**TCP Attacks**

Chawin Metah

Department of Computer Science, Purdue University Fort Wayne

ACS 545000: Cryptography and Network Security

Lab 4

Dr. Zesheng Chen

February 9, 2022

**Task 1: SYN Flooding Attack**

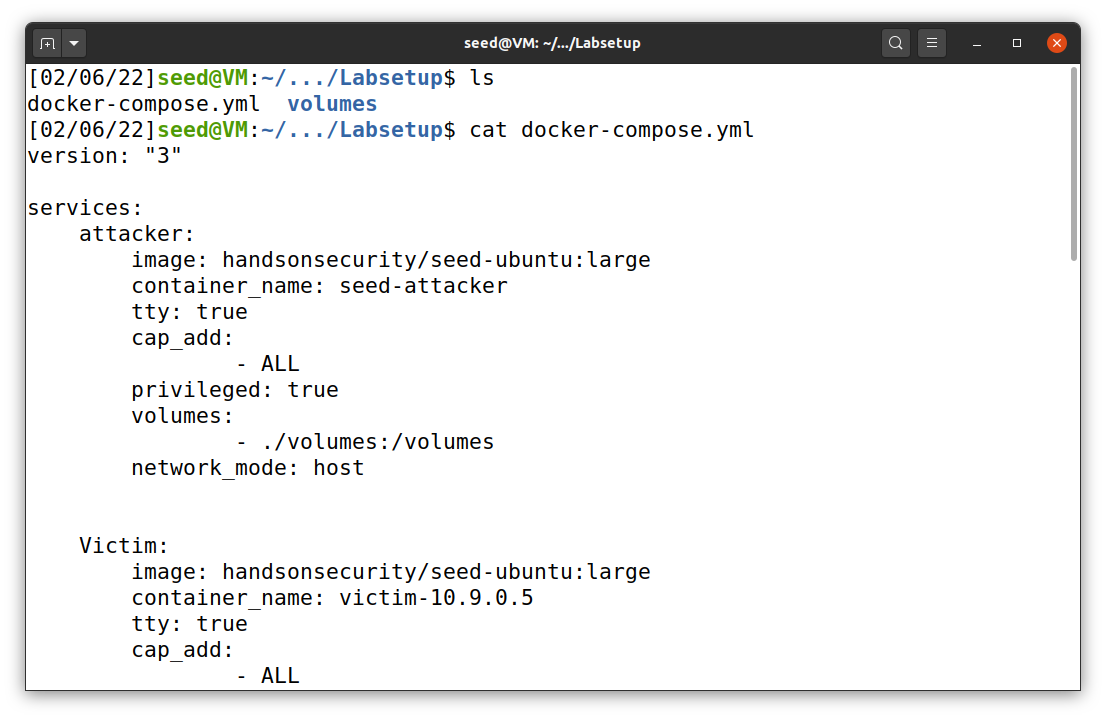
Similar to the previous labs, we download and unzip Labsetup.zip, which contains docker-compose.yml and volume directory (Figure 1). Then, we use the command below to initiate network, as shown in Figure 2. This lab, we have four users: attacker (10.9.0.1), victim (10.9.0.5), user1 (10.9.0.6), and user2 (10.9.0.7).

dcbuild

dcup

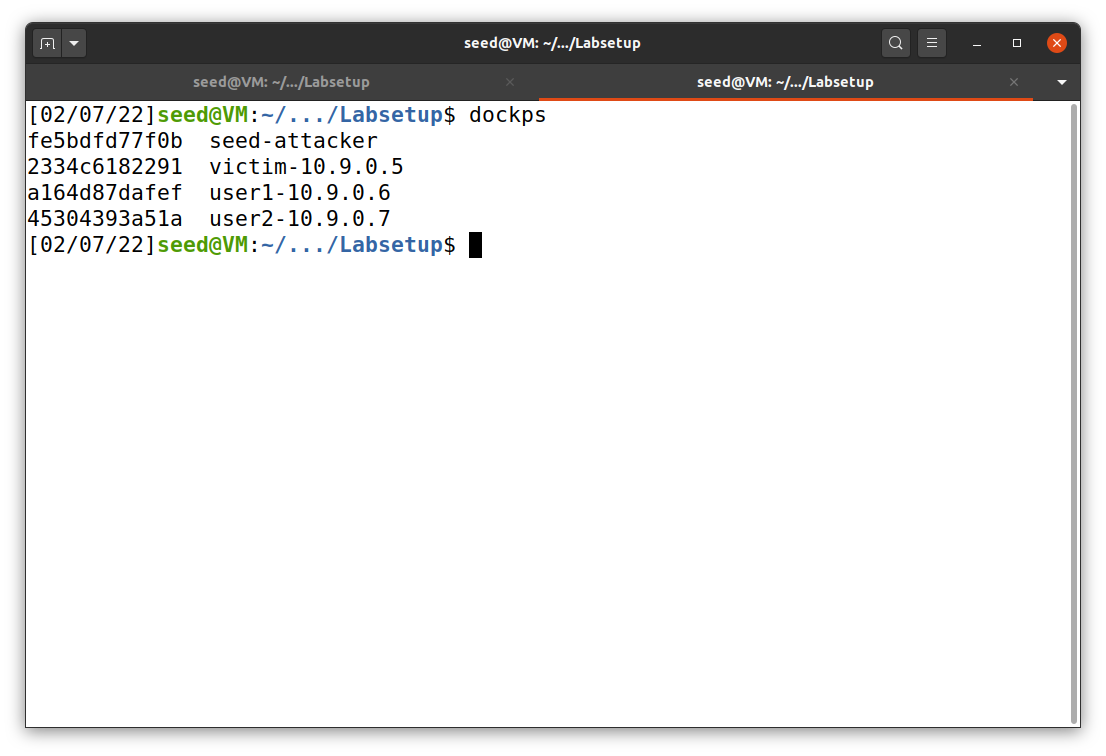
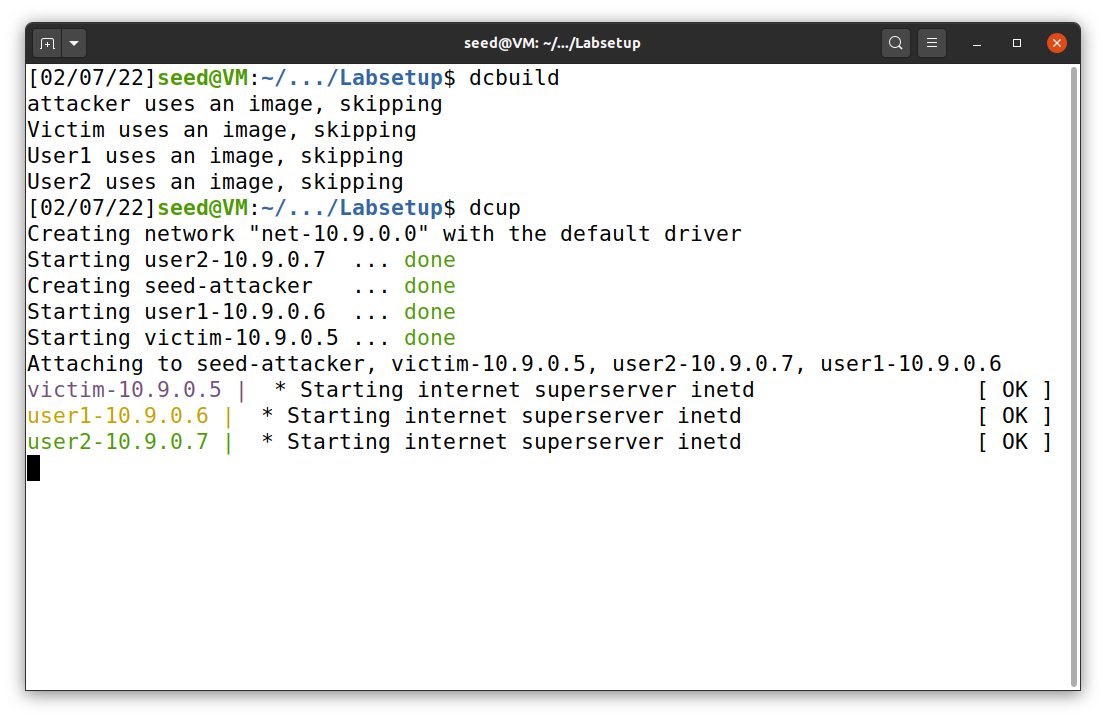
**Figure 1**

*Labsetup.zip*

**

**Figure 2**

*Network initiation*

**

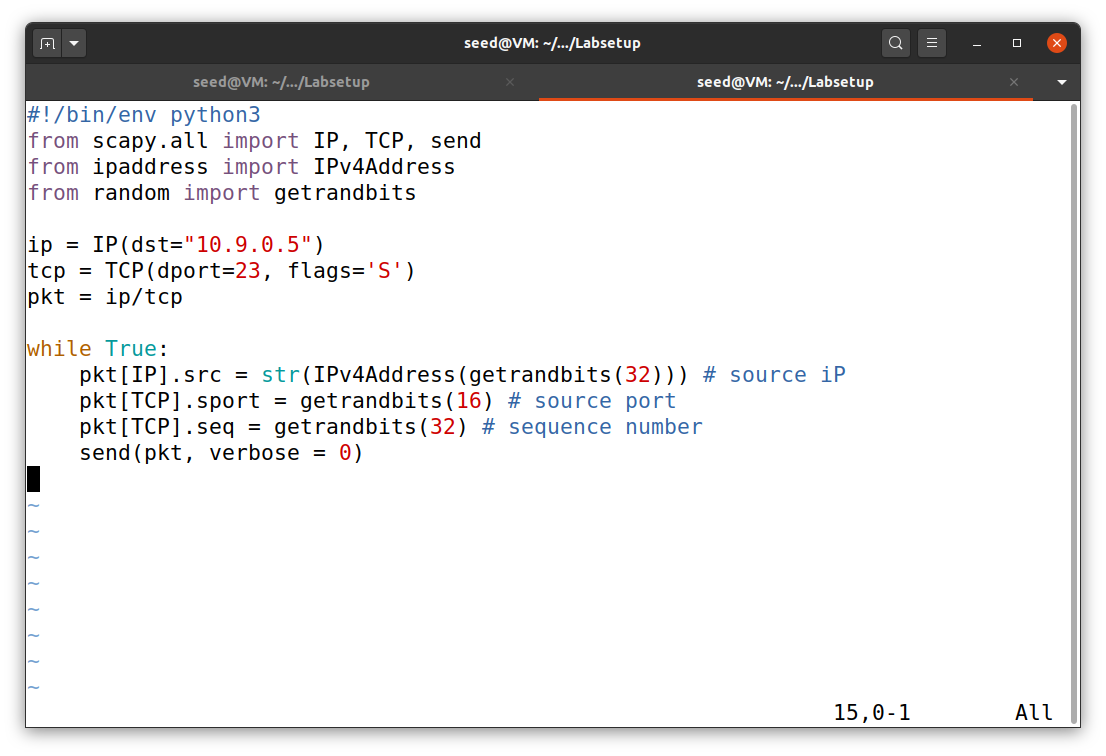
***Task 1.1: Launching the Attack Using Python***

To begin, I wrote a program that is able to flood a server with SYN flag with random source IPs (Du, n.d.), as shown in Figure 3. This is a python program that is set to attack victim server (dst 10.9.0.5) on port 23 (telnet). After we save the file, we have to set it as executable file with the command below (Figure 4). The file will turn green if it is executable. This will apply to every file that we will run in this lab.

chmod a+x synflood.py

**Figure 3**

*synflood.py*

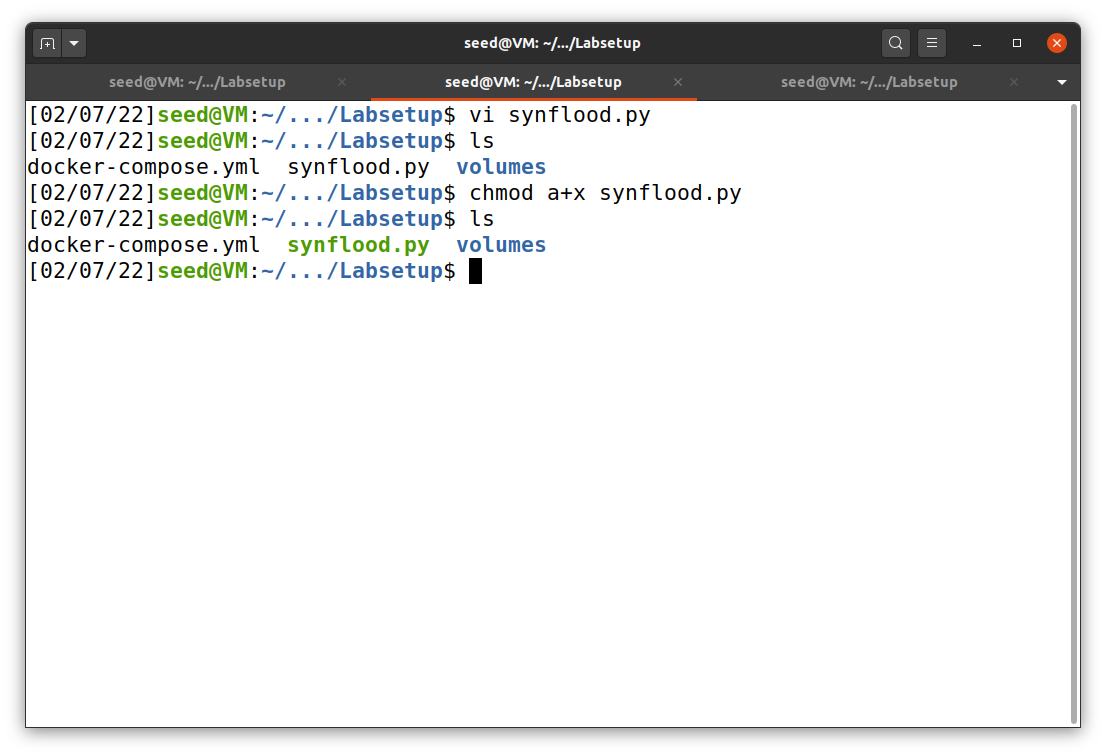


TCP port 23 (telnet) with SYN flag

Victim’s IP

**Figure 4**

*Executable synflood.py*



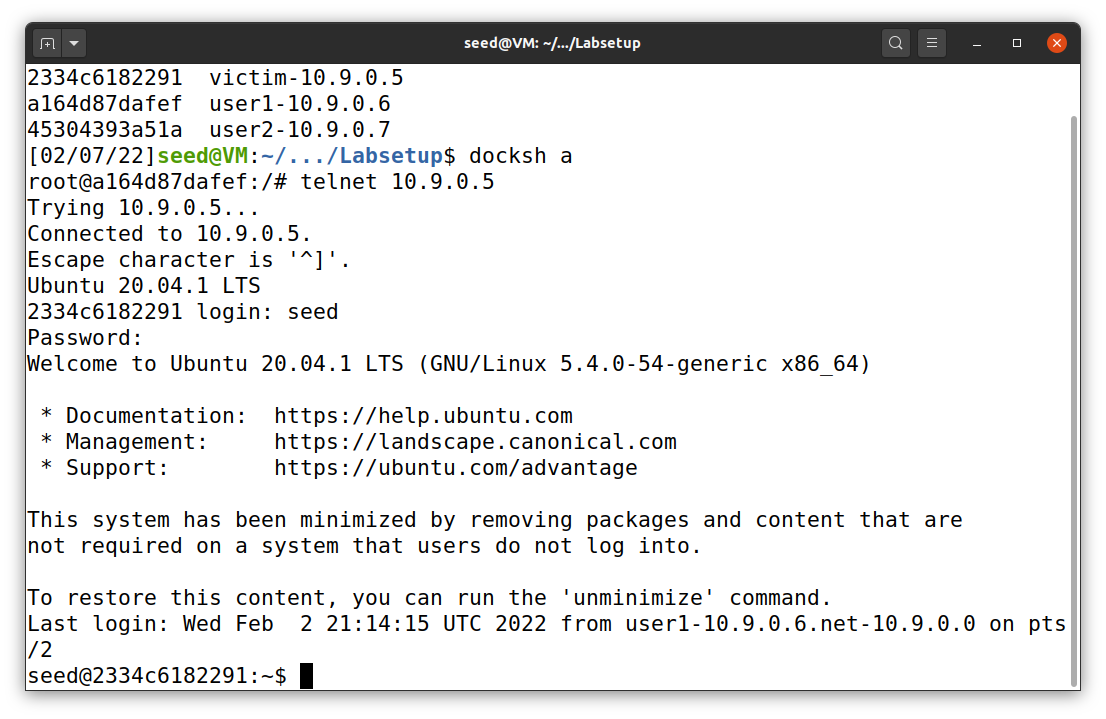
Next, I ran synflood.py file for a few minutes. A server can get rid of half-open connections much faster than the python program can generate random source IPs and send it to server with SYN flag. Therefore, we have to let the program run for some period in order to flood any server. The result from first run with default settings and a single instance is shown in Figure 5, using the command below to run synflood.py. The result shows that user1 (10.9.0.6) can still telnet the victim (10.9.0.5)

sudo python3 synflood.py

**Figure 5**

*Utilization of synflood.py on port 23*

Graphical user interface, text

Description automatically generated

telnet was successful

user1 (10.9.0.6) telnet victim (10.9.0.5)

Terminal 1

User 1

The result above indicates an unsuccessful attempt of SYN flooding attack. So, we have to perform the attack with a few more instances. Before doing that, on Ubuntu 20.04, we have to clear a list of previously connected (trusted) device on victim server with the command below. As shown in Figure 6, the victim can “remember” connected devices and provide special space in the queue for them, which SYN flood attack will no longer effective. This will apply every time before performing an attack in this lab.

ip tcp\_metrics flush

**Figure 6**

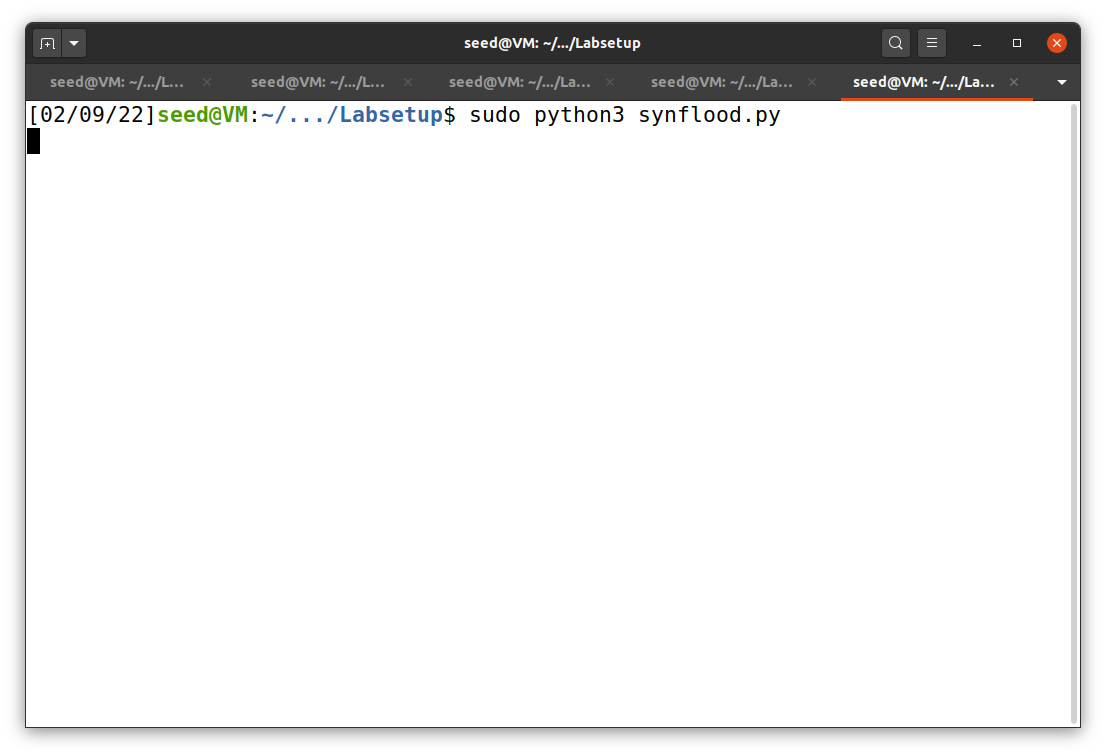
*Network queue flush*

**

After that, we can perform another attack. This time I run synflood.py with 5 instances. I got the result that when I run synflood.py with 2, 3, and 4 instances, user1 can still be able to telnet victim. The result is shown in Figure 7.

**Figure 7**

*SYN flood attack result*

**

Instance 5

Instance 4

Instance 3

Instance 2

Instance 1

User1

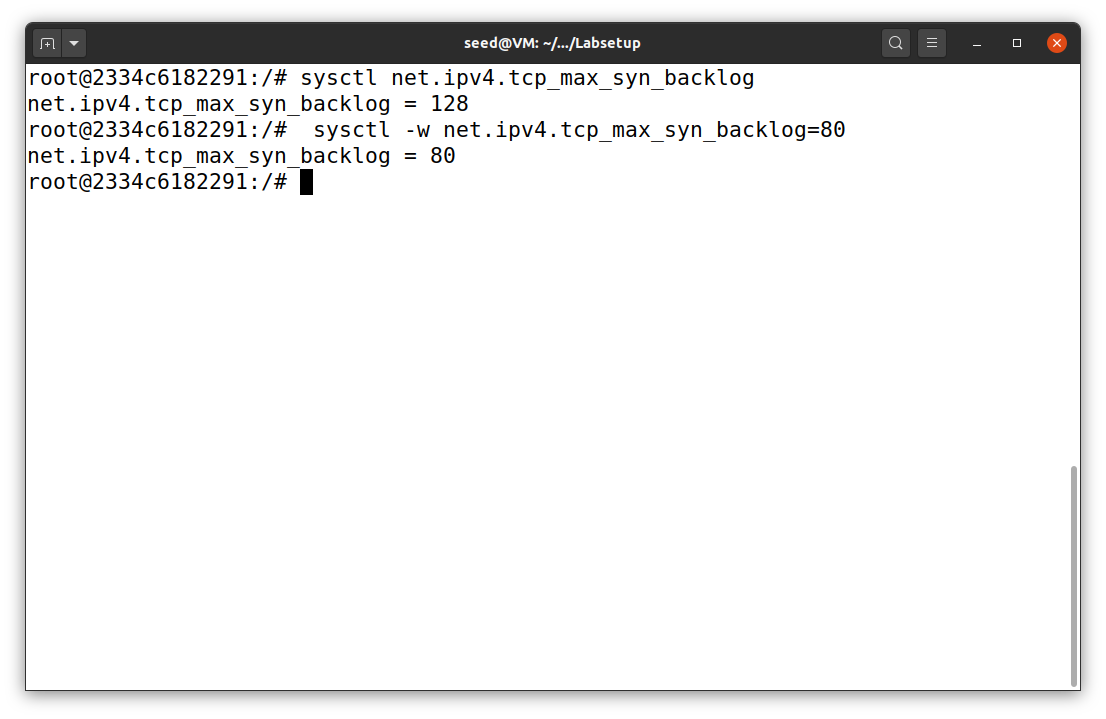
Terminal

Next, I lowered the number of queue of the victim from default (128) to 80 using the command below, as shown in Figure 8.

sysctl -w net.ipv4.tcp\_max\_syn\_backlog=80

**Figure 8**

*Network queue number modification*

**

Default value

victim

Then, I tried SYN flood attack using synflood.py again. This time, only one instance of synflood.py is required to block user1 to telnet victim. The result is not shown because it is similar to User1 in Figure 7.

***Task 1.2: Launch the Attack Using C***

After using synflood.py, I increased queue limit back to 128 and perform SYN flood attack again with synflood.c. The file is provided under the directory of volume in the Labsetup.zip file. I, then, compiled and ran the code for a few minutes with the commands below. Similar to python code, compiled file synflood also have to run in privilege mode (super user). The result in Figure 9 shows that we can perform SYN attack using synflood with just one instance. This concludes that C program runs much faster than python.

gcc -o synflood synflood.c

sudo synflood 10.9.0.5 23

**Figure 9**

*SYN flood attack using synflood.c*

Text

Description automatically generatedText

Description automatically generated

User 1

Cannot connect to victim

SYN flood attack success

Number of queues

Terminal 1

victim

***Task 1.3: Enable the SYN Cookie Countermeasure***

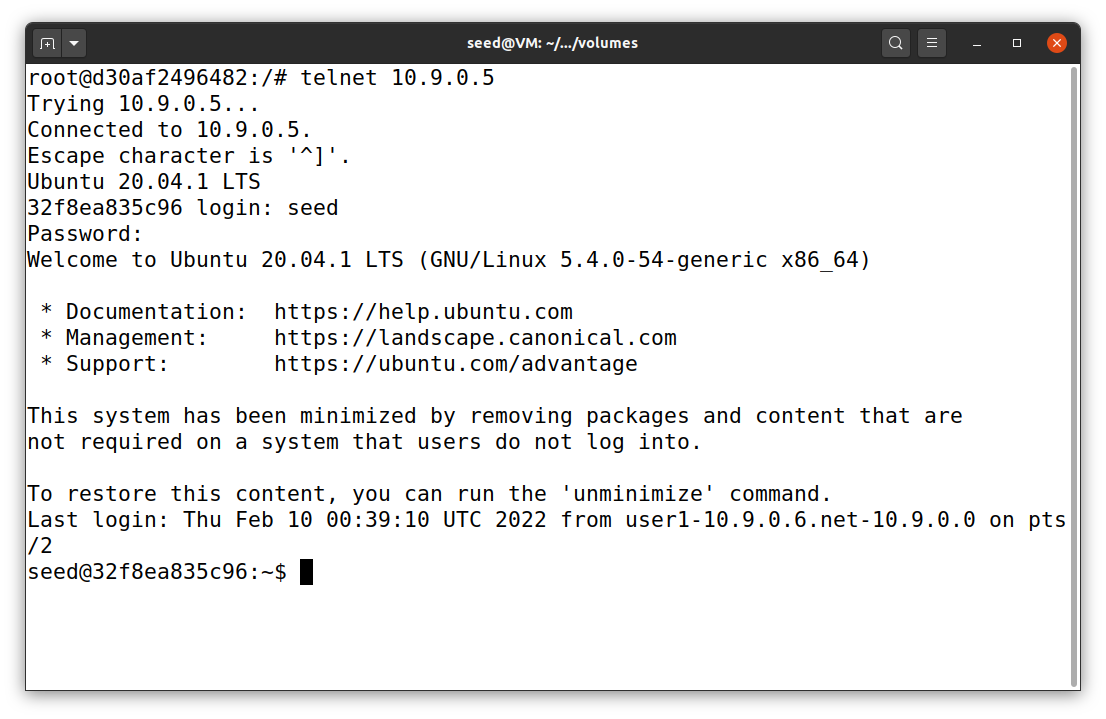
To enable SYN cookie, I used the command below. Then, I performed SYN attack with both 5 instances of synflood.py and synflood in C program. The results in Figures 10 and 11 show that we can easily use telnet from user1 to victim.

sysctl -w net.ipv4.tcp\_syncookies=1

**Figure 10**

*SYN flood attack with cookie using synflood.c*

Graphical user interface, text, application

Description automatically generated

User1 telnet victim

Telnet Success!

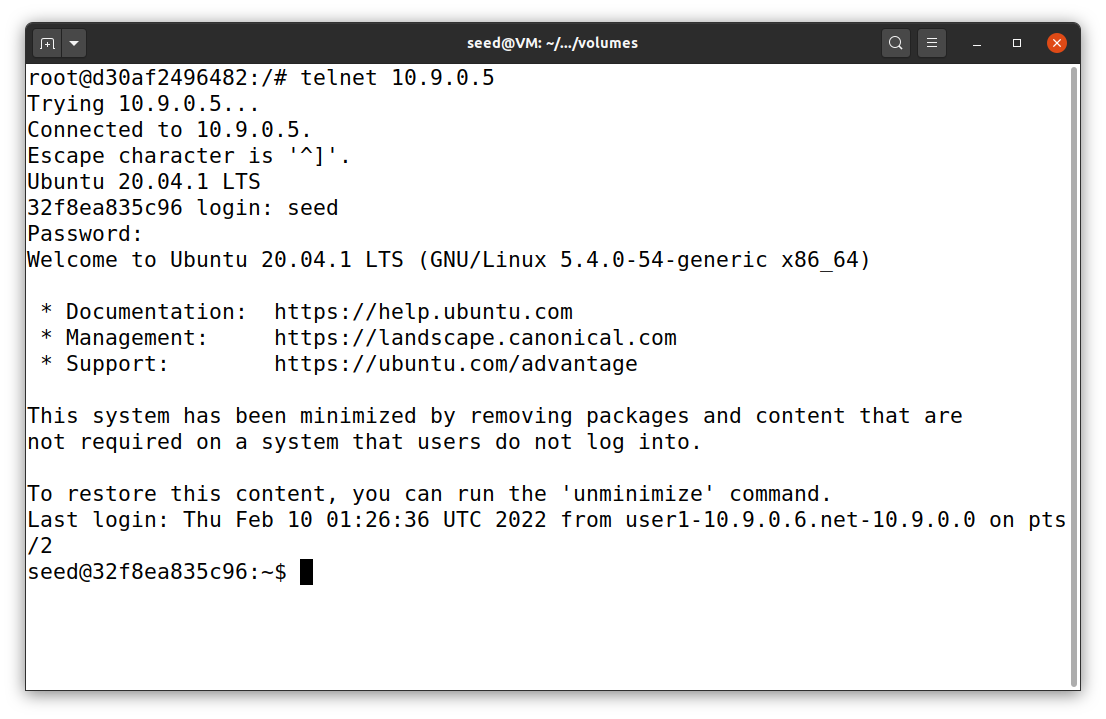
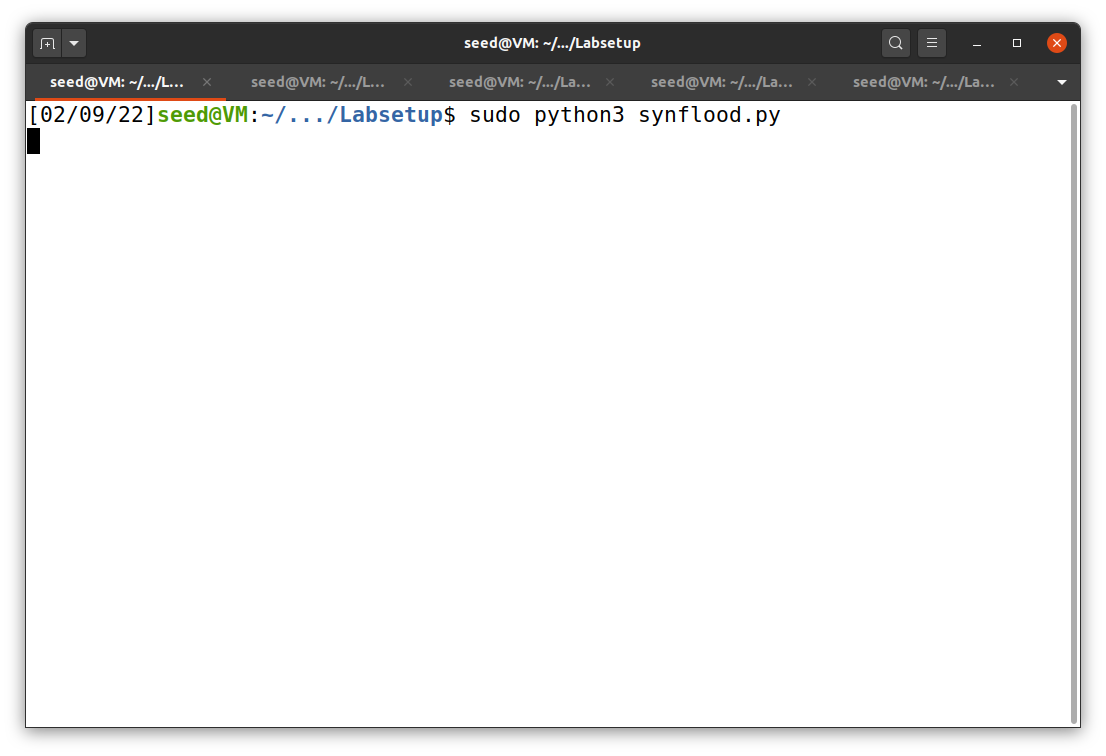
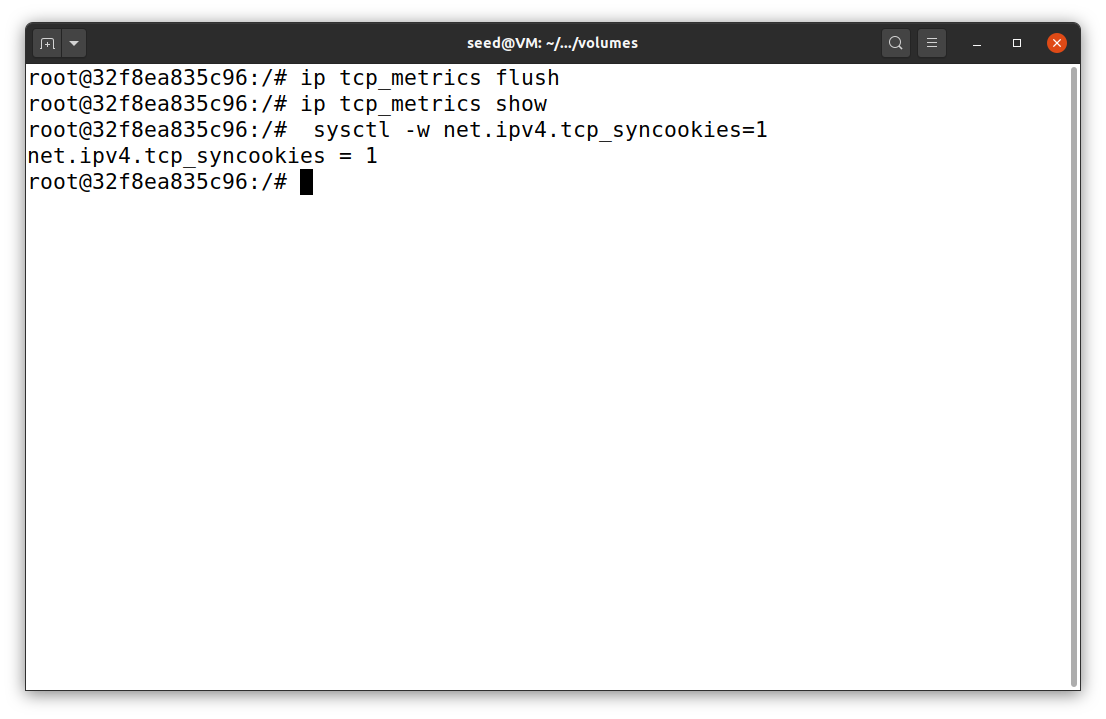
Terminal 1

Victim

User 1

**Figure 11**

*SYN flood attack with cookie using synflood.py*



Telnet Success!

User1 telnet victim

Victim

User 1

Terminal 1

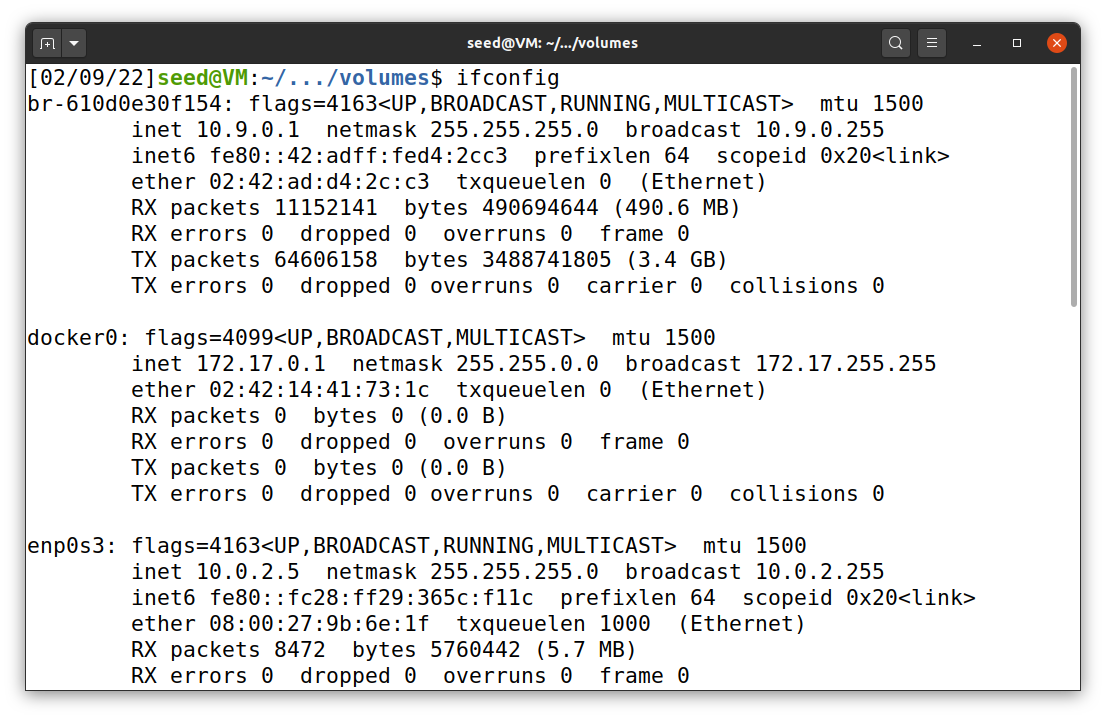
**Task 2: TCP RST Attacks on telnet Connections**

In this task, I used the TCP RST python program to reset a connection between two users. I chose the connection on port 23 (telnet) from user 1 (10.9.0.6) to victim (10.9.0.5). Before we can create python file, we have to know our interface ID and form tcpdump first. To get the interface ID, I used ifconfig, as shown in Figure 12. After that, I start to use telnet from user1 to victim (Figure 13) and used tcpdump to collect connection data with the command below (Figure 14).

sudo tcpdump -i br-610d0e30f154 -w /tmp/packets host 10.9.0.5

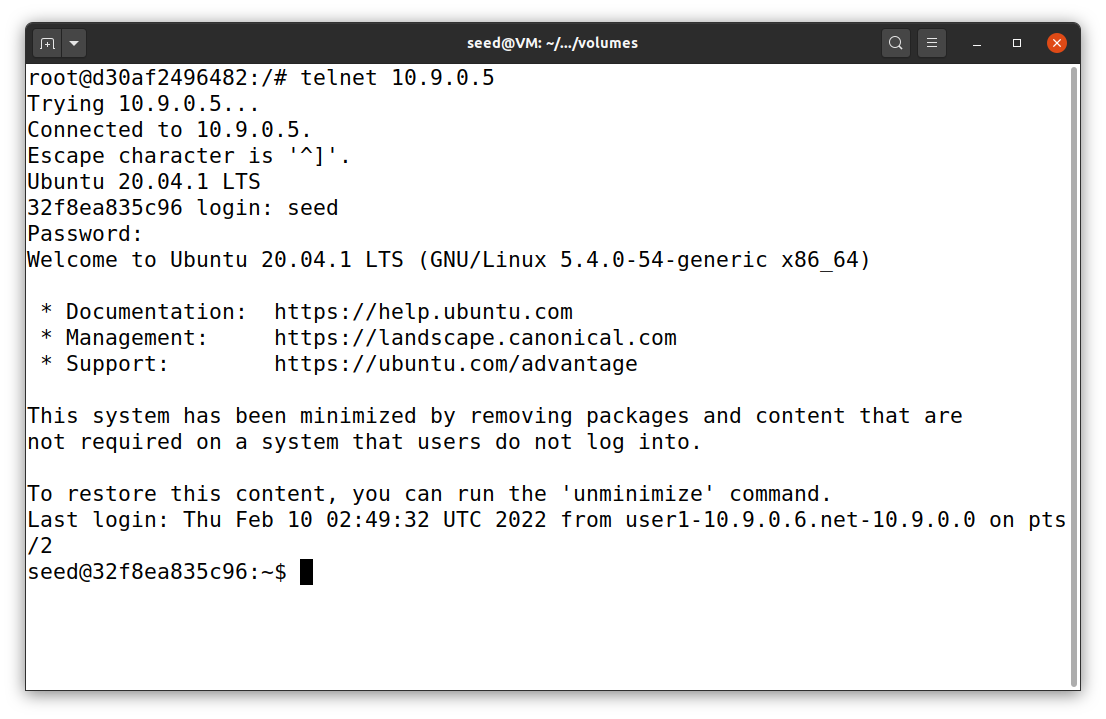
**Figure 12**

*Interface ID with ifconfig*

**

**Figure 13**

*User1 telnet victim*



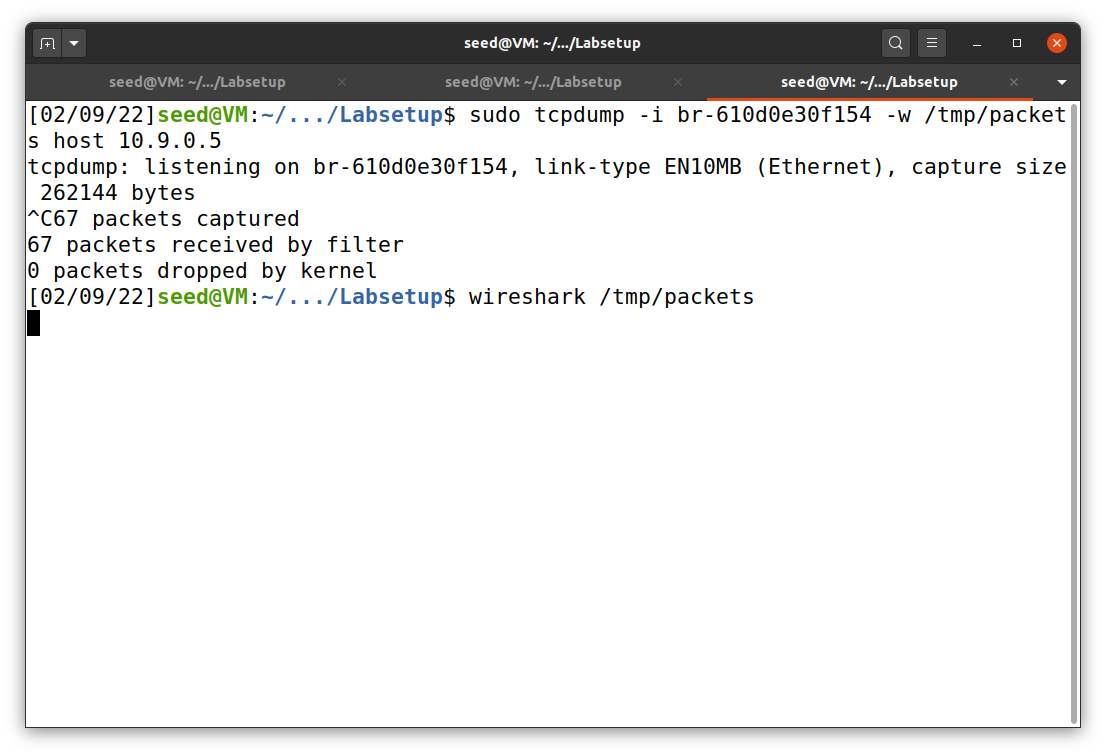
Telnet Success!

User1 telnet victim

User 1

**Figure 14**

*Tcpdump and wireshark*

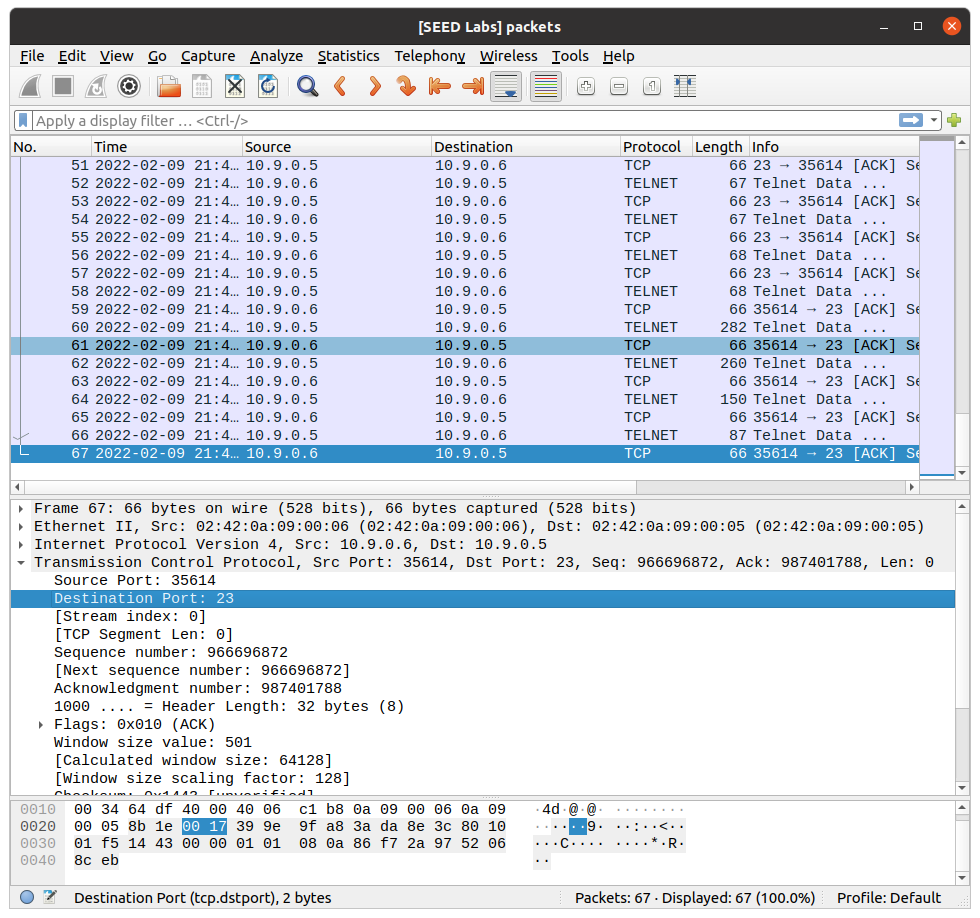


In Figure 14, tcpdump command is used. Then, I open Wireshark using the command below.

wireshark /tmp/packets

**Figure 15**

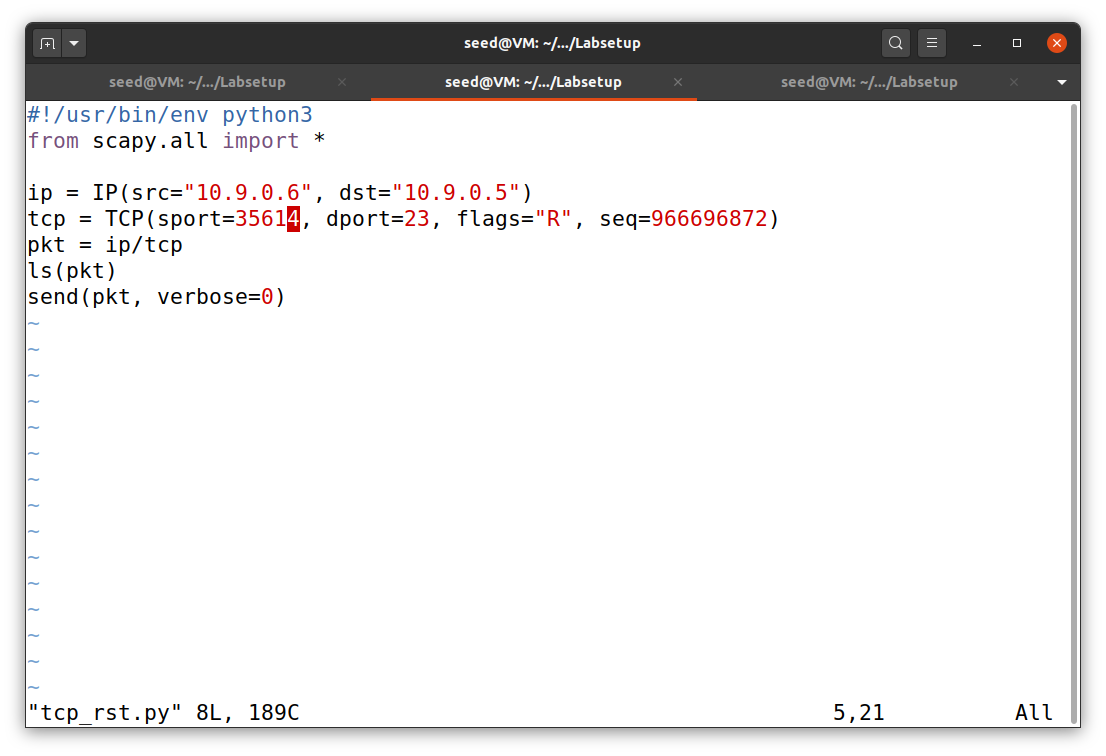
*Wireshark for task 2*



After Wireshrak is opened, we will see the seen as shown in Figure 15 above. We have to scroll down to find the last connection and get two information: source port and sequence number. These data are complete, and we can create tcp\_rst.py. The code of the file is shown in Figure 16.

**Figure 16**

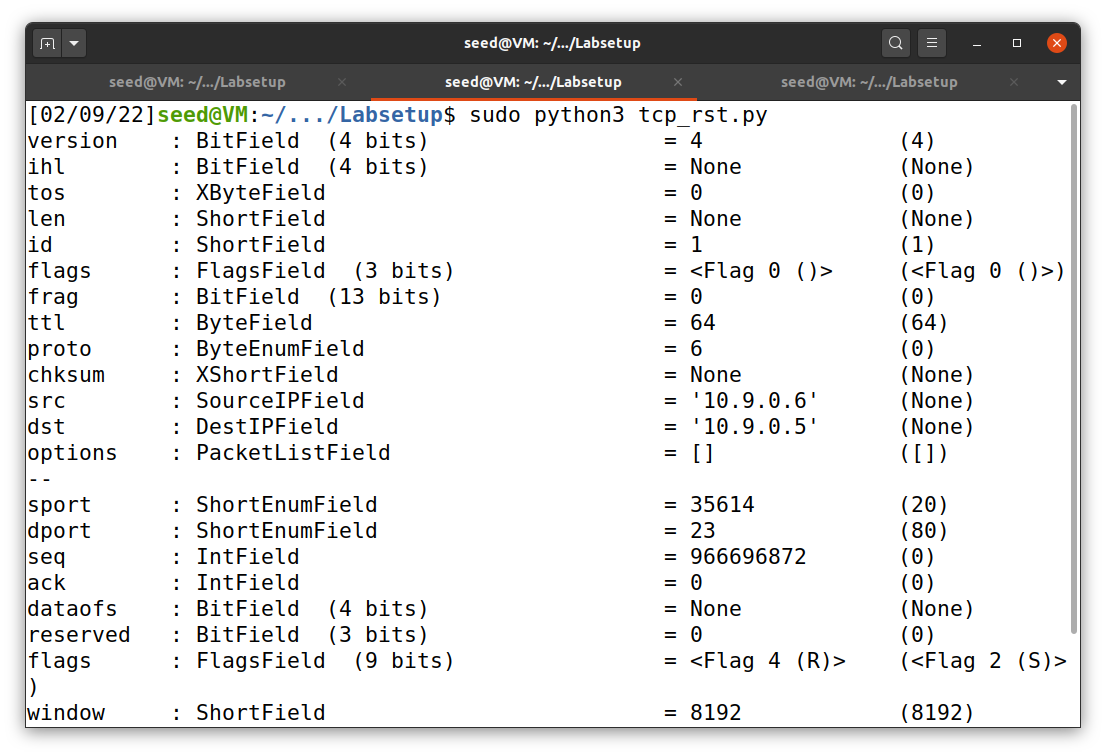
*tcp\_rst.py*



Then, we can run the program and get the result as shown in Figure 17. The program will terminate the connection on TCP port 23 between host user1 and victim. We can observe that from our source server (user1 10.9.0.6), as shown in Figure 18.

**Figure 17**

*Execution of tcp\_rst.py*



**Figure 18**

*Disconnection of telnet between user1 and victim*

Text

Description automatically generated

**Task 3: TCP Session Hijacking**

TCP session hijacking have a procedure close to TCP RST attack. Before we can continue, we have to create a file in victim server (Figure 19) and remove /tmp/packets of previous connection using the command below (Figure 20).

sudo rm /tmp/packets

**Figure 20**

*Removal of /tmp/packets*



**Figure 21**

*Removal of /tmp/packets*

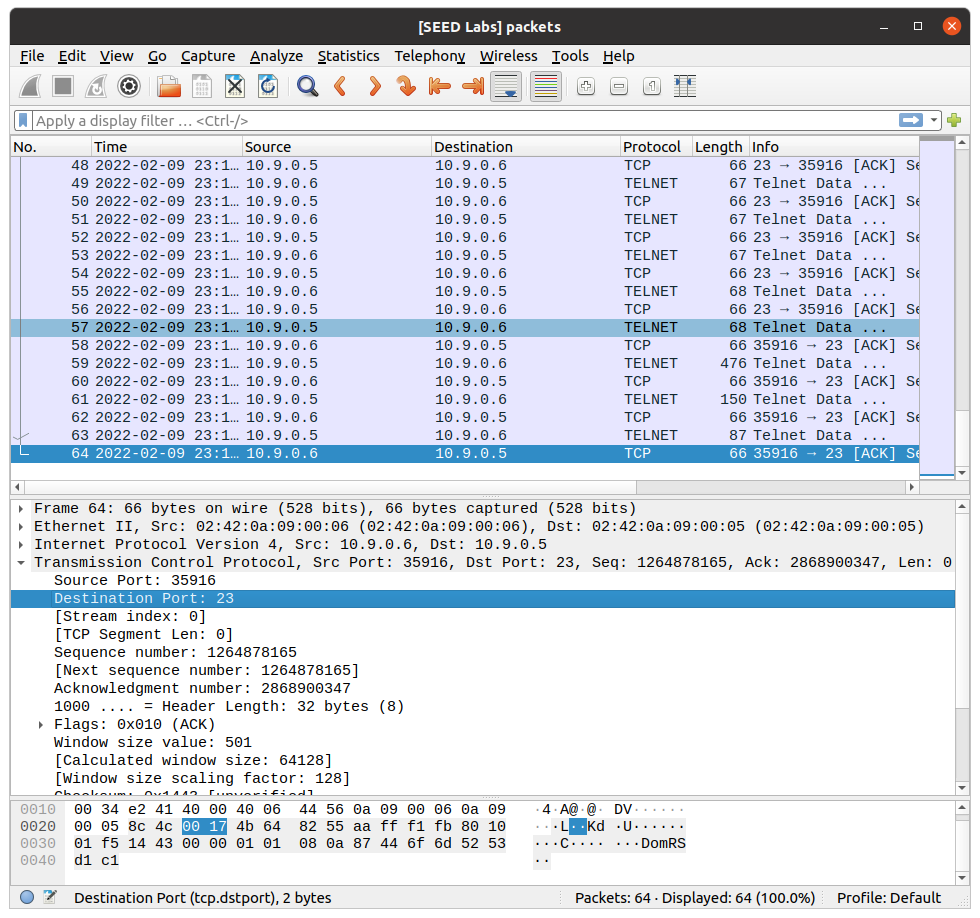
Text

Description automatically generated

After that, I used telnet from user1 to victim and tcpdump as shown in the previous task. Then, I opened the connection data with Wireshark, as shown in Figure 22.

**Figure 22**

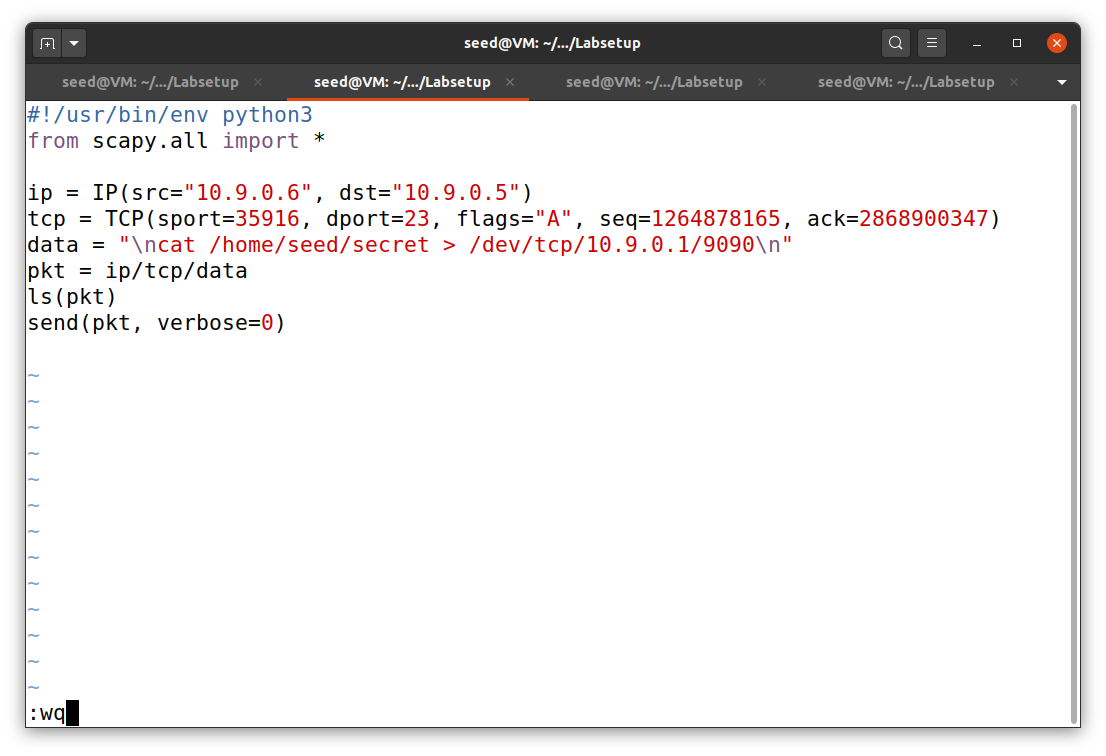
*Wireshark for task 3*



In this task, we have to use three pieces of information from Wireshark: source port, sequence number, and acknowledgement number. Those information will be used in the program as shown in Figure 23.

**Figure 23**

*tcp\_hijack.py*

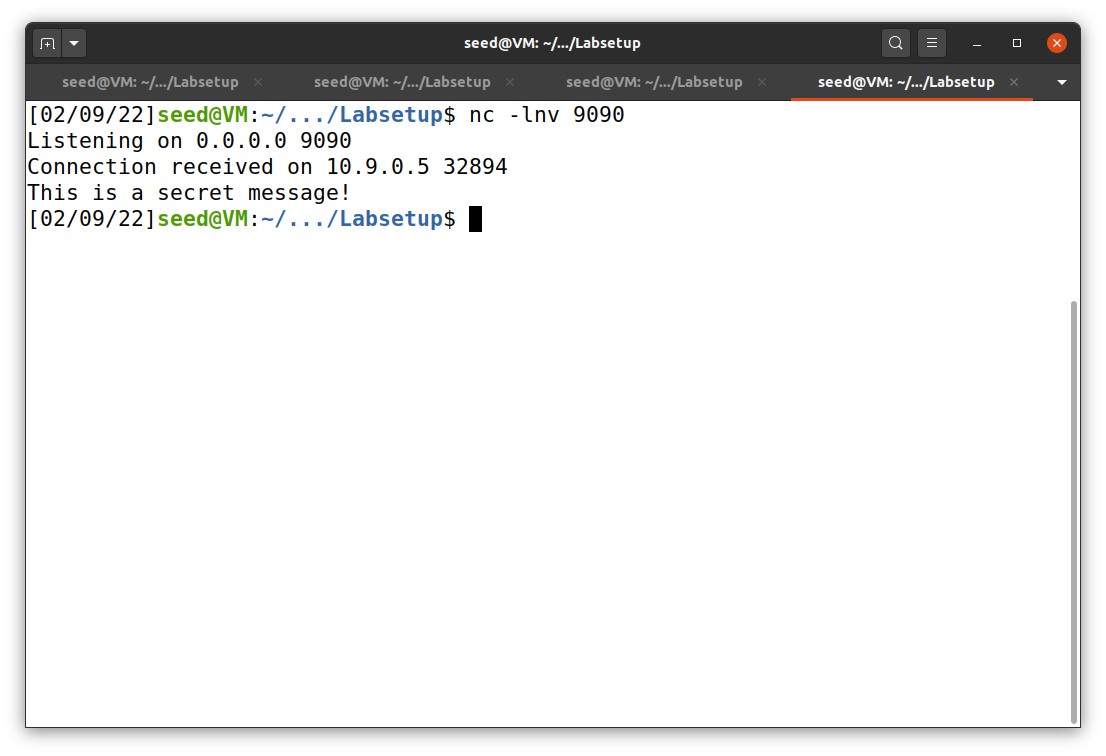


Before we can redirect the data in victim’s machine, we have to use the command below in one of our terminals (Figure 24).

nc -lnv 9090

**Figure 24**

*Terminal with redirected data of task 3*



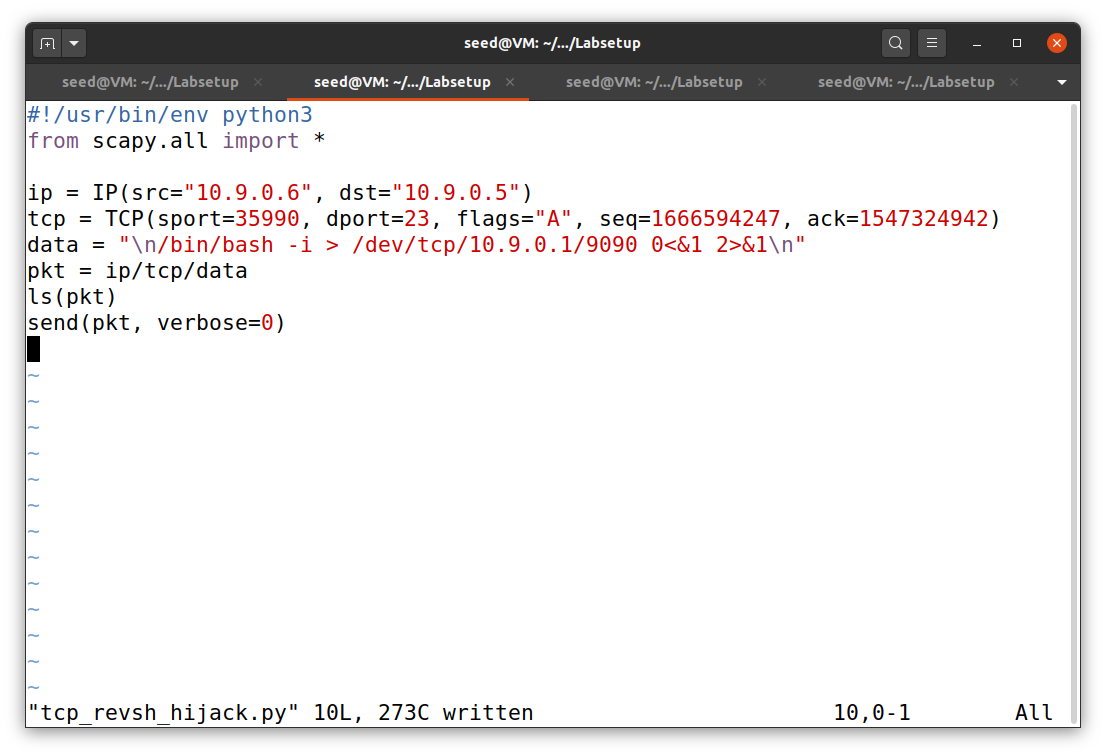
We can also see that the user1 machine is froze after we run the python program.

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

Finally, task 4 is closely similar to task 3. So, I will show only the parts that are different. The main different is the code that contains reverse shell capability, as shown in Figure 25.

**Figure 25**

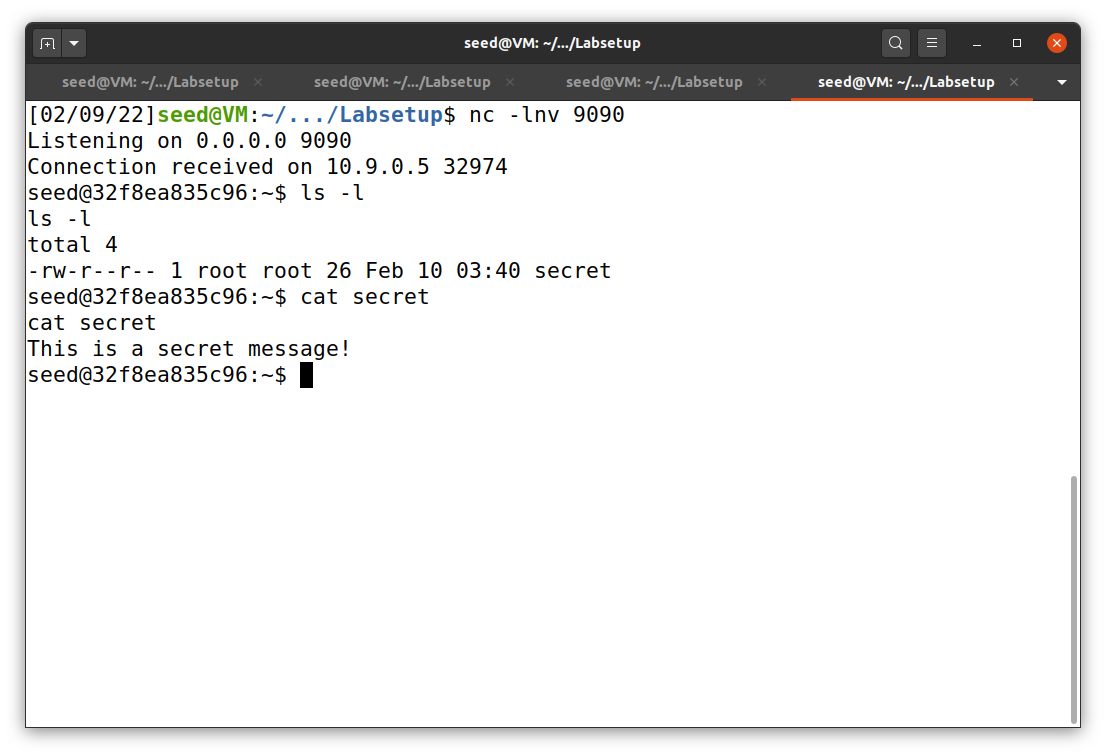
*tcp\_revsh\_hijack.py*



To obtain source port, sequence number, and acknowledgement number, I used Wireshark that similar to previous tasks (Figure 27). After I ran the code, reverse shell can be access via the terminal, as shown in Figure 26.

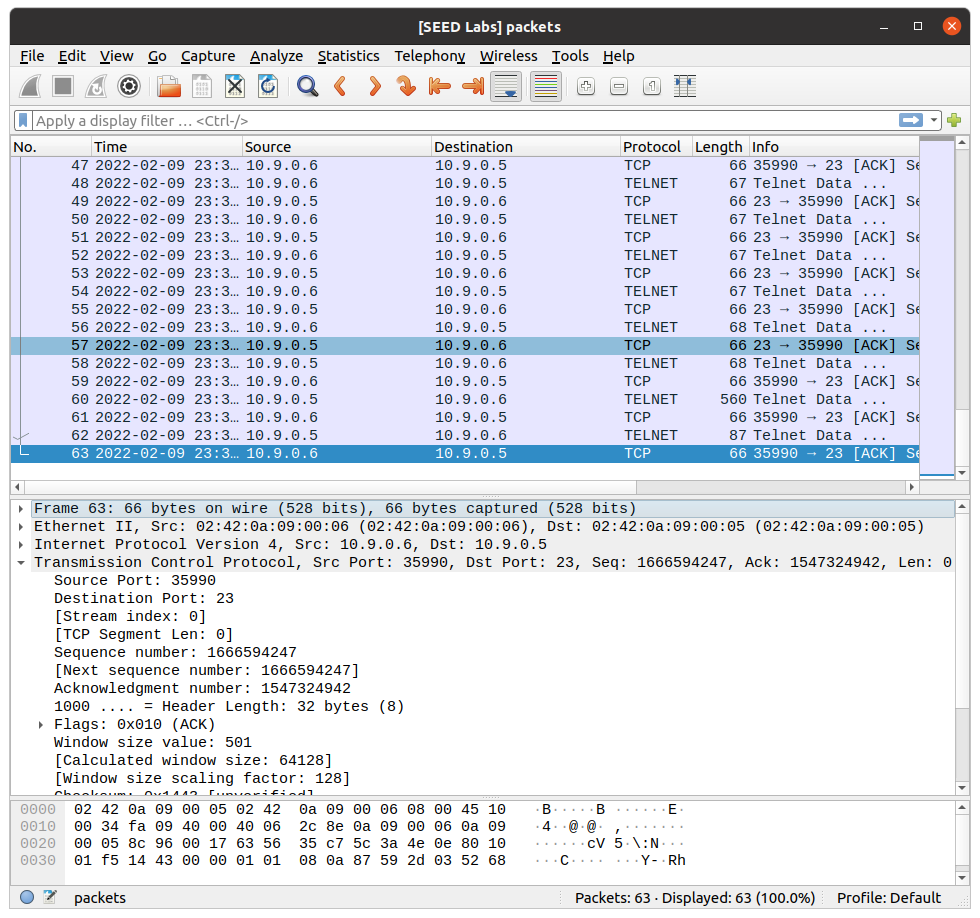
**Figure 26**

*Result of task 4*

**

**Figure 27**

*Wireshark for task 4*



**References**

Du, W. (2019). Computer & internet security: A hands-on approach (2nd ed.). Independently published.

Du, W. (n.d.). TCP Attacks Lab. SeedLabs 2.0. https://seedsecuritylabs.org/Labs\_20.04/‌Networking/TCP\_Attacks/