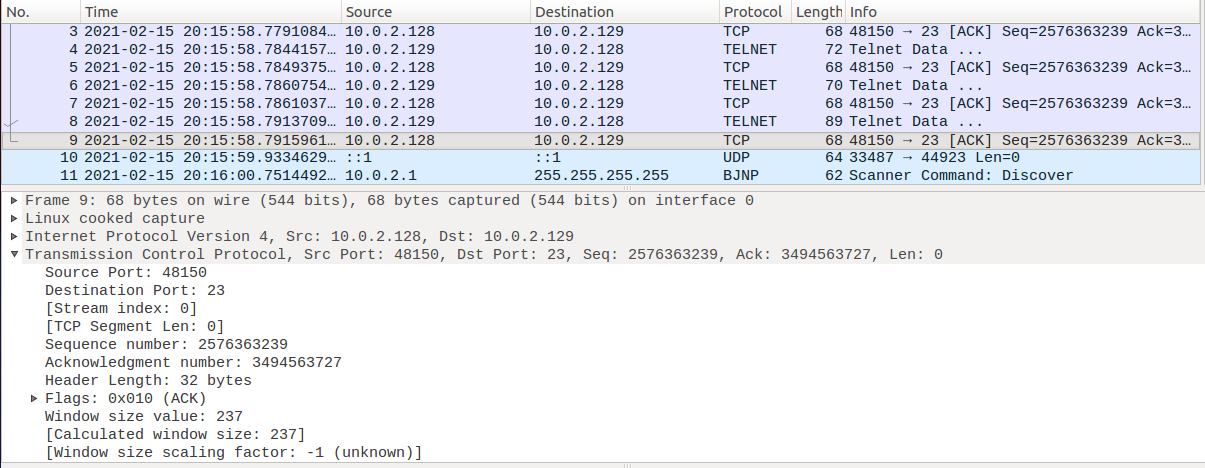
The following sniffer.py code is run on the Attacker machine to sniff all packets in the LAN so that it can sniff the SSH packets between the Client and the Server:



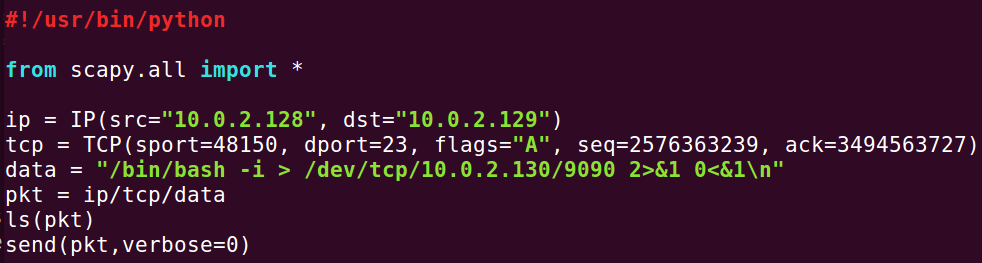
A simple command is run on the Telnet connection to allow some SSH packets to get captured:



Wireshark is opened to view the details of the packets captured:



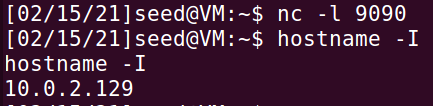
From here, we can determine the source port number (48150), the destination port number (23), the next sequence number (2576363239), the TCP window size (237) and the acknowledgement number (3493453727). We edit the reverse\_shell.py code accordingly:



We open a netcat listener on port 9090 to listen for the reverse shell command:



The code is then run on the Attacker machine to carry out the TCP hijacking attack. The victim machine connects to the netcat listening on the Attacker machine and a connection is established, spawning the reverse shell:



This shows that using the TCP Session Hijacking Attack to create a reverse shell is successful.