**Lab 3: ARP Cache Poisoning Attack Lab**

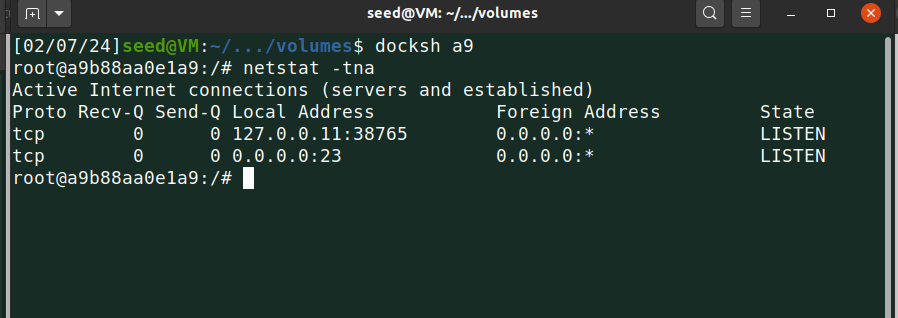
**Task 1: SYN Flooding Attack**

A SYN flood is a type of Denial of Service (DoS) attack where attackers overwhelm a victim's TCP port with numerous SYN requests. The attackers, however, do not intend to complete the TCP three-way handshake process.

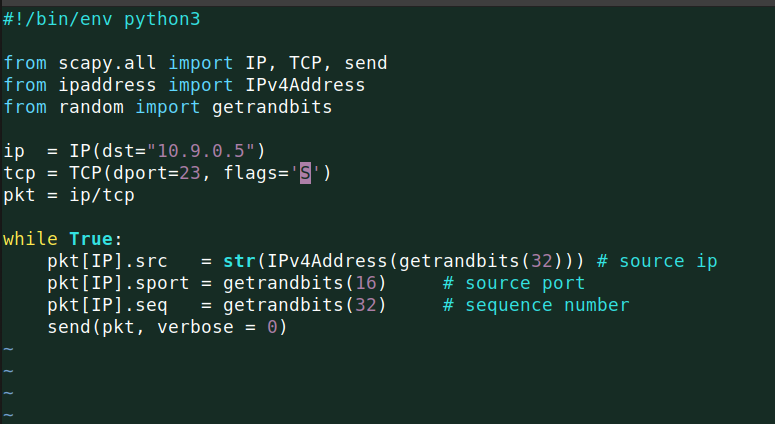
In a SYN flood attack, attackers bombard the victim with requests from random, potentially fake IP addresses, targeting the Transmission Control Block (TCB) queue. Once the TCB reaches its capacity, it cannot accept new requests, causing legitimate connections to fail with a "connection timed out" error.

Before starting let’s check the victim's current TCP connections using netstat -tna

As I see there are no connections currently other than localhost.



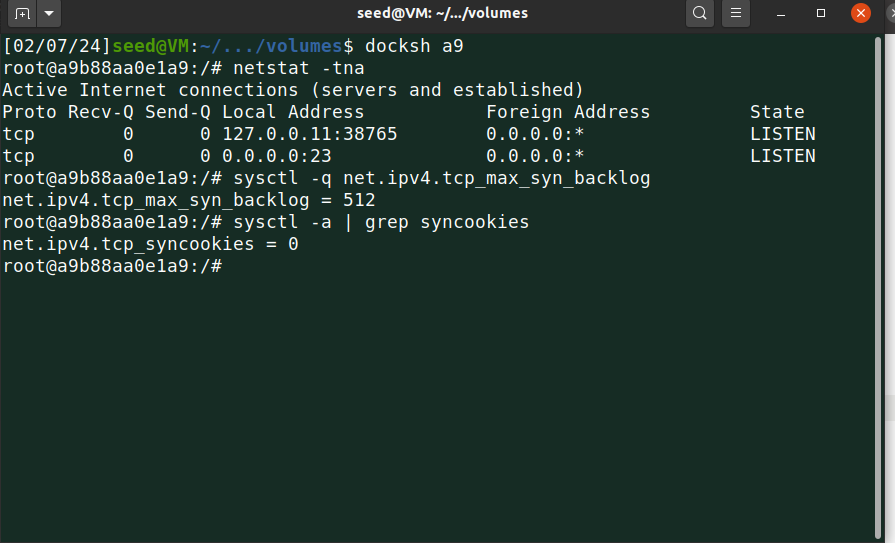
**Code**

****

In the above code, I am using the victim IP address as the destination and TCP packet with destination port 23 and flag SYNC (S).

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

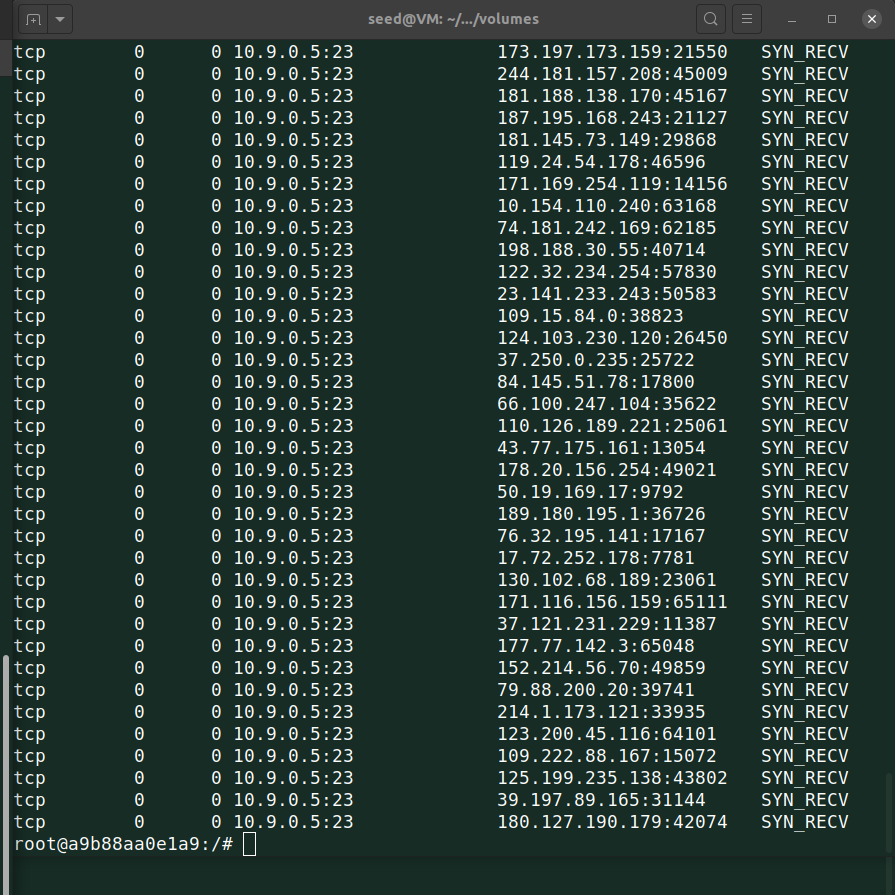
Before initiating the attack, I verified the default TCB size using a command and observed that it is set to 512, with SYN cookies disabled by default.

****

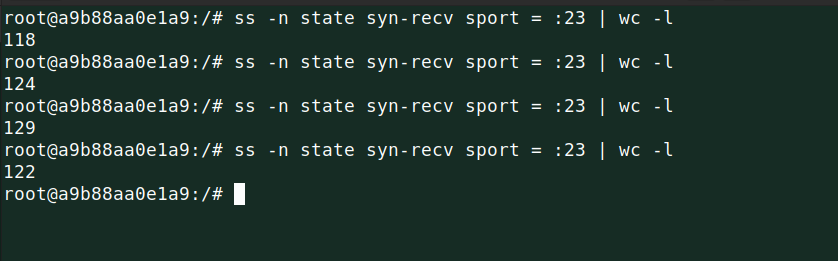
**SS 1.2**

Start the attack…

The below screenshot shows a list of TCP connections to the victim’s TCB, a lot of random IPs from various fake connections.

****

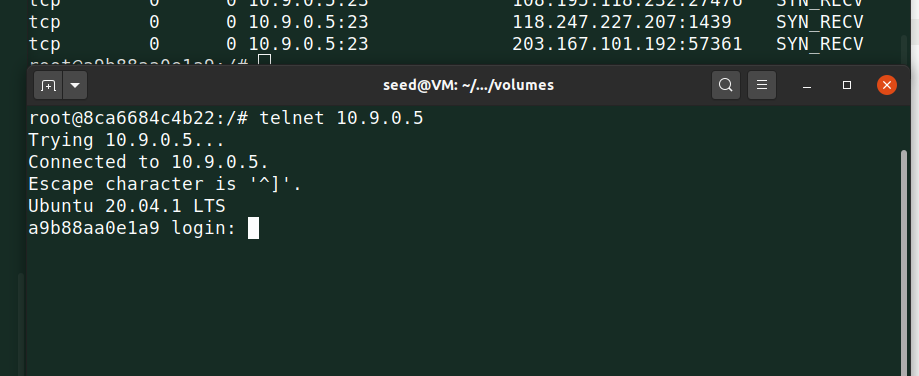
I was constantly checking the TCB size on another terminal using the below command, I was getting different sizes every time. Remember even though the size is 512 default still the full size is not allocated to store entries of requests ¼ size is reserved for proven destinations.

****

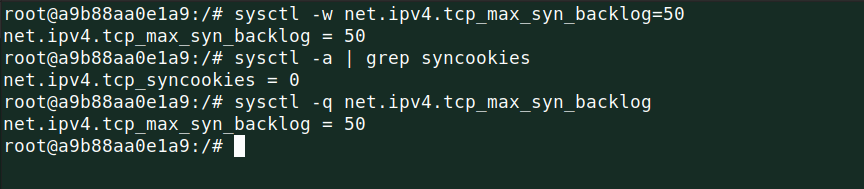
Then I tried to connect the victim from user 1 using telnet and I was able to connect successfully. That means my attack was unsuccessful.

The reason is simple, python is slow.

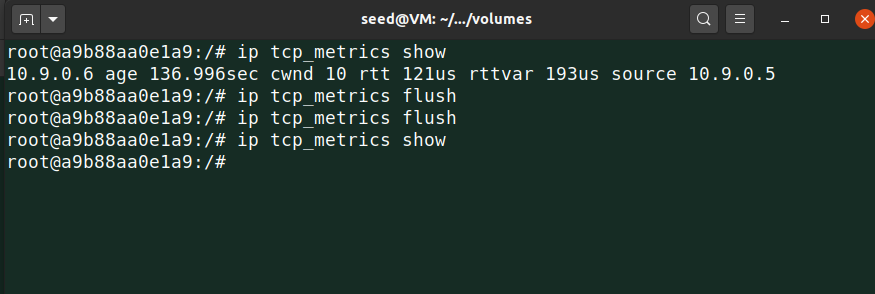
While code tries to flood, TCB never becomes full in one go.

****

Now to make it successful with Python, I updated TCB size to 50 only.

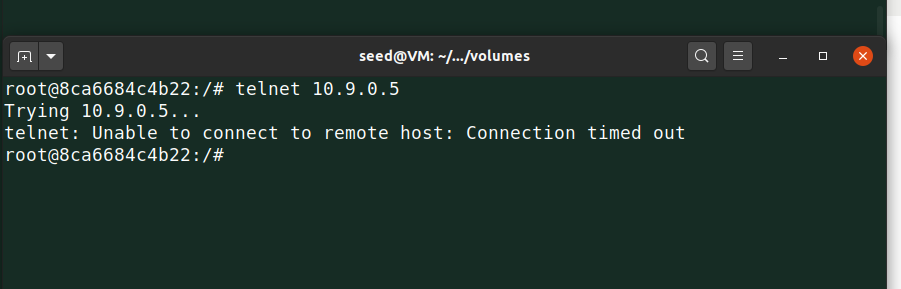
****

Before that flush out the connections which are connected with the victim.

****

Ran the code again….

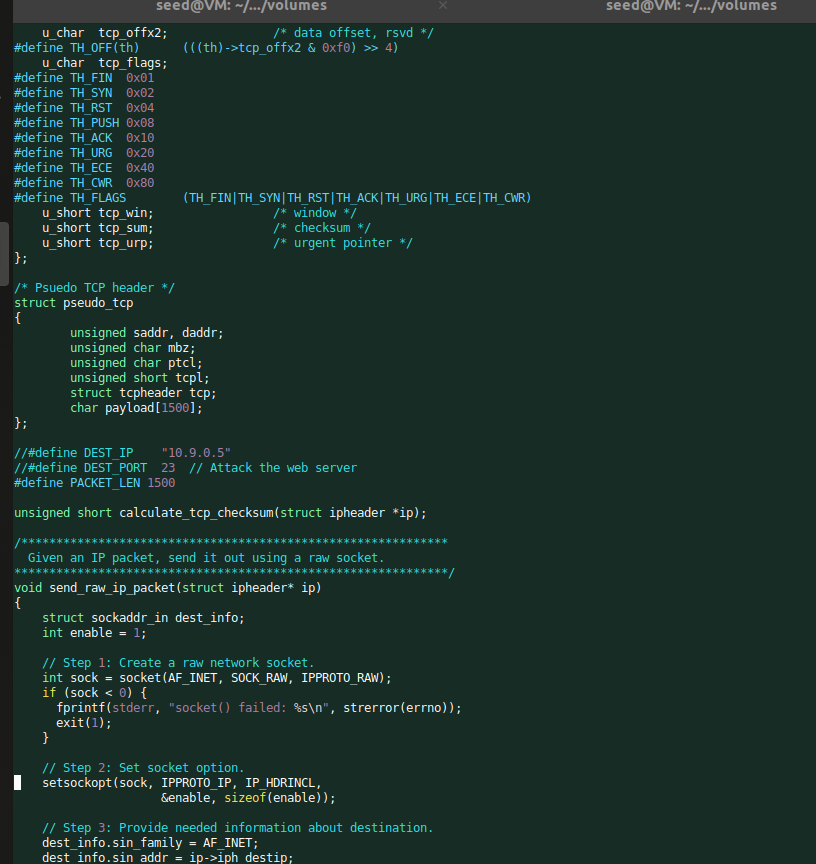
Now the attack is successful.

****

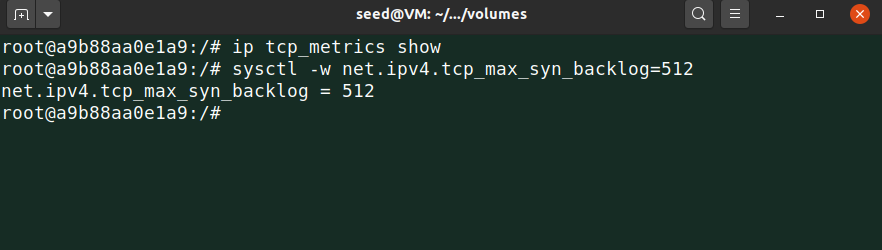
**Task 1.2: With C Code**

I used the same code provided by the professor and changed the destination IP with the victim’s IP.

**Code**

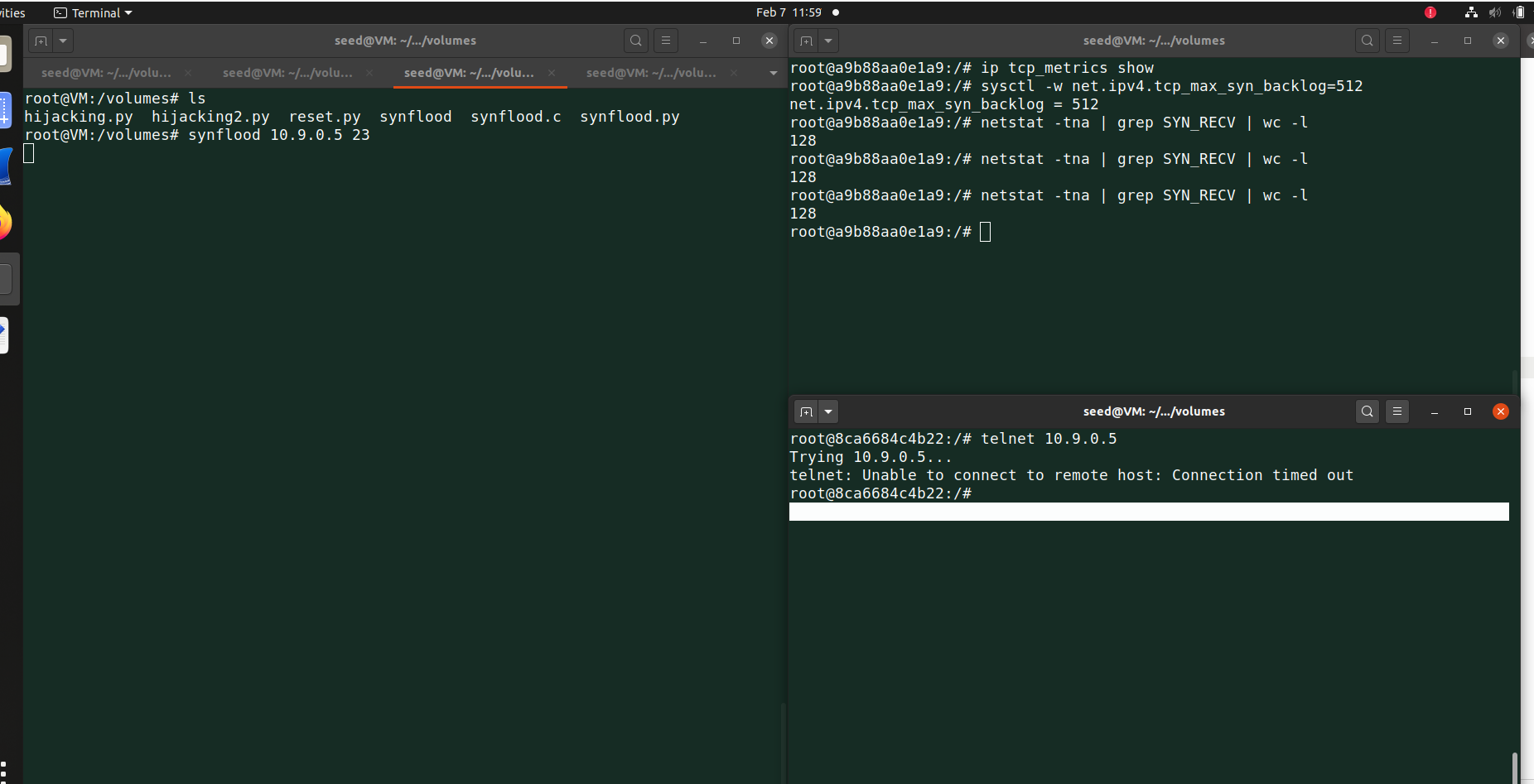
****

Before the attack confirm that there is no connection and TCB size is the default one.

****

Below shows, when I ran the C code and tried to connect with the victim, I got an error message that Unable to connect…

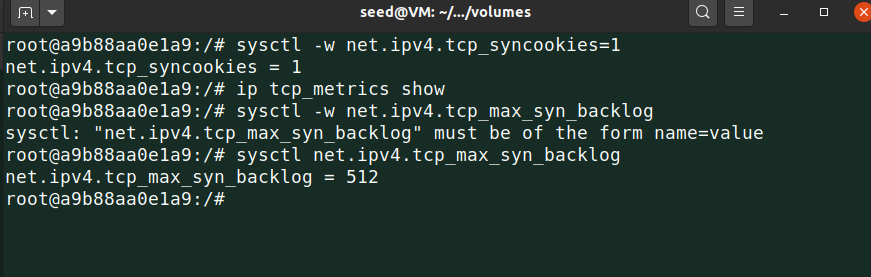
The reason is simple again…C code is fast.

****

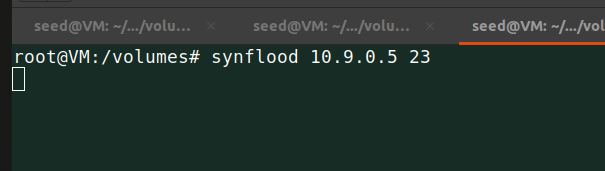
**Task 1.3: Enable SYN Cookie Countermeasure**

Now with this task, I enabled SYN cookies.

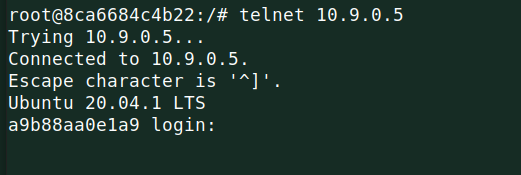
TCB size is 512.

****

Then again I ran both codes and found that the attack is failing because now SYN cookies countermeasure is enabled so OS will discard the requests as OS detected a flooding attack.

**\\**

The connection is successful.

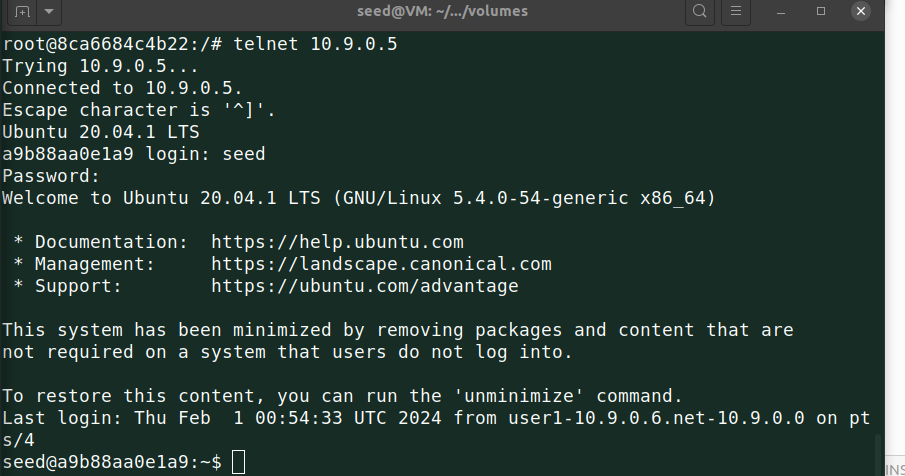
****

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

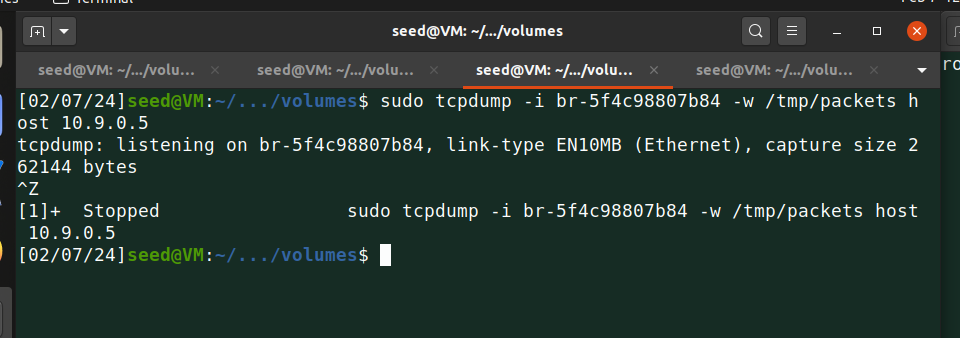
To execute this attack, I require the initial sequence number and the source port.

These can be obtained using TCPDUMP and Wireshark.

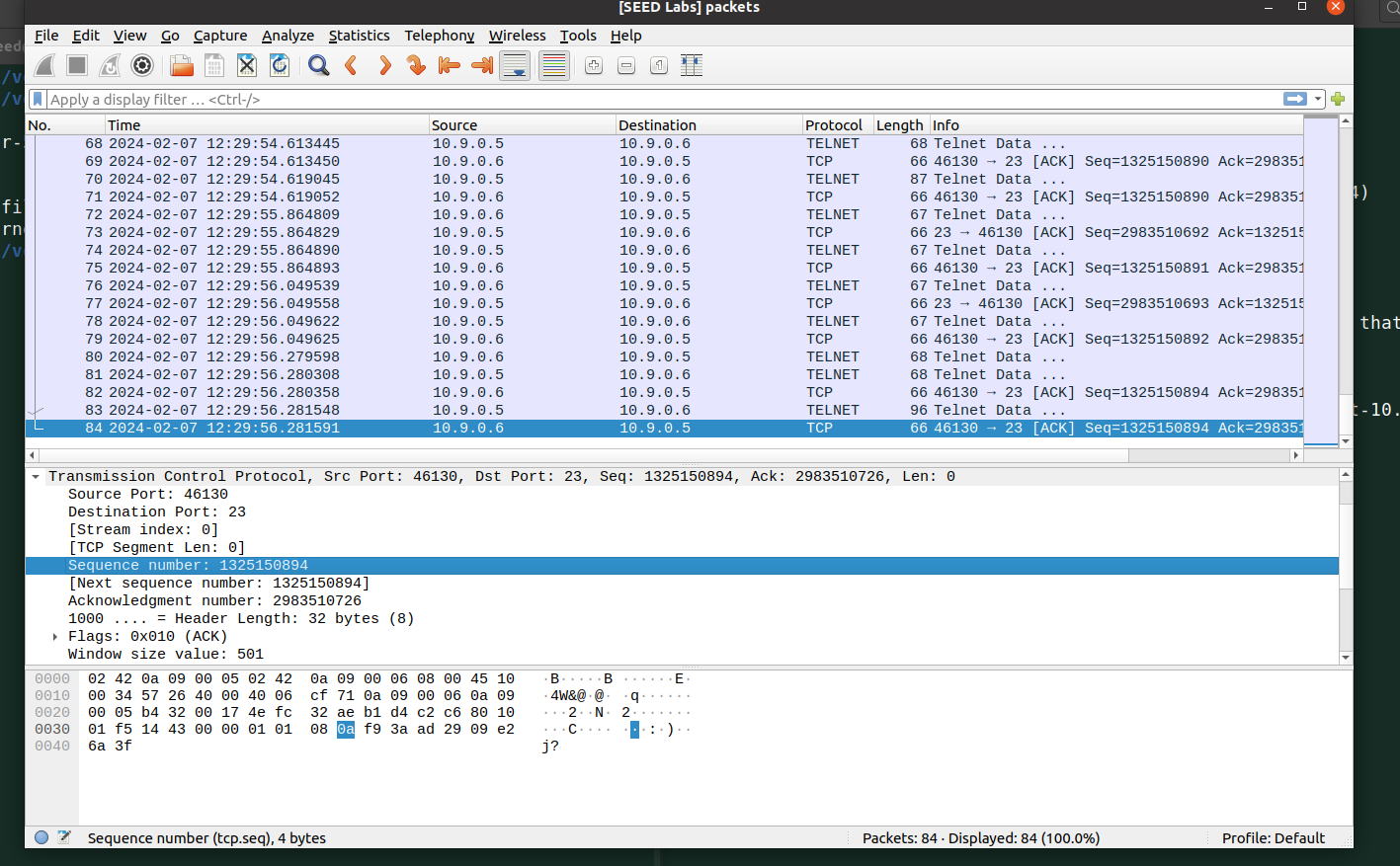
First, executed the "telnet 10.9.0.5" command from User 1 (10.9.0.6).

****

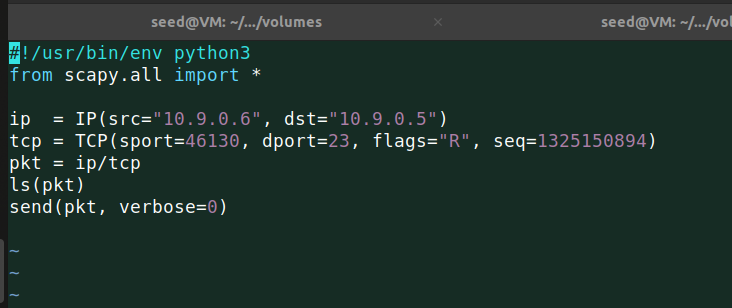
After logging in, stop TCPDUMP and launch Wireshark. Review the most recent TCP packet to extract the sequence number and source port. Make sure user 1 connection window should be open until final attack.

****

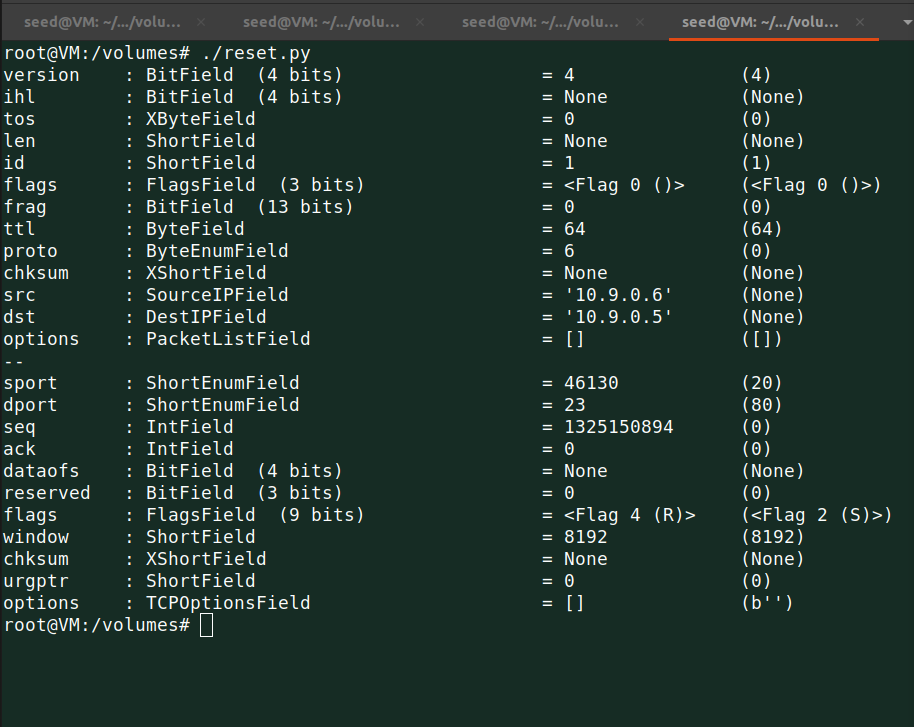
Below screenshot shows wireshark result.

****

**Code**I used those values in below code, used R flag here because I want to reset the connection.

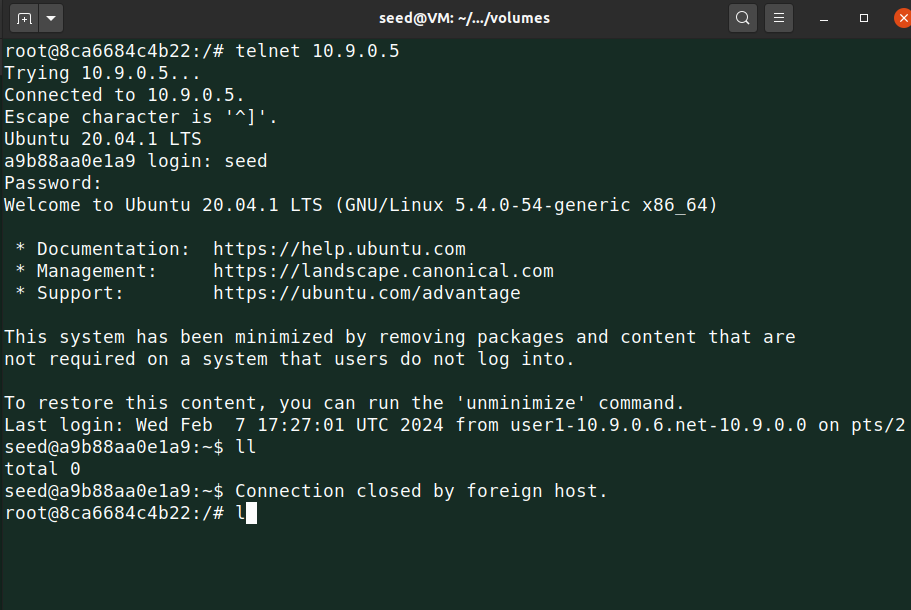
****

Now I ran the code.

****

After that I tried to run ll command on user 1 machine and I got an error saying connection closed by foreign host.

So my attack was successful, server (victim) closed the connection with user 1.

****

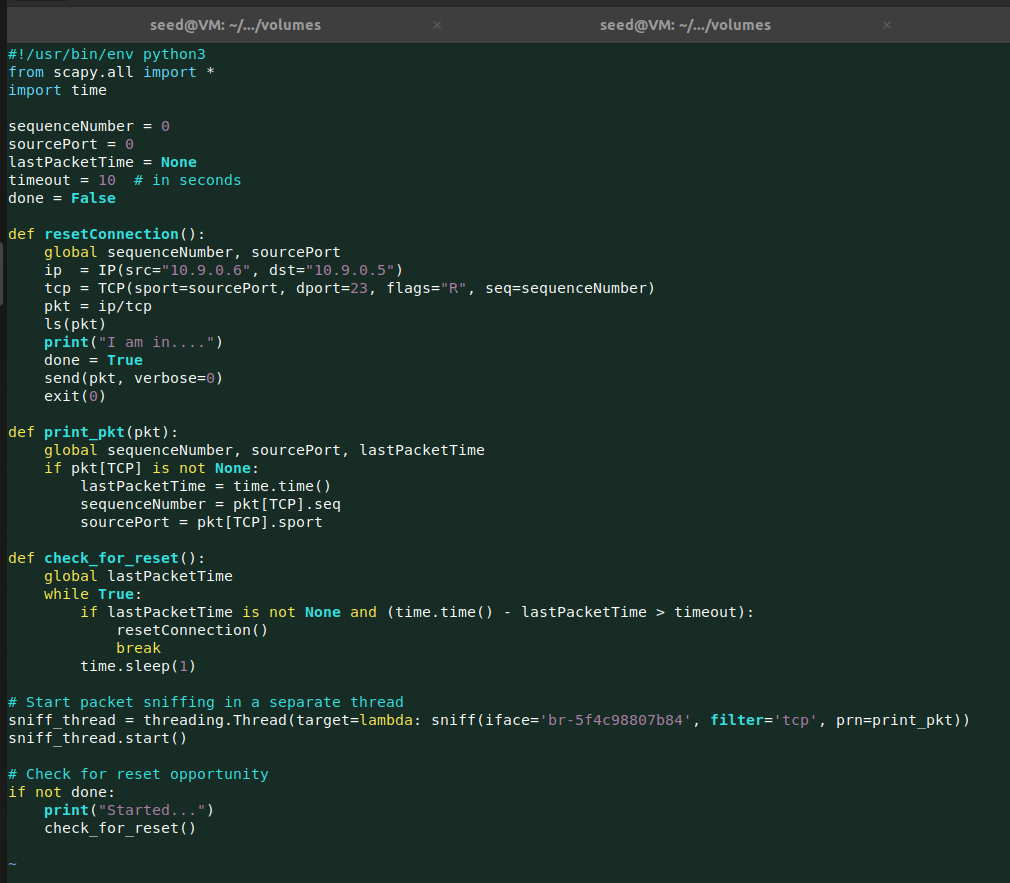
**Optional:**

I recorded optional automation code output on video, which is sniffing and spoofing and then reset the connection.

Logic is simple, code is retrieving each TCP packet on interface.

Once users stop trying to access his terminal, code waits for 5 second and confirms that TCP packets are stopped and immediately copies the information for attack and starts the reset attack.

**Code:**

****

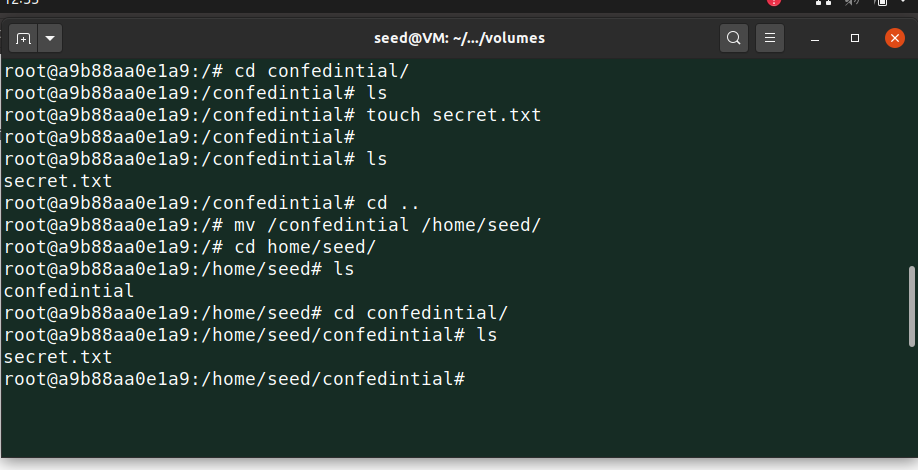
**Output:** Check below video.

[**SEED Ubuntu 20.04 [Running] - Oracle VM VirtualBox 2024-02-07 18-01-12.mp4**](https://ind657-my.sharepoint.com/:v:/g/personal/bagurm01_pfw_edu/ERX-EMuYoudEuEDexNsgExUBnE9HqIw6I42_7D6QOKcasQ)

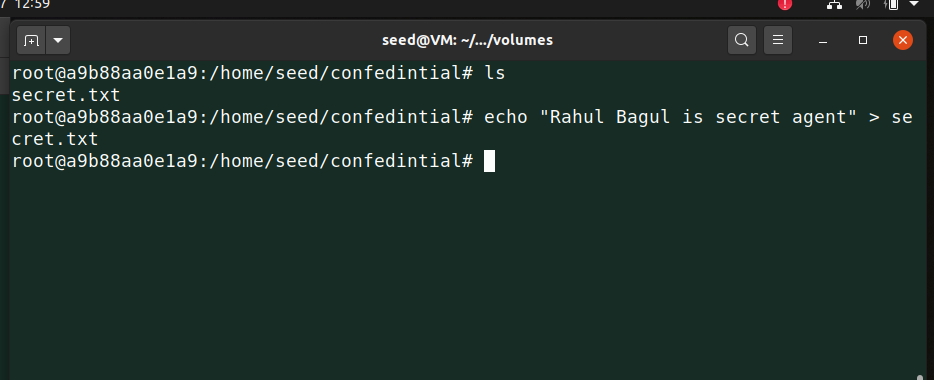
**Task 3: TCP Session Hijacking**

The TCP Session Hijacking attack aims to take over an existing TCP connection (session) between two victims by injecting malicious content into the session.

For this purpose, I created a folder named "confidential" and placed a text file named "secret.txt" inside that folder.

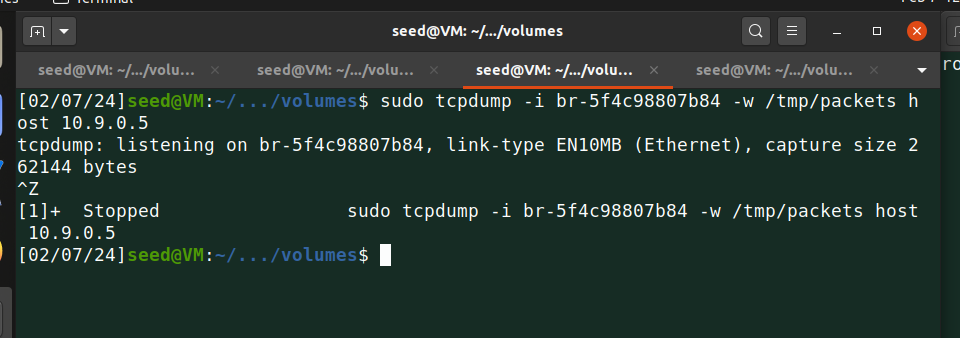
****

I added some text in that file

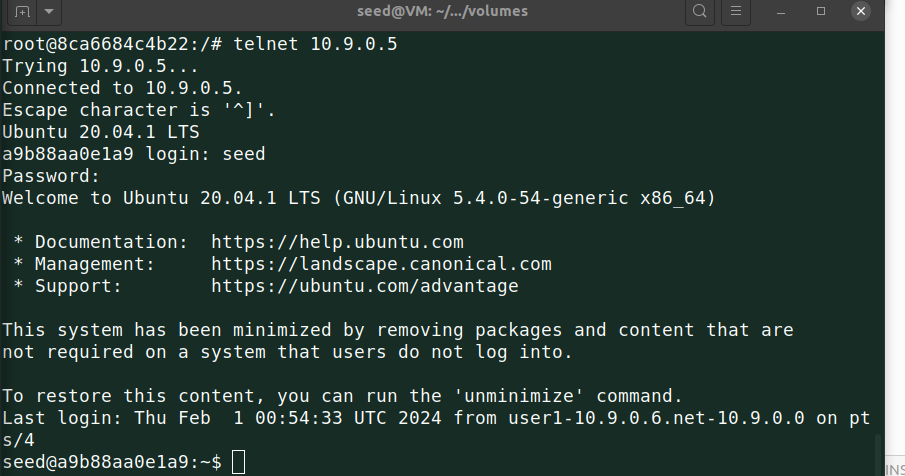
****

This time, I need one more field from packet with sequence and destination port which is acknowledge number.

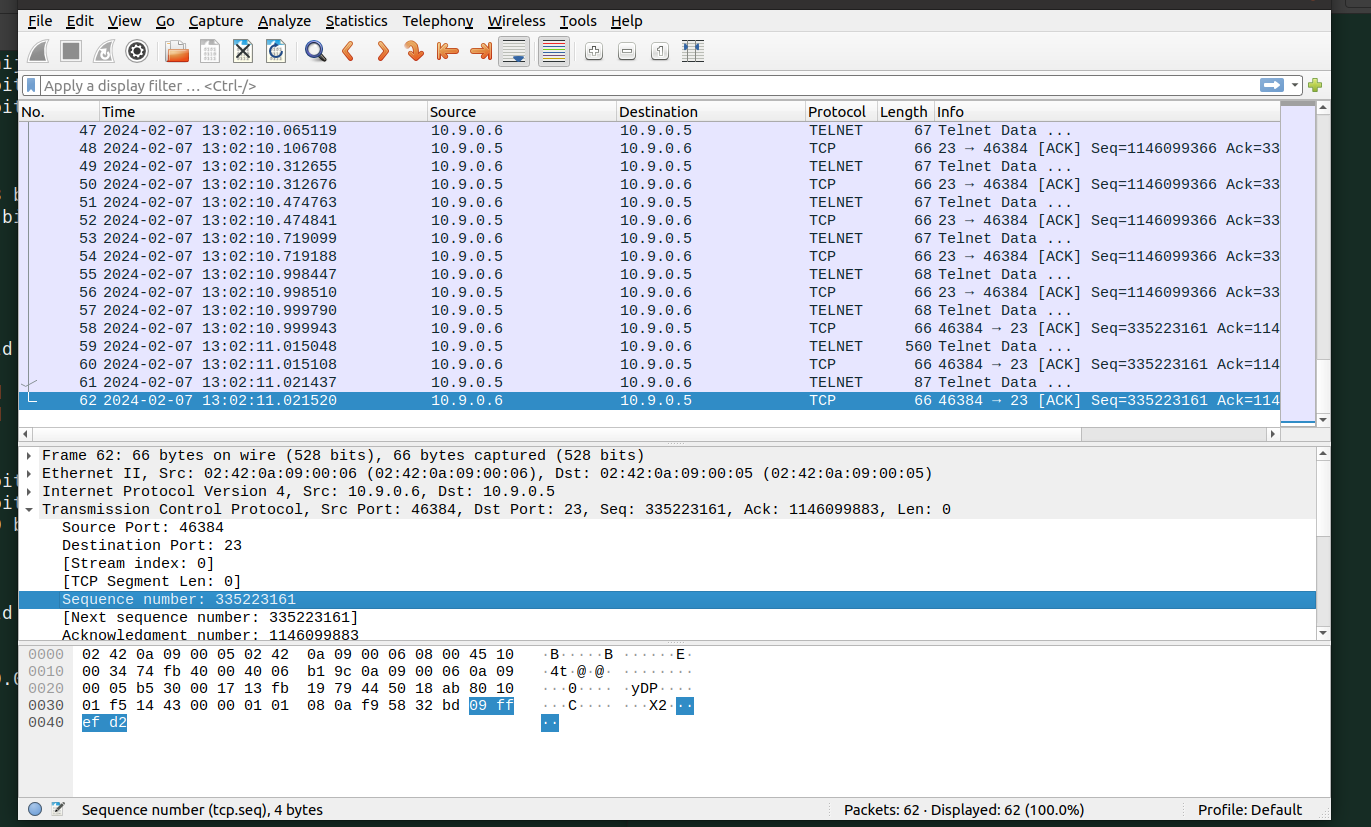
To get that again I sed TCPDUMP and wireshark.

****

First, executed the "telnet 10.9.0.5" command from User 1 (10.9.0.6).

****

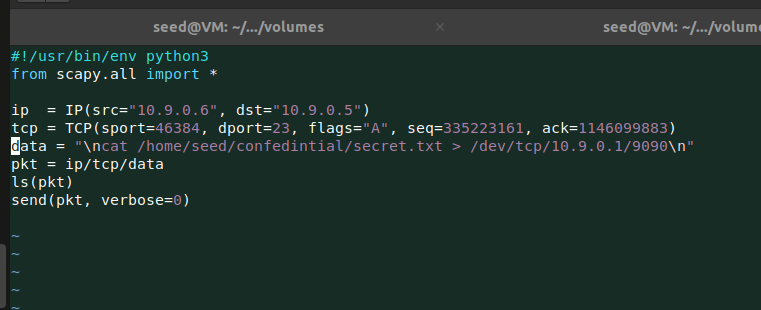
Opened wireshark and checked last TCP packet and copied the values of fields which I need into my code.

****

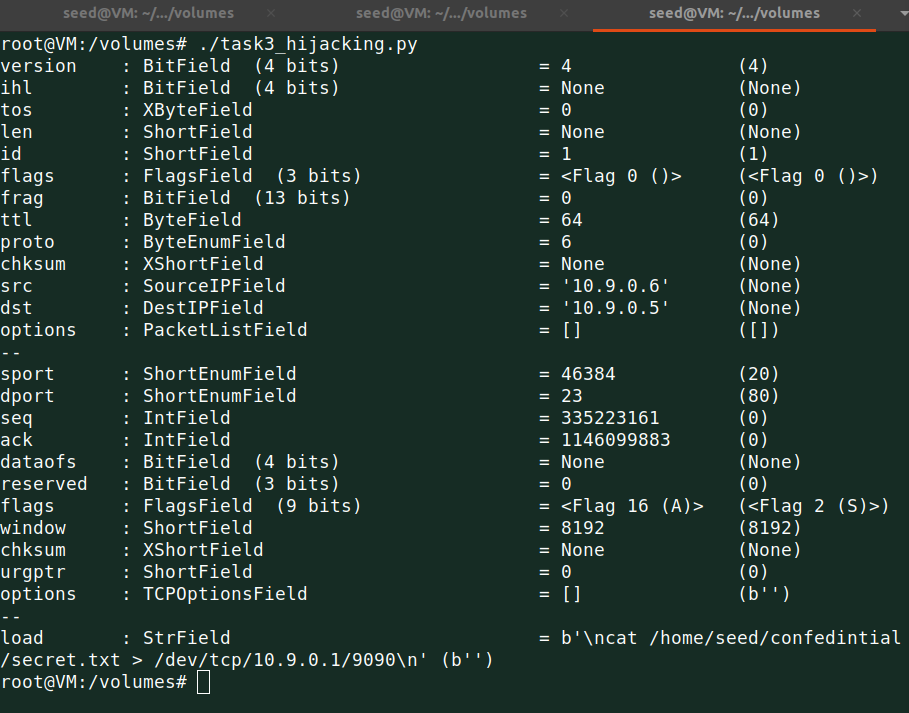
**Code**

Below code shows the values which I copied from TCP packet using wireshark.

I am passing data as to run my command which is ***cat /home/seed/cofedintial/secret.txt*** and send that output to IP 10.9.0.1 and port 9090.

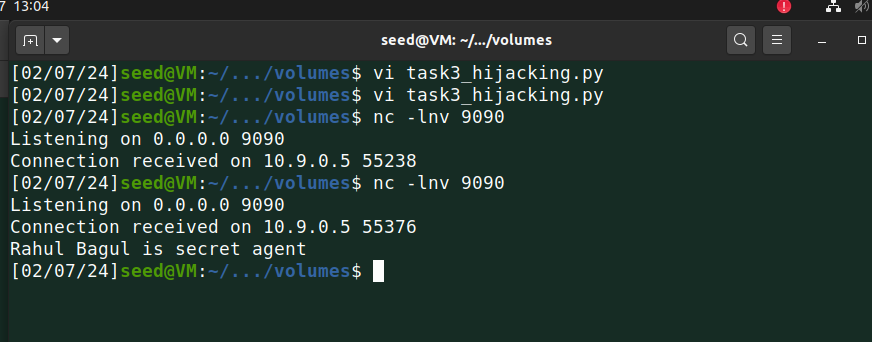
****

Now I ran the code.

****

To get the output I have to start listen on port 9090 using nc command.

After successful code run, I accessed the contents of file on attacker machine.

****

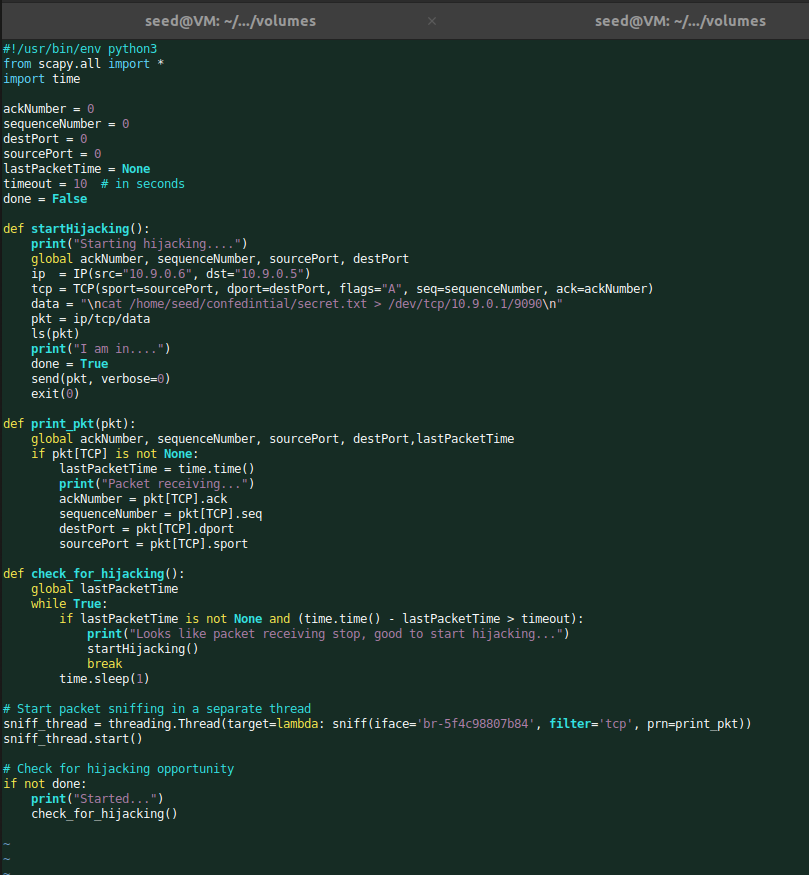
**Optional:**

I recorded optional automation code output on video, which is sniffing and spoofing and then hijack the victim.

Logic is simple, code is retrieving each TCP packet on interface.

Once users stop trying to access his terminal, code waits for 5 second and confirms that TCP packets are stopped and immediately copies the information for attack and starts the hijack attack.

**Code**

****

**Output:** Check below video.

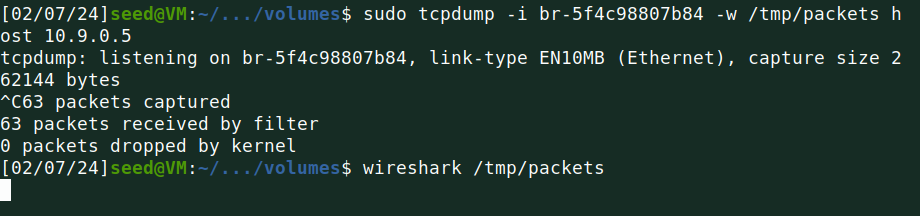
[**SEED Ubuntu 20.04 [Running] - Oracle VM VirtualBox 2024-02-07 17-43-29.mp4**](https://ind657-my.sharepoint.com/:v:/r/personal/bagurm01_pfw_edu/Documents/Attachments/SEED%20Ubuntu%2020.04%20%5BRunning%5D%20-%20Oracle%20VM%20VirtualBox%202024-02-07%2017-43-29.mp4?csf=1&web=1&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=bj7mZU)

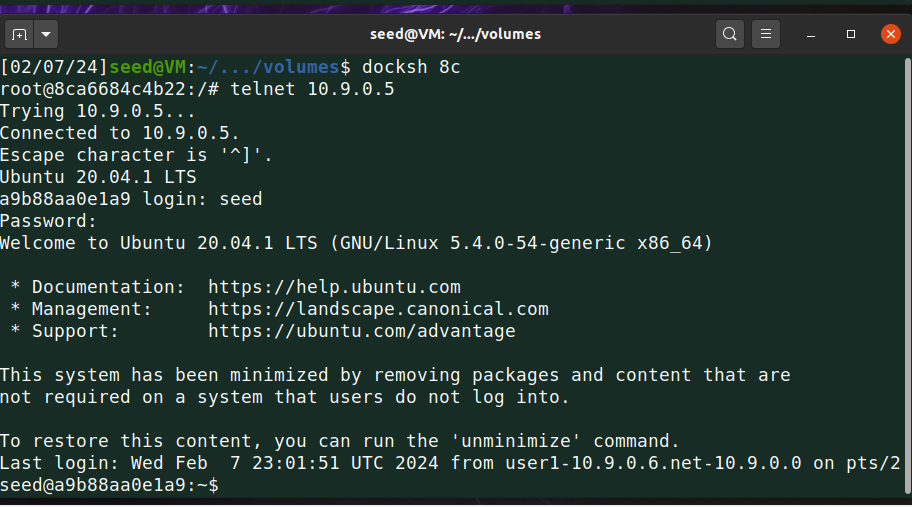
**Task 4: Creating Reverse Shell using TCP Session Hijacking**

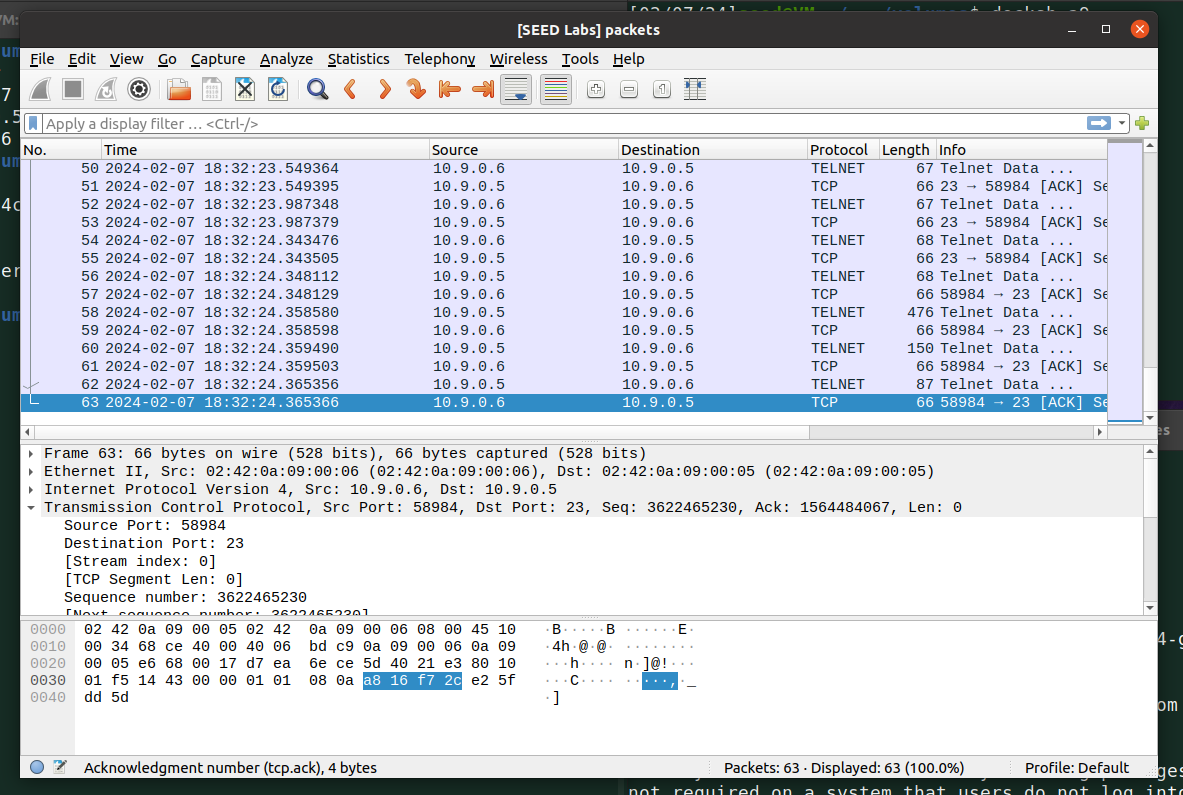
Now to run more commands on victim’s machine, I need to access shell.

To do that I am passing now new data which contains that run the shell and pass output to standard output and take input from standard input.

Before that we again need to find information like sequence number, port and acknowledge number using TCPDUMP and wireshark.

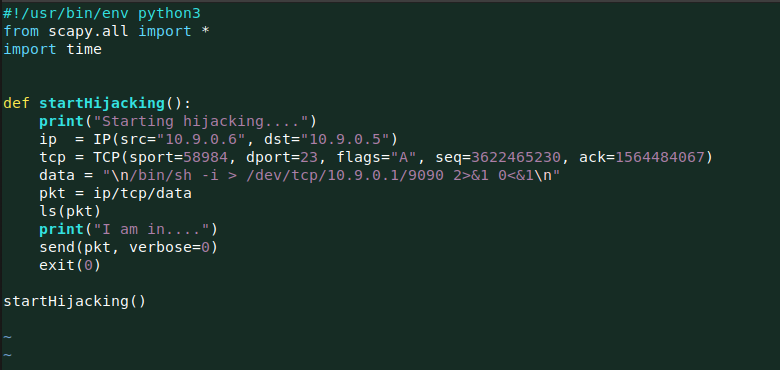
****

****

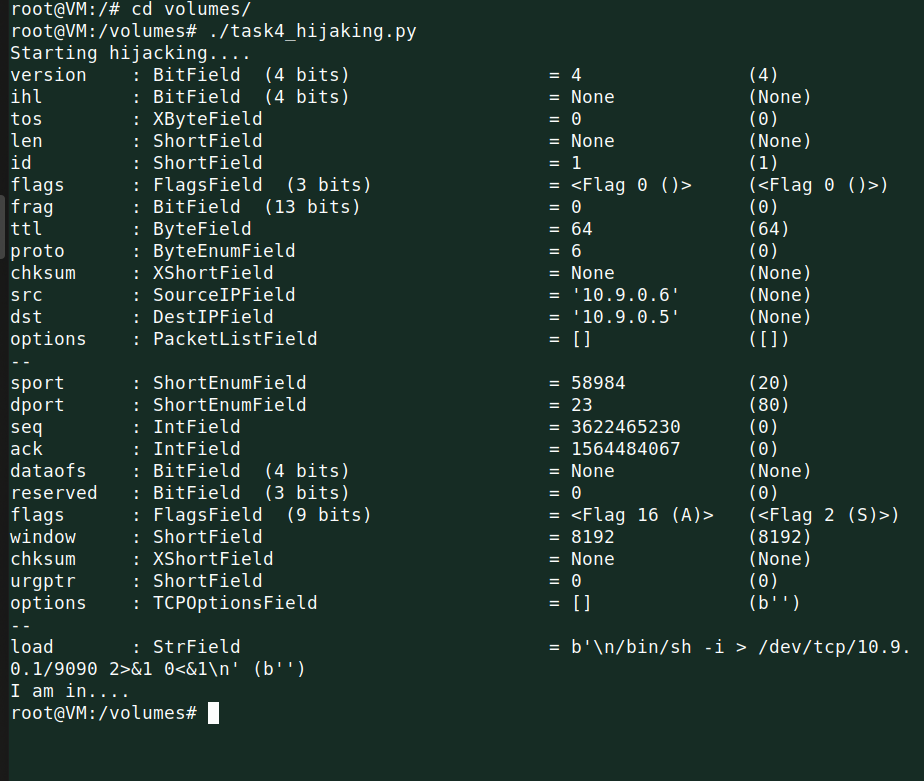
****

I copied those values into my below code.

**Code**

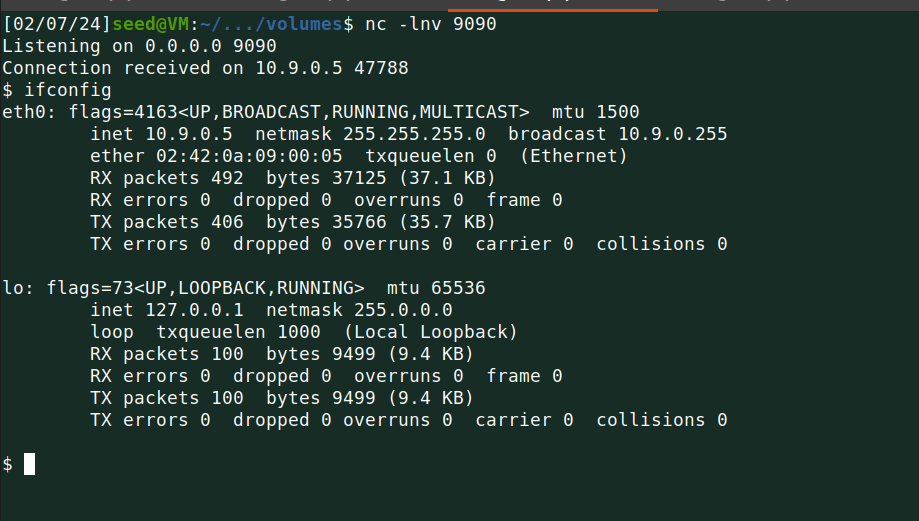
****

I ran the code….

****

To get shell access, I have to start listening at port 9090.

Once attack is successful, I saws the $ sign of shell on my attacker machine and I ran ifconfig command to show that I log in to victim machine.

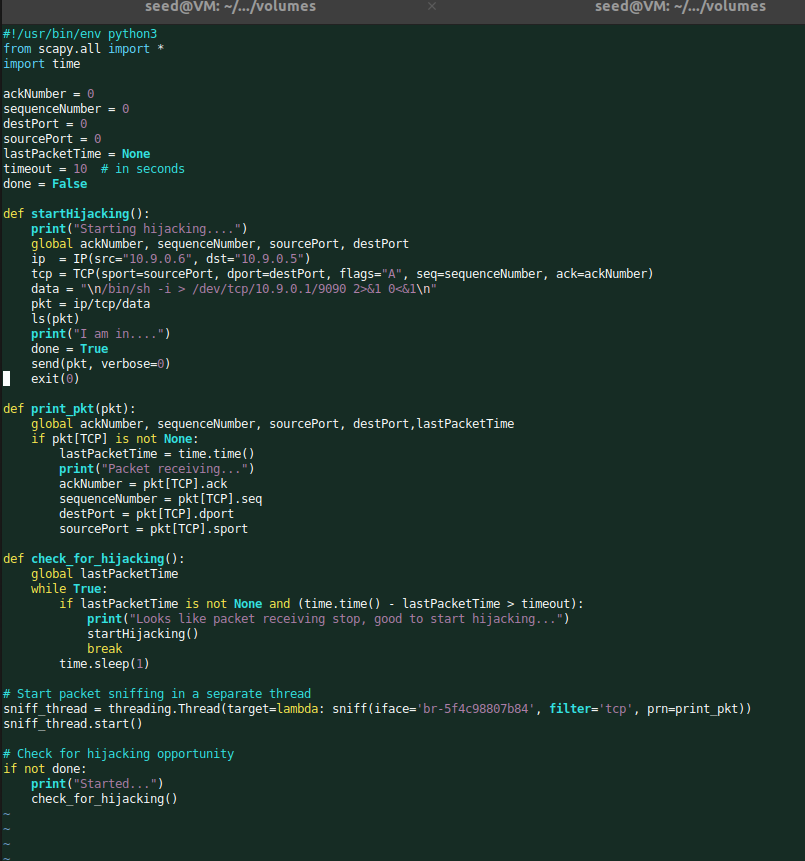
****

**Optional:**

I did this task as class activity.

[**SEED Ubuntu 20.04 [Running] - Oracle VM VirtualBox 2024-01-31 19-53-54.mp4**](https://outlook.office.com/mail/safelink.html?url=https://nam04.safelinks.protection.outlook.com/?url=https%3A%2F%2Find657-my.sharepoint.com%2F%3Av%3A%2Fg%2Fpersonal%2Fbagurm01_pfw_edu%2FEcsrIEoeeXZElmyW4owAvugB-_XC5VbvDzrOAIE2NUEnbw&data=05%7C02%7Cbagurm01%40pfw.edu%7C7f7dbc7673a84d11acc708dc22c7c65c%7Cb7dc318e8abb4c849a6a3ae9fff0999f%7C0%7C0%7C638423488616020275%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C0%7C%7C%7C&sdata=d5ZjMvIp%2B4vJnr35p6aXSmJx7qB91QLwIU8Tn3MHqdk%3D&xdata=Q1Q9MTcwNzM1MzQ4MjUzMyZPUj1PV0EtTlQmQ0lEPTI4ZWNiODBmLWRmMjAtZGJhYy02OTc2LTcwMTJkYzk2ZmFhZQ%3D%3D&reserved=0&corid=d8aab21e-3ead-4060-f3d4-6c4aa7c9ded9)

**Code**

****