[REPORT - 실습4]

-Program Memory Segment-

과 목 명 : 컴퓨터보안 - 02
교 수 : 김영부 교수님
학번 : 2020112119
이름 : 강동희
제출일 : 2023.04.10

목 차

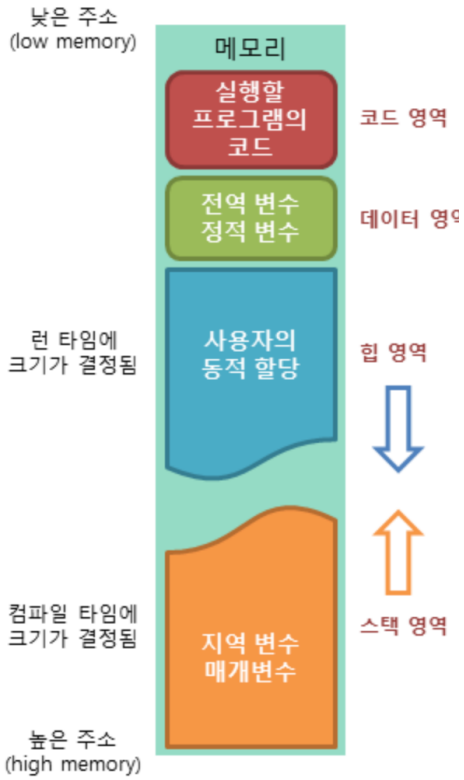
1 서 론

1.1 문제 정의

1. 프로그램 메모리 세그먼트

(1) 프로그램 메모리의 구조

프로그램을 실행하기 위해서는 가장 먼저 메모리에 load 되어야 한다. 또한, 프로그램 내에서 사용되는 변수들을 저장할 메모리도 필요하게 된다. 프로그램을 실행하기 위해 컴퓨터의 운영체제는 메모리 공간을 다음과 같이 영역을 나누어 로드 및 관리하게 된다.



<그림 1. 운영체제가 제공하는 메모리의 공간>

Code segment는 실행할 프로그램의 코드(text)가 저장되는 영역으로 CPU는 Code segment에서 저장된 명령어를 하나씩 가져와 처리하게 된다.

Data segment는 프로그램의 전역/정적 변수가 저장되는 영역이다. 프로그램 시작과 함께 할당되며, 프로그램이 종료되면 할당된 내용, 공간은 소멸된다.

Stack segment는 함수의 호출과 관계되는 지역 변수와 매개 변수가 저장되는 영역이다.

2. 컴파일 명령어(gcc)

GCC(GNU Compiler Collection)는 C, C++ 등을 위한 컴파일러로 리눅스

및 유닉스 운영체제에서 가장 많이 사용되는 컴파일러 중 하나이다.
“gcc [option] [source file] -o [output file name]”의 형태로 사용한다.
-c option : 컴파일만 하고 링크는 하지 않고 오브젝트 파일을 생성한다.
-l option : 링크할 라이브러리 파일의 이름을 지정한다.

3. ELF 파일
ELF(Executable and Linkable Format)는 컴퓨터 파일 포맷 중 하나로, 주로 리눅스와 유닉스 계열의 운영체제에서 사용된다. ELF 파일은 실행 가능한 프로그램, 공유 라이브러리, 코어 덤프(core dump), 오브젝트 파일 등을 포함할 수 있다. ELF 파일은 세 가지 섹션으로 구성된다.
1) Header section
ELF 파일의 기본 정보를 포함한다. (파일 형식, 아키텍처 등)
2) Section Header section
ELF 파일이 가지고 있는 모든 섹션에 대한 정보를 포함한다. (섹션의 이름, 시작 주소, 크기, 플래그 등)
3) Section Data
ELF 파일의 실제 데이터를 포함한다. (실행코드, 데이터, 심볼 테이블 등)

ELF 파일은 Dynamic Linking을 지원한다. 즉, 실행 시간에 필요한 라이브러리를 로드가 가능하며, 여러 프로그램에서 같은 라이브러리를 공유하여 메모리를 절약할 수 있다는 장점이 있다.

1.2 문제 분석

1. 변수 선언 및 초기화에 따른 세그먼트 크기 분석
Practice1.c 파일을 생성하여 전역 변수와 정적 변수의 선언, 그리고 각각의 변수를 값에 따라 초기화함에 따라 세그먼트의 크기를 비교하여 차이점을 분석한다. 실습을 진행한 환경은 Mac OS로, 파일의 세그먼트 크기를 확인하기 위해 터미널에 ‘size’ 명령어를 통해 파일 정보를 확인할 수 있으며, 각 세그먼트 크기를 확인하려면 -m 옵션(mapping mode)를 통해 세그먼트 매핑 정보를 출력할 수 있다. 이를 통해 알 수 있는 정보는 test, data, bss, dec, hex 가 있다.

2. 오브젝트 파일 수정 및 링킹 실습(1)
해당 실습을 통해 프로그램 실행 파일의 ELF 파일 형식을 확인하여 바이너리 파일 포맷의 정보를 확인할 수 있다. Mac OS에서 ELF 파일 형식을 확인하기 위해 터미널에 ‘objdump -p’ 명령어와 옵션을 통해 바이너리 파일의 종류 및 크기 등의 정보를 확인할 수 있다. 또한 VSCode의 확장 프로

그램으로 ‘Hex Editor’를 설치하여 오브젝트 파일의 내용을 확인하고 파악할 수 있는 정보들에 대해 정리한다. 링킹을 통해 소소코드에서 컴파일 된 오브젝트 파일을 실행가능한 바이너리 파일을 생성하여 출력 결과를 확인해본다.

3. 오브젝트 파일 수정 및 링킹 실습(2)
   실습2와 마찬가지로 소스 코드로부터 오브젝트 파일을 생성하기 위해 컴파일을 진행하고, Hex Editor를 이용하여 소스 코드를 직접 수정하지 않고 코드를 수정하여 올바른 문자열을 출력하도록 한다. 이 때, 다양한 방법으로 접근하여 여러 케이스별로 문자열을 출력할 수 있게 시도한다.

4. SetUID Program 실습
   SetUID 프로그램은 일반적으로 특정 프로그램을 루트 권한으로 실행시키는 방법이다. 파일의 소유자가 root이고, 실행 권한이 설정되어 있으면, 해당 프로그램이 root 권한으로 실행된다. 이를 통해 특정 사용자가 권한이 필요한 작업을 수행할 수 있도록 해준다.

2 본 론

2.1 실습 1 : 변수 선언과 초기화에 따른 세그먼트 크기 분석

(0) 실습 환경

macOS 에서 진행한다.

(1) 'Practice1.c' 파일을 생성 후 실행 파일을 생성하고 세그먼트 크기 확인

Vim 을 사용하여 Practice1.c 파일을 생성 후 코드를 작성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % pwd
/Users/kangdonghee/Desktop/Computer Security/week4
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % vi Practice1.c
```

파일 내용을 수정하기 위해 <i>를 통해 다음 코드를 입력한 후
<ESC + :wq!>를 통해 수정한 파일 내용을 저장 후 종료한다.

```
#include<stdio.h>

int main(){
        return 0;
```

<Practice1.c 의 코드>

result1 의 이름으로 컴파일하여 실행파일을 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc Practice1.c -o result1
```

size 명령어를 통해 실행파일의 세그먼트 크기 값을 확인하고 분석한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result1
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __LINKEDIT: 16384
total 4295000064
```

먼저, 실행 파일의 세그먼트 크기를 확인하기 위해 size -m <실행파일>을 통해 확인할 수 있으며, Segment __TEXT 는 16,384KB 이며, 세부적으로는 text 15 바이트, unwind_info(예외처리 및 함수 호출을 위한 정보를 제공하는 섹션)은 72 바이트임을 알 수 있다. Segment __LINKEDIT 의 정보는 링커가 사용하는 정보를 포함하는 내용으로, 바이너리 파일에서 사용하는 함수와 변수의 이름과 위치 정보 등이 저장된다. 해당 크기는 16,384KB 로 확인할 수 있었다.

(2) Practice1.c 코드에 정수형 전역 변수 global 을 선언하고 result2 로 컴파일하여 기존 세그먼트 크기와 비교하기.

Vim 을 사용하여 Practice1.c 코드를 수정한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % vim Practice1.c
```

파일 내용을 수정하기 위해 <i>를 통해 정수형 global 전역 변수를 선언하고,
<ESC + :wq!>를 통해 수정한 파일 내용을 저장 후 종료한다.

```c
#include<stdio.h>

int global;

int main(){
        return 0;
}
```

〈Practice1.c 의 수정 1 코드〉

result2 의 이름으로 컴파일하여 실행파일을 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc Practice1.c -o result2
```

size 명령어를 통해 result2 의 세그먼트 크기 값을 확인하고 분석한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result2
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __common: 4
        total 4
Segment __LINKEDIT: 16384
total 4295016448
```

TEXT 세그먼트의 크기는 16,384KB 이며, 전역 변수를 선언하면서 DATA 세그먼트의 크기도 확인할 수 있었다. 그 중 __common 섹션은 전역 범위에서 선언된 변수 중 초기화를 하지 않은 경우에 해당한다. 이 섹션은 해당 변수(global, 4 바이트)가 사용될 때 메모리에 할당된다는 것을 알 수 있다.

(3) 정적 변수 i 를 선언하고, result3 의 이름으로 컴파일, 세그먼트 크기 확인하기.

Vim 을 사용하여 Practice1.c 코드를 수정한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % vim Practice1.c
```

파일 내용을 수정하기 위해 〈i〉를 통해 정적 변수 i 를 선언하고,
〈ESC + :wq!〉를 통해 수정한 파일 내용을 저장 후 종료한다.

```c
#include<stdio.h>

int global;

int main(){
        static int i;
        return 0;
}
```

〈Practice1.c 의 수정 2 코드〉

result3 의 이름으로 컴파일하여 실행파일을 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc Practice1.c -o result3
```

size 명령어를 통해 result3 의 세그먼트 크기 값을 확인하고 분석한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result3
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __bss: 4
        Section __common: 4
        total 8
Segment __LINKEDIT: 16384
total 4295016448
```

__DATA 세그먼트에서 기존 세그먼트와의 차이점을 확인할 수 있다. 전역 변수는 그대로 4 바이트로 유지되며, __bss 가 4 바이트로 새로 추가되었다. 초기화되지 않은 정적 변수의 크기를 확인할 수 있다.

(4) 전역 변수 global 을 10 으로 초기화하고, result4 의 이름으로 컴파일, 세그먼트 크기 확인하기.

Vim 을 사용하여 Practice1.c 코드를 수정한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % vim Practice1.c
```

파일 내용을 수정하기 위해 <i>를 통해 global 변수를 10 으로 초기화하고,
<ESC + :wq!>를 통해 수정한 파일 내용을 저장 후 종료한다.

```
#include<stdio.h>

int global = 10;

int main(){
        static int i;
        return 0;
}
```

<Practice1.c 의 수정 3 코드>

result4 의 이름으로 컴파일하여 실행파일을 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc Practice1.c -o result4
```

size 명령어를 통해 result4 의 세그먼트 크기 값을 확인하고 분석한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result4
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __data: 4
        Section __bss: 4
        total 8
Segment __LINKEDIT: 16384
total 4295016448
```

Result4 는 전역변수를 4 로 초기화하여 컴파일을 하여 __DATA 세그먼트의 변화를 확인할 수 있다. 기존 __common 섹션이 4 바이트로 할당되었었지만, 값을 부여하여 __data 섹션으로 4 바이트가 할당되었다.

(5) 정적변수 i 를 100 으로 초기화하고, result5 의 이름으로 컴파일, 세그먼트 크기 확인하기.

Vim 을 사용하여 Practice1.c 코드를 수정한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % vim Practice1.c
```

파일 내용을 수정하기 위해 <i>를 통해 i 변수를 100 으로 초기화하고,
<ESC + :wq!>를 통해 수정한 파일 내용을 저장 후 종료한다.

```c
#include<stdio.h>

int global = 10;

int main(){
        static int i = 100;
        return 0;
}
```

〈Practice1.c 의 수정 4 코드〉

result5 의 이름으로 컴파일하여 실행파일을 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc Practice1.c -o result5
```

size 명령어를 통해 result5 의 세그먼트 크기 값을 확인하고 분석한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result5
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __data: 8
        total 8
Segment __LINKEDIT: 16384
total 4295016448
```

DATA 세그먼트의 __bss 섹션의 4 바이트 대신 정적 변수에도 값을 부여함으로써 __data 섹션에 추가가 된 모습을 확인할 수 있다. __data 섹션에서는 초기화된 전역 변수와 정적 변수를 저장하기 위해 사용된다는 것을 알 수 있다.

2.2 실습 2 : 오브젝트 파일 수정/링킹 실습 (1)

(0) 실습 환경

macOS 에서 진행한다.

(1) 2.1 실습에서 진행한 result1 실행 파일을 대상으로 ELF 파일 확인

result1 의 ELF 파일 형식을 확인한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % objdump -p result1

result1:        file format mach-o 64-bit x86-64
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
MH_MAGIC_64  X86_64        ALL  0x00     EXECUTE    15        728   NOUNDEFS DYLDLINK TWOLEVEL PIE
```

(2) 'Practice2.c' 파일을 생성한다.

Vim 을 사용하여 Practice2.c 파일을 생성 후 코드를 작성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % vim Practice2.c
```

파일 내용을 수정하기 위해 <i>를 통해 다음 코드를 입력한 후
<ESC + :wq!>를 통해 수정한 파일 내용을 저장 후 종료한다.

```
#include <stdio.h>

int main(){
        printf("hello world!!");
        return 0;
}
```

<Practice2.c 의 코드>

Practice2.o 의 이름으로 컴파일 및 오브젝트 파일을 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc -c Practice2.c
```

Practice2.o 의 ELF 파일 형식을 확인한다.

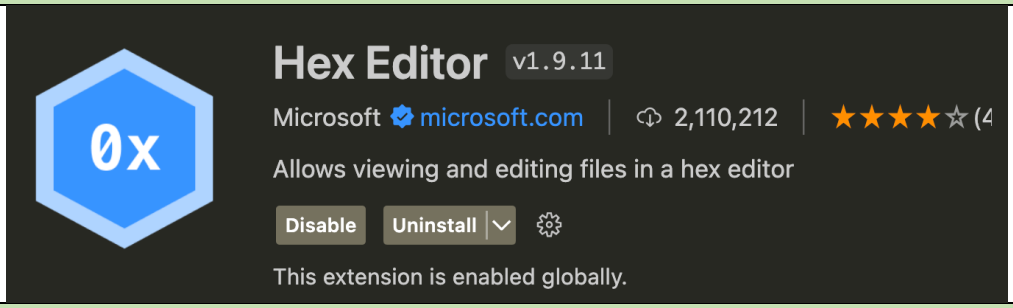
```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % objdump -p Practice2.o

Practice2.o:    file format mach-o 64-bit x86-64
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
MH_MAGIC_64  X86_64        ALL  0x00      OBJECT     4        520 SUBSECTIONS_VIA_SYMBOLS
```

ELF 파일 형식(Executable and Linkable Format)은 Linux, Unix 운영체제에서 사용되는 바이너리 파일 포맷으로 filetype 을 통해 어떤 형태의 바이너리인지 확인할 수 있다. 먼저, Practice1 의 실행파일을 대상으로 ELF filetype 을 확인하면 EXECUTE 로 실행 가능한 프로그램 파일을 의미한다. 이 파일은 직접 실행이 가능하고, 프로그램의 entry point 부터 시작해 코드를 실행한다. 주로 사용자가 실행할 프로그램이나 시스템 서비스를 제공하는 바이너리 파일에서 사용이된다.

반면에, Practice2 를 컴파일하여 ELF filetype 을 확인하면, OBJECT 로 오브젝트 파일임을 확인할 수 있다. 오브젝트 파일은 컴파일된 코드와 데이터가 포함된 이진 형식의 파일이고, 다른 오브젝트 파일과 결합한 실행 파일을 생성하는 데 사용되기도 한다. 오브젝트 파일은 컴파일러가 생성하는 중간 파일로, 라이브러리나 실행 파일을 생성하기 위해 링커가 필요하다는 것을 알 수 있다.

(3) Hex Editor 를 설치, Practice2.o 파일을 열어 정보 분석한다.

VSCode 에 Hex Editor 확장프로그램을 설치한다.



Practice2.o 의 파일을 열어 (commend + shift + P)를 통해 Hex Editor 로 열어 확인한다.

1. ELF 파일 헤더 정보 : ELF 파일 형식과 entry point, 프로그램 헤더와 섹션 헤더의 시작 위치 정보를 확인할 수 있다.
2. 프로그램 헤더 정보 : 세그먼트의 위치와 크기, 권한 등의 정보를 확인할 수 있다.
3. 섹션 정보 : 섹션 이름, 주소와 크기, 플래그 등의 정보를 확인할 수 있다.

(4) 기존 문자열을 'have a look~!'으로 변경하여 저장한다.

기존 문자열을 'have a look~!'으로 변경하여 저장한다.

```
00000210  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000220  00 00 00 00 00 00 00 00 55 48 89 E5 48 83 EC 10  ........UH..H...
00000230  C7 45 FC 00 00 00 00 48 8D 3D 0F 00 00 00 B0 00  .E.....H.=......
00000240  E8 00 00 00 00 31 C0 48 83 C4 10 5D C3 68 65 6C  .....1.H...].hel
00000250  6C 6F 20 77 6F 72 6C 64 21 21 00 00 00 00 00 00  lo world!!......
00000260  00 00 00 00 00 00 00 00 25 00 00 00 00 00 00 01  ........%.......
00000270  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

<Practice2.o 의 문자열 부분>

```
00000230  C7 45 FC 00 00 00 00 48 8D 3D 0F 00 00 00 B0 00  .E.....H.=......
00000240  E8 00 00 00 00 31 C0 48 83 C4 10 5D C3 68 61 76  .....1.H...].hav
00000250  65 20 61 20 6C 6F 6F 6B 7E 21 00 00 00 00 00 00  e a look~!......
00000260  00 00 00 00 00 00 00 00 25 00 00 00 00 00 00 01  ........%.......
```

<Practice2.o 의 수정한 파트>

(5) 수정된 'Practice2.o' 파일을 링킹으로 modified 이름의 실행 파일 생성.

수정된 Practice2.o 파일을 gcc 링킹으로 modified 이름의 실행 파일을 생성 및 실행한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc Practice2.o -o modified
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % ./modified
have a look~!%
```

Hex Editor 를 사용해 오브젝트 파일을 직접 수정하여, 코드나 데이터를 변경할 수 있었다. 이를 통해 오브젝트 파일이 다른 오브젝트 파일이나 함수를 변경하거나 상수 값을 대체할 수 있다는 점을 알 수 있다. 또한, gcc 를 이용한 링킹으로 컴파일을 하게 되면, 실행 파일이 생성되는데, 해당 실행 파일은 수정된 코드 및 데이터가 반영되어 기존 실행 결과와 동작이 다르게 출력된다.

2.3 실습 3 : 오브젝트 파일 수정/링킹 실습 (2)

(0) 실습 환경

macOS 에서 진행한다.

(1) 'smile.c' 프로그램을 생성하고 컴파일한다.

vim 을 통해 smile.c 프로그램 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % vim smile.c
```

smile.c 를 다음과 같이 작성한다.

```
#include <stdio.h>

int main(){
        int a = 85;
        int b = 15;
        int sum = a + b;

        if (sum != (a+b)){
                printf("a+b와  sum값 이  같 습 니 다 :)");
        }
        else {
                printf("a+b와  sum값 이  다 릅 니 다 !!");
        }
        return 0;
}
```

컴파일을 통해 오브젝트 파일을 생성한다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc -c smile.c
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % gcc smile.o -o smile
```

(2) 'smile.o' 를 Hex Editor 로 열어 올바른 출력값이 나오도록 수정한다.

Case1. 조건문의 연산자를 수정하기.

'!=' 연산자를 '=='로 변경하여 올바른 출력값을 나타낼 수 있도록 한다.

변경 내용 : 84 -> 85 (! -> =)

<원본>

```
600 C80F8413 00000048 8D3D2200 0000B000 E8000000 00E90E00 . .    H.="    . .    .
```

<수정>

```
600 C80F8513 00000048 8D3D2200 0000B000 E8000000 00E90E00 . .    H.="    . .    .
```

<결과>

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % ./smile1
a+b와  sum값 이  같 습 니 다 :)%
```

Case2. 출력문 수정하기.

논리 연산자에 맞는 출력문으로 수정한다.

변경 내용 : 20EAB099 EC8AB5EB -> 20EB8BA4 EBA685EB (같습니다->다릅니다)

20EB8BA4 EBA685EB -> 20EAB099 EC8AB5EB(다릅니다->같습니다)

〈원본〉

```
648 612B62EC 99802073 756DEAB0 92EC9DB4 20EAB099 EC8AB5EB  a+b... sum..... .....
672 8B88EB8B A43A2900 612B62EC 99802073 756DEAB0 92EC9DB4  .....:) a+b... sum.....
696 20EB8BA4 EBA685EB 8B88EB8B A4212100 00000000 00000000   ............!!
```

〈수정〉

```
648 612B62EC 99802073 756DEAB0 92EC9DB4 20EB8BA4 EBA685EB  a+b... sum..... .....
672 8B88EB8B A43A2900 612B62EC 99802073 756DEAB0 92EC9DB4  .....:) a+b... sum.....
696 20EAB099 EC8AB5EB 8B88EB8B A4212100 00000000 00000000   ............!!
```

〈결과〉

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % ./smile2
a+b와  sum값 이  같 습 니 다 !!%
```

2.4 실습 4 : SetUID Program 실습

(0) 실습 환경

(1) 1 번 문제

Figure out why "passwd", "chsh", "su", and "sudo" commands need to be Set-UID programs. What will happen if they are not? If you are not familiar with these programs, you should first learn what they can do by reading their manuals. Please copy these commands to your own directory; the copies will not be Set-UID programs. Run the copied programs, and observe what happens.

<문제 요약>

'passwd', 'chsh', 'su', 'sudo' 명령어가 SetUID 프로그램으로 동작해야 하는 이유에 대해 알아보고, 해당 명령어를 디렉토리에 복사한 후, 그 복사본이 SetUID 프로그램이 아닌 경우에 어떻게 동작하는지 확인하기.

```
kangdonghee@ubuntu:~/Desktop$ which passwd
/usr/bin/passwd
kangdonghee@ubuntu:~/Desktop$ ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 68208 Nov 29 03:53 /usr/bin/passwd
kangdonghee@ubuntu:~/Desktop$ cp /usr/bin/passwd /tmp/
kangdonghee@ubuntu:~/Desktop$ ls -al /tmp/passwd
-rwxr-xr-x 1 kangdonghee kangdonghee 68208 Apr  9 21:30 /tmp/passwd
```

[풀이]

"passwd" 명령어를 실행하면 현재 로그인한 사용자의 암호를 변경할 수 있다. 그러나 암호를 변경하려면 /etc/shadow 파일에 액세스해야 한다. 이 파일은 root 권한으로만 액세스할 수 있으므로, "passwd" 명령어는 일반 사용자의 암호를 변경할 수 있도록 Set-UID root 로 설정된다. 따라서 "passwd" 명령어는 root 권한으로 실행될 수 있도록 Set-UID 프로그램으로 동작해야 한다.

"chsh" 명령어는 현재 로그인한 사용자의 셸을 변경하는 데 사용한다. 이 작업은 /etc/passwd 파일을 수정할 권한이 필요로 한다. 그러므로 "chsh" 명령어는 이 파일을 수정할 수 있도록 Set-UID root 로 설정된다.

"su" 명령어는 다른 사용자로 로그인하는 데 사용된다. 일반 사용자는 일반적으로 다른 사용자로 로그인할 수 없지만, root 권한을 가진 사용자는 "su" 명령어를 사용하여 다른 사용자로 전환할 수 있다. 이때 "su" 명령어는 전환할 사용자의 권한으로 실행되도록 Set-UID root 로 설정된다.

"sudo" 명령어는 특정 명령어나 스크립트를 root 권한으로 실행할 수 있도록 허용한다. 이때 "sudo" 명령어는 root 권한으로 실행될 수 있도록 Set-UID root 로 설정된다.
만약 이들 명령어가 Set-UID 프로그램이 아닐 경우, 일반 사용자가 이들을 실행하더라도 해당 명령어에서 필요한 파일 또는 리소스에 액세스할 수 없으므로 명령어가 실패한다. 또한, 이들 명령어를 복사하여 실행하면, 일반 사용자의 권한으로 실행되므로 필요한 작업을 수행할 수 없다.

(2) 2 번 문제

(a) Login as root, copy /bin/zsh to /tmp, and make it a set-root-uid program with permission 4755. Then login as a normal user, and run /tmp/zsh. Will you get root privilege? Please describe your observation. If you cannot find /bin/zsh in your operating system, please use the following command to install it.

<문제 요약>

Linux 에서 root 로 로그인하여 /bin/zsh 를 /tmp 로 복사하고, 권한 4755 로 Set-root-UID 프로그램으로 만든 다음, 일반 사용자로 로그인하여 /tmp/zsh 를 실행한다. 이렇게 하면 root 권한을 얻을 수 있는지 여부를 확인하고, 만약 시스템에서 /bin/zsh 를 찾을 수 없는 경우, 설치 명령을 사용한다.

```
root@ubuntu:/# cd /tmp/
root@ubuntu:/tmp# sudo su
root@ubuntu:/tmp# cp /usr/bin/zsh /tmp/
root@ubuntu:/tmp# chmod u+s zsh
root@ubuntu:/tmp# ls -al zsh
-rwsr-xr-x 1 root root 878288 Apr  9 21:32 zsh
root@ubuntu:/tmp# exit
exit
root@ubuntu:/tmp# exit
exit
kangdonghee@ubuntu:/tmp$ ./zsh
ubuntu# id
uid=1000(kangdonghee) gid=1000(kangdonghee) euid=0(root) groups=1000(kangdonghee),4(adm),24(cdrom),27(
sudo),30(dip),46(plugdev),120(lpadmin),132(lxd),133(sambashare)
```

[풀이]

Root 권한으로 /bin/zsh 파일을 Set-root-UID 프로그램으로 만들어서, /tmp/zsh 를 실행할 때 root 권한을 얻을 수 있는지 확인한다.

(1) cp 명령어를 통해 /bin/zsh 를 /tmp/ 디렉토리로 복사한다.

(2) 복사한 /tmp/zsh 파일을 Set-root-UID 프로그램으로 만든다. (chmod u+s zsh)

(3) 로컬(일반사용자로)로 로그인해서, ./zsh 를 실행한다.

root 권한을 얻게 된다. 이를 통해 쉘 프로그램이 가진 모든 권한을 얻게 될 수 있어 보안 문제를 발생시킬 수 있다는 점을 알 수 있다.

(b) Instead of copying /bin/zsh, this time, copy /bin/bash to /tmp, make it a set-root-uid program. Run /tmp/bash as a normal user. Will you get root privilege? Please describe and explain your observation.

<문제 요약>

/bin/bash 파일을 /tmp/ 디렉토리로 복사하고, 해당 파일을 set-root-UID 프로그램으로 만든 후 일반 user 가 /tmp/bash 를 실행했을 때 Root 권한을 얻을 수 있는지 여부를 확인한다.

```
kangdonghee@ubuntu:~/Desktop$ cd /tmp/
kangdonghee@ubuntu:/tmp$ sudo su
[sudo] password for kangdonghee:
root@ubuntu:/tmp# cp /bin/bash /tmp/
root@ubuntu:/tmp# chmod u+s bash
root@ubuntu:/tmp# exit
exit
kangdonghee@ubuntu:/tmp$ ls -al bash
-rwsr-xr-x 1 root root 1183448 Apr  9 21:34 bash
kangdonghee@ubuntu:/tmp$ ./bash
bash-5.0$ id
uid=1000(kangdonghee) gid=1000(kangdonghee) groups=1000(kangdonghee),4(adm),24(cdrom),27(sudo),30(dip)
,46(plugdev),120(lpadmin),132(lxd),133(sambashare)
```

[풀이]

(1) root 로 로그인 후 /bin/bash 파일을 /tmp 디렉토리로 복사한다.

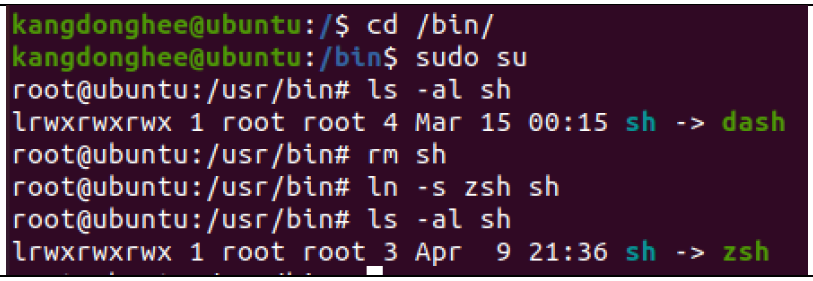
(2) 복사한 /tmp/bash 파일을 set-root-UID 프로그램으로 만든다. (chmod u+s bash)

(3) exit 후 일반 user 에서 ./bash 를 실행한다.
이 문제에서는 /bin/bash 파일이 Set-root-UID 프로그램으로 설정되어도 root 권한을 얻을 수 없다는 점을 확인하는 문제이다.

(3) 3 번 문제 (Setup for the rest of the tasks)

As you can find out from the previous task, /bin/bash has certain built-in protection that prevent the abuse of the Set-UID mechanism. To see the life before such a protection scheme was implemented, we are going to use a different shell program called /bin/zsh. In some Linux distributions (such as Fedora and Ubuntu), /bin/sh is actually a symbolic link to /bin/bash. To use zsh, we need to link /bin/sh to /bin/zsh. The following instructions describe how to change the default shell to zsh.

<문제 요약>
/bin/bash 에는 Set-UID 메커니즘을 남용하는 것을 방지하는 내장 보호 기능이 있기 때문에, /bin/zsh 를 사용하여 Set-UID 메커니즘을 남용할 수 있는 상황을 재현한다.



<풀이>
/bin/zsh 가 기본 쉘로 설정한다. 이전 문제와 같이 /bin/zsh 를 /tmp 디렉토리로 복사하고, 해당 파일을 Set-root-UID 프로그램으로 만든 후 일반 사용자로 로그인한 후 /tmp/zsh 를 실행하여 Set-UID 메커니즘을 남용할 수 있는지 확인했다.

(4) 4 번 문제 **The PATH environment variable**

The system(const char *cmd) library function can be used to execute a command within a program. The way system(cmd) works is to invoke the /bin/sh program, and then let the shell program to execute cmd. Because of the shell program invoked, calling system() within a Set-UID program is extremely dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are under user's control. By changing these variables, malicious users can control the behavior of the Set-UID program.

<문제 요약>
시스템 함수(system())를 사용해 프로그램 내에서 명령을 실행하는 방법과 Set-UID 프로그램에서 이 함수를 사용할 때 발생할 수 있는 문제에 대해 알아본다.

(a) Can you let this Set-UID program (owned by root) run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations.

<문제 요약>

Set-UID 프로그램에서 우리가 작성한 코드를 실행시킬 수 있는지, 그리고 실행시켰을 때 해당 코드가 root 권한으로 실행되는지 여부를 확인한다.

```
kangdonghee@ubuntu:/tmp$ vi system.c
kangdonghee@ubuntu:/tmp$ cat system.c

#include<stdlib.h>
int main()
{
        system("ls");
        return 0;
}
kangdonghee@ubuntu:/tmp$ sudo su
root@ubuntu:/tmp# gcc -o system system.c
root@ubuntu:/tmp# chmod u+s system
root@ubuntu:/tmp# exit
exit
kangdonghee@ubuntu:/tmp$ cp /bin/sh /tmp/ls
```

```
kangdonghee@ubuntu:/tmp$ ./system
bash
config-err-OboLoB
ls
passwd
snap-private-tmp
ssh-q8elgnjoKwt2
system
system.c
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-colord.service-S4aDri
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-ModemManager.service-SnA6Ug
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-switcheroo-control.service-7wtApi
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-systemd-logind.service-e7hy0g
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-systemd-resolved.service-z3dqzf
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-systemd-timesyncd.service-LxKrHh
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-upower.service-ePXDlj
tracker-extract-files.1000
tracker-extract-files.125
VMwareDnD
vmware-root_739-4248680507
zsh
kangdonghee@ubuntu:/tmp$ id
uid=1000(kangdonghee) gid=1000(kangdonghee) groups=1000(kangdonghee),4(adm),24(cdrom),27(sudo),30(dip)
,46(plugdev),120(lpadmin),132(lxd),133(sambashare)
```

<풀이>

root 권한으로 실행되지 않는다. 코드가 Set-UID 프로그램에서 실행되더라도, 해당 코드는 Set-UID 프로그램에서 사용되는 root 권한만을 사용할 수 있다는 것을 알 수 있다.

(b) Now, change /bin/sh so it points back to /bin/bash, and repeat the above attack. Can you still get the root privilege? Describe and explain your observations.

<문제 요약>

/bin/sh 를 /bin/bash 로 변경한 후 (a)와 같은 방법으로 시도하여 root 권한을 얻을 수 있는지 확인한다.

```
kangdonghee@ubuntu:/tmp$ sudo su
root@ubuntu:/tmp# cd /bin
root@ubuntu:/bin# rm sh
root@ubuntu:/bin# ln -s bash sh
root@ubuntu:/bin# ls -al sh
lrwxrwxrwx 1 root root 4 Apr  9 21:50 sh -> bash
root@ubuntu:/bin# exit
exit
```

```
kangdonghee@ubuntu:/tmp$ ./system
bash
config-err-OboLoB
ls
passwd
snap-private-tmp
ssh-q8elgnjoKwt2
system
system.c
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-colord.service-S4aDri
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-fwupd.service-vB0yNi
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-ModemManager.service-SnA6Ug
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-switcheroo-control.service-7wtApi
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-systemd-logind.service-e7hy0g
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-systemd-resolved.service-z3dqzf
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-systemd-timesyncd.service-LxKrHh
systemd-private-b5eddb25ad7548738f2ac1e43015b4be-upower.service-ePXDlj
tracker-extract-files.1000
tracker-extract-files.125
VMwareDnD
vmware-root_739-4248680507
zsh
kangdonghee@ubuntu:/tmp$ id
uid=1000(kangdonghee) gid=1000(kangdonghee) groups=1000(kangdonghee),4(adm),24(cdrom),27(sudo),30(dip)
,46(plugdev),120(lpadmin),132(lxd),133(sambashare)
```

〈풀이〉

Set-UID 프로그램에서 실행되더라도, 프로그램에서 사용되는 root 권한만 사용할 수 있었다.

(5) 5 번 문제

Bob works for an auditing agency, and he needs to investigate a company for a suspected fraud. For the investigation purpose, Bob needs to be able to read all the files in the company's Unix system; on the other hand, to protect the integrity of the system, Bob should not be able to modify any file. To achieve this goal, Vince, the superuser of the system, wrote a special set-root-uid program (see blow), and then gave the executable permission to Bob. This program requires Bob to type a file name at the command line, and then it will run /bin/cat to display the specified file. Since the program is running as a root, it can display any file Bob specifies. However, since the program has no write operations, Vince is very sure that Bob cannot use this special program to modify any file.

〈문제요약〉

Bob 은 회사의 Unix 시스템에 있는 모든 파일을 읽어야 한다. 그러나 시스템의 무결성을 보호하기 위해 Bob 은 어떤 파일도 수정할 수 없어야 한다. 이를 위해 시스템 슈퍼유저인 Vince 는 특수한 set-root-uid 프로그램을 작성하여 Bob 에게 실행 권한을 부여한다. 이 프로그램은 Bob 이 명령행에서 파일 이름을 입력하면 /bin/cat 을 실행하여 지정된 파일을 표시한다. 이 프로그램은 root 로 실행되기 때문에 Bob 이 지정한 모든 파일을 표시할 수 있다. 그러나 프로그램에는 쓰기 작업이 없기 때문에 Vince 는 Bob 이 이 특별한 프로그램을 사용하여 어떤 파일도 수정할 수 없다고 확신한다는 가정하에 실습을 진행한다.

(a) Set q=0 in the program. This way, the program will use system() to invoke the command. Is this program safe? If you were Bob, can you compromise the integrity of the system? For example, can you remove any file that is not writable to you? (Hink: remember that system() actually invokes /bin/sh, and then runs the command within the shell environment. We have tried

the environment variable in the previous task: here let us try a different attack. Please pay attention to the special characters used in a normal shell environment).

〈문제 요약〉

이 프로그램에서 q=0 으로 설정할 경우, 시스템()을 사용하여 명령을 실행한다. 이 프로그램은 안전한지, 시스템의 무결성을 침해할 수 있는지 확인한다.

```
root@ubuntu:/tmp# vi SEC.c
root@ubuntu:/tmp# cat SEC.c
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
        char *v[3];
        if(argc < 2)
        {
        printf("Please type a file name.");
        return 1;
        }
        v[0] = "/bin/cat";
        v[1] = argv[1];
        v[2] = 0;

        int q = 0;
        if (q == 0)
        {
        char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
        sprintf(command, "%s %s", v[0], v[1]);
        system(command);

        }else{
        execve(v[0], v, 0);
        }
        return 0;
}
```

```
root@ubuntu:/bin# ls -al /bin/sh
lrwxrwxrwx 1 root root 4 Apr  9 21:50 /bin/sh -> bash
root@ubuntu:/bin# cd /tmp/
root@ubuntu:/tmp# sudo su
root@ubuntu:/tmp# gcc -o SEC SEC.c
root@ubuntu:/tmp# chmod u+s SEC
root@ubuntu:/tmp# exit
exit
```

```
root@ubuntu:/tmp# ls -al file SEC
-rw-r--r-- 1 root root     0 Apr  9 22:21 file
-rwsr-xr-x 1 root root 16952 Apr  9 22:19 SEC
root@ubuntu:/tmp# ./SEC "file;mv file file_new"
root@ubuntu:/tmp# ls file*
file_new
```

〈풀이〉

시스템 명령어를 실행하는 데 있어서 쓰기 권한이 없는 파일을 삭제할 수 있다. 또한, 일반적인 shell 환경에서 사용되는 특수 문자를 사용하면 보안 문제가 발생할 수 있었다.

(b) Set q=1 in the program. This way, the program will use execve() to invoke the command. Do your attack in task (a) still work? Please describe and explain your observations.

<문제 요약>
q=1 으로 설정할 경우 execve()를 사용하여 명령을 실행한다. (a)와 같은 동작을 하는지 확인하기.

```
root@ubuntu:/tmp# gcc -o SEC2 SEC.c
root@ubuntu:/tmp# chmod u+s SEC2
root@ubuntu:/tmp# exit
exit
root@ubuntu:/tmp# ./SEC2 "file;mv file file_new2"
/bin/cat: file: No such file or directory
mv: cannot stat 'file': No such file or directory
root@ubuntu:/tmp# ls file*
file_new
```

<풀이>
execve()는 명령어를 실행하기 이전에 파일의 경로와 인수를 전달해야 한다. 따라서 이 함수는 shell 을 사용하지 않으므로, 일반적인 환경에서 사용되는 특수문자를 사용한 보안 공격을 불가능하게 된다. 그리고, 쓰기 권한이 있는 파일을 삭제하는 등의 공격은 가능하게 된다.

(6) 6 번 문제

The LD_PRELOAD environment variable. To make sure Set-UID programs are safe from the manipulation of the LD_PRELOADenvironment variable, the runtime linker (ld.so) will ignore this environment variable if the program is a Set-UID root program, except for some conditions. We will figure out what these conditions are in this task.

<문제 요약>
LD_PRELOAD 환경 변수에 대해. Set-UID 프로그램이 LD_PRELOAD 환경 변수의 조작으로부터 안전하도록 하기 위해, 실행시간 링커(ld.so)는 해당 프로그램이 Set-UID root 프로그램인 경우에만, 일부 조건을 제외하고 LD_PRELOAD 환경 변수를 무시하는데. 이러한 조건을 확인해본다.

(a) Let us buid a dynamic link library. Create the following program, and name it mylib.c. It basically overrides the sleep()function in libc:

```
kangdonghee@ubuntu:/tmp$ vi mylib.c
kangdonghee@ubuntu:/tmp$ cat mylib.c
#include <stdio.h>
void sleep (int s)
{
        printf("I am not sleeping!\n");
}
```

<풀이>
동적 링크 라이브러리를 작성하고 mylib.c라는 이름으로 저장한다. libc의 sleep() 함수를 재정의했다.

(b) We can compile the above program using the following commands (in the -Wl argument, the third character is l, not one; in the -lc argument, the second

character is l):

```
kangdonghee@ubuntu:/tmp$ sudo su
root@ubuntu:/tmp# gcc -fPIC -g -c mylib.c
root@ubuntu:/tmp# gcc -shared -Wl,-soname,libmylib.so.1 -o libmylib.so.1.0.1 mylib.o -lc
```

〈풀이〉

이전 문제에서 작성한 mylib.c 파일을 컴파일하여 mylib.so라는 동적 링크 라이브러리를 생성하기 위해서는 위와 같은 명령어를 사용했다.

(c) Now, set the LD_PRELOAD environment variable:

```
root@ubuntu:/tmp# export LD_PRELOAD=./libmylib.so.1.0.1
root@ubuntu:/tmp#
```

〈풀이〉

이전에 생성한 mylib.so 동적 링크 라이브러리를 사용하려면, LD_PRELOAD 환경 변수를 설정해주기 위해 위와 같은 명령어를 사용한다.

(d) Finally, compile the following program myprog (put this program in the same directory as libmylib.so.1.0.1):

```
root@ubuntu:/tmp# vi myprog.c
root@ubuntu:/tmp# cat myprog.c
int main()
{
        sleep(1);
        return 0;
}
```

〈1〉 일반 User 권한으로 실행

```
kangdonghee@ubuntu:/tmp$ export LD_PRELOAD=./libmylib.so.1.0.1
kangdonghee@ubuntu:/tmp$ echo $LD_PRELOAD
./libmylib.so.1.0.1
kangdonghee@ubuntu:/tmp$ gcc -o myprog myprog.c
/usr/bin/ld: cannot open output file myprog: Permission denied
collect2: error: ld returned 1 exit status
```

〈2〉 root 권한으로 실행

```
root@ubuntu:/tmp# export LD_PRELOAD=./libmylib.so.1.0.1
root@ubuntu:/tmp# gcc -o myprog myprog.c
root@ubuntu:/tmp# chmod u+s myprog
root@ubuntu:/tmp# exit
exit
kangdonghee@ubuntu:/tmp$ ./myprog
kangdonghee@ubuntu:/tmp$
```

〈3〉 Set-UID root 권한으로 실행

```
kangdonghee@ubuntu:/tmp$ sudo su
root@ubuntu:/tmp# export LD_PRELOAD=./libmylib.so.1.0.1
root@ubuntu:/tmp# gcc -o myprog myprog.c
root@ubuntu:/tmp# chmod u+s myprog
root@ubuntu:/tmp# ./myprog
I am not sleeping!
```

<풀이>
Set-UID root 권한으로 실행되는 프로그램은 보안상 위험할 수 있다. Root 권한으로 실행되기 때문에 공격자가 악용하여 시스템을 손상시킬 수 있다는 점을 확인할 수 있다.
또한, LD_PRELOAD 환경 변수를 사용해 Set-UID root 프로그램의 동작을 변경할 수 있다는 점을 확인할 수 있다.

(7) 7 번 문제

Relinquishing privileges and cleanup. To be more secure, Set-UID programs usually call setuid() system call to permanently relinquish their root privileges. However, sometimes, this is not enough. Compile the following program, and make the program a set-root-uid program. Run it in a normal user account, and describe what you have observed. Will the file /etc/zzz be modified? Please explain your observation.

<문제 요약>
이 문제에서는 다음 프로그램을 컴파일하여 Set-UID root 권한으로 실행시키고, 일반 사용자 계정에서 실행했을 때 어떤 일이 일어나는지 확인해보기.

```
kangdonghee@ubuntu:/tmp$ vi test
kangdonghee@ubuntu:/tmp$ cat test
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
void main()
{
        int fd;
        fd = open('/etc/zzz/, O_RDWR | O_APPEND);
        if(fd==-1) {
                printf("Cannot open /etc/zzz\n");
                exit(0);
        }
        sleep(1);

        setuid(getuid());
        if(fork()) {
                close(fd);
                exit(0);
        } else {
                write(fd, "Malicious Data", 14);
                close(fd);
        }
}
```

```
kangdonghee@ubuntu:/tmp$ sudo su
root@ubuntu:/tmp# gcc -o test test.c
root@ubuntu:/tmp# chmod u+s test
root@ubuntu:/tmp# exit
exit
```

```
kangdonghee@ubuntu:/tmp$ ls -al zzz test
ls: cannot access 'zzz': No such file or directory
-rwsr-xr-x 1 root root 17040 Apr  9 22:54 test
```

<풀이>
일반 사용자 계정으로는 /etc/zzz 파일에 접근할 수 없었다. 따라서 Set-UID root 권한으로 프로그램을 실행되더라도, 일반 사용자 계정으로 실행하면 /etc/zzz 파일을 수정할 수 없다는 것을 확인할 수 있다.

3 결 론

3.1 실습 결과 및 분석

1) 실습 1 : 변수 선언과 초기화에 따른 세그먼트 크기 분석

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result1
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __LINKEDIT: 16384
total 4295000064
```

먼저, 실행 파일의 세그먼트 크기를 확인하기 위해 size -m <실행파일>을 통해 확인할 수 있으며, Segment __TEXT는 16,384KB이며, 세부적으로는 text 15바이트, unwind_info(예외처리 및 함수 호출을 위한 정보를 제공하는 섹션)은 72바이트임을 알 수 있다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result2
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __common: 4
        total 4
Segment __LINKEDIT: 16384
total 4295016448
```

TEXT 세그먼트의 크기는 16,384KB 이며, 전역 변수를 선언하면서 DATA 세그먼트의 크기도 확인할 수 있었다. 그 중 __common 섹션은 전역 범위에서 선언된 변수 중 초기화를 하지 않은 경우에 해당한다. 이 섹션은 해당 변수(global, 4바이트)가 사용될 때 메모리에 할당된다는 것을 알 수 있다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result3
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __bss: 4
        Section __common: 4
        total 8
Segment __LINKEDIT: 16384
total 4295016448
```

__DATA 세그먼트에서 기존 세그먼트와의 차이점을 확인할 수 있다. 전역 변수는 그대로 4 바이트로 유지되며, __bss가 4바이트로 새로 추가되었다. 초기화되지 않은 정적 변수의 크기를 확인할 수 있다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result4
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __data: 4
        Section __bss: 4
        total 8
Segment __LINKEDIT: 16384
total 4295016448
```

Result4 는 전역변수를 4 로 초기화하여 컴파일을 하여 __DATA 세그먼트의 변화를 확인할 수 있다. 기존 __common 섹션이 4바이트로 할당되었었지만, 값을 부여하여 __data 섹션으로 4바이트가 할당 되었다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % size -m result5
Segment __PAGEZERO: 4294967296
Segment __TEXT: 16384
        Section __text: 15
        Section __unwind_info: 72
        total 87
Segment __DATA: 16384
        Section __data: 8
        total 8
Segment __LINKEDIT: 16384
total 4295016448
```

DATA 세그먼트의 __bss 섹션의 4 바이트 대신 정적 변수에도 값을 부여함으로써 __data 섹션에 추가가 된 모습을 확인할 수 있다. __data 섹션에서는 초기화된 전역 변수와 정적 변수를 저장하기 위해 사용된다는 것을 알 수 있다.

2) 실습 2 : 오브젝트 파일 수정 및 링킹(Linking) 실습 1

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % objdump -p Practice2.o

Practice2.o:    file format mach-o 64-bit x86-64
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
MH_MAGIC_64  X86_64        ALL  0x00      OBJECT     4        520 SUBSECTIONS_VIA_SYMBOLS
```

ELF 파일 형식(Executable and Linkable Format)은 Linux, Unix 운영체제에서 사용되는 바이너리 파일 포맷으로 filetype 을 통해 어떤 형태의 바이너리인지 확인할 수 있다. 먼저, Practice1 의 실행파일을 대상으로 ELF filetype을 확인하면 EXECUTE로 실행 가능한 프로그램 파일을 의미한다. 이 파일은 직접 실행이 가능하고, 프로그램의 entry point 부터 시작해 코드를 실행한다. 주로 사용자가 실행할 프로그램이나 시스템 서비스를 제공하는 바이너리 파일에서 사용이된다.

Practice2 를 컴파일하여 ELF filetype 을 확인하면, OBJECT 로 오브젝트 파일임을 확인할 수 있다. 오브젝트 파일은 컴파일된 코드와 데이터가 포함된 이진 형식의 파일이고, 다른 오브젝트 파일과 결합한 실행 파일을 생성하는 데 사용되기도 한다. 오브젝트 파일은 컴파일러가 생성하는 중간 파일로, 라이브러리나 실행 파일을 생성하기 위해 링커가 필요하다는 것을 알 수 있다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % ./modified
have a look~!%
```

Hex Editor 를 사용해 오브젝트 파일을 직접 수정하여, 코드나 데이터를 변경할 수 있었다. 이를 통해 오브젝트 파일이 다른 오브젝트 파일이나 함수를 변경하거나 상수 값을 대체할 수 있다는 점을 알 수 있다. 또한, gcc 를 이용한 링킹으로 컴파일을 하게 되면, 실행 파일이 생성되는데, 해당 실행 파일은 수정된 코드 및 데이터가 반영되어 기존 실행 결과와 동작이 다르게 출력된다.

3) 실습 3 : 오브젝트 파일 수정 및 링킹(Linking) 실습 2

```
600 C80F8413 00000048 8D3D2200 0000B000 E8000000 00E90E00 . .    H.="    . .    .
```

```
600 C80F8513 00000048 8D3D2200 0000B000 E8000000 00E90E00 . .    H.="    . .    .
```

'!=' 연산자를 '=='로 변경하여(84 -> 85) 실행하여 수정할 수 있었다.

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % ./smile1
a+b와  sum값 이  같 습 니 다 :)%
```

```
648 612B62EC 99802073 756DEAB0 92EC9DB4 20EAB099 EC8AB5EB a+b... sum...... ......
672 8B88EB8B A43A2900 612B62EC 99802073 756DEAB0 92EC9DB4 .....:) a+b... sum.....
696 20EB8BA4 EBA685EB 8B88EB8B A4212100 00000000 00000000  ............!!
```

```
648 612B62EC 99802073 756DEAB0 92EC9DB4 20EB8BA4 EBA685EB a+b... sum...... ......
672 8B88EB8B A43A2900 612B62EC 99802073 756DEAB0 92EC9DB4 .....:) a+b... sum.....
696 20EAB099 EC8AB5EB 8B88EB8B A4212100 00000000 00000000  ............!!
```

논리 연산자에 맞는 출력문으로 다음과 같이 수정할 수 있었다.

20EAB099 EC8AB5EB -> 20EB8BA4 EBA685EB (같습니다->다릅니다)

20EB8BA4 EBA685EB -> 20EAB099 EC8AB5EB(다릅니다->같습니다)

```
(base) kangdonghee@gangdonghuiui-MacBookPro week4 % ./smile2
a+b와  sum값 이  같 습 니 다 !!%
```

4) 실습 4 : SetUID Program 실습

[1번] "passwd" 명령어를 실행하면 현재 로그인한 사용자의 암호를 변경할 수 있다. "chsh" 명령어는 현재 로그인한 사용자의 셸을 변경하는 데 사용한다. "su" 명령어는 다른 사용자로 로그인하는 데 사용한다. "sudo" 명령어는 특정 명령어나 스크립트를 root 권한으로 실행할 수 있도록 허용한다. 명령어가 Set-UID 프로그램이 아닐 경우, 일반 사용자가 이들을 실행하더라도 해당 명령어에서 필요한 파일 또는 리소스에 액세스할 수 없으므로 명령어가 실패한다. 또한, 이들 명령어를 복사하여 실행하면, 일반 사용자의 권한으로 실행되므로 필요한 작업을 수행할 수 없는 것을 알 수 있다.
[2번] Linux에서 root로 로그인하여 /bin/zsh를 /tmp로 복사하고, 권한 4755로 Set-root-UID 프로그램으로 만든 다음, 일반 사용자로 로그인하여 /tmp/zsh를 실행한다. 이렇게 하면 root 권한을 얻을 수 있는지 여부를 확인했다. 이를 통해 쉘 프로그램이 가진 모든 권한을 얻게 될 수 있어 보안 문제를 발생시킬 수 있다는 점을 알 수 있다. 또한 /bin/bash 파일이 Set-root-UID 프로그램으로 설정되어도 root 권한을 얻을 수 없다는 점을 확인할 수 있다.
[3번] bin/bash에는 Set-UID 메커니즘을 남용하는 것을 방지하는 내장 보호 기능이 있기 때문에, /bin/zsh를 사용하여 Set-UID 메커니즘을 남용할 수 있는 상황을 재현해보았다.
[4번] 시스템 함수(system())를 사용해 프로그램 내에서 명령을 실행하는 방법과 Set-UID 프로그램에서 이 함수를 사용할 때 발생할 수 있는 문제에 대해서는 root 권한으로 실행되지 않는다. 코드가 Set-UID 프로그램에서 실행되더라도, 해당 코드는 Set-UID 프로그램에서 사용되는 root 권한만을 사용할 수 있다는 것을 알 수 있다. 또한 Set-UID 프로그램에서 실행되더라도, 프로그램에서 사용되는 root 권한만 사용할 수 있다고 확인했다.
[5번] 시스템 명령어를 실행하는 데 있어서 쓰기 권한이 없는 파일을 삭제할 수 있다. 또한, 일반적인 shell 환경에서 사용되는 특수 문자를 사용하면 보안 문제가 발생할 수 있었다. execve()는 명령어를 실행하기 이전에 파일의 경로와 인수를 전달해야 한다. 따라서 이 함수는 shell을 사용하지 않으므로, 일반적인 환경에서 사용되는 특수문자를 사용한 보안 공격을 불가능하게 된다. 그리고, 쓰기 권한이 있는 파일을 삭제하는 등의 공격은 가능하게 된다.
[6번] LD_PRELOAD 환경 변수에 대해. Set-UID 프로그램이 LD_PRELOAD 환경 변수의 조작으로부터 안전하도록 하기 위해, 실행시간 링커(ld.so)는 해당 프로그램이 Set-UID root 프로그램인 경우에만, 일부 조건을 제외하고 LD_PRELOAD 환경 변수를 무시하는데. 이러한 조건을 확인해보았다. Set-UID root 권한으로 실행되는 프로그램은 보안상 위험할 수 있다. Root 권한으로 실행되기 때문에 공격자가 악용하여 시스템을 손상시킬 수 있다는 점을 확인할 수 있다. 또한, LD_PRELOAD 환경 변수를 사용해 Set-UID root 프로그램의 동작을 변경할 수 있다는 점을 확인할 수 있다.
[7번] 프로그램을 컴파일하여 Set-UID root 권한으로 실행시키고, 일반 사용자 계정으로는 /etc/zzz 파일에 접근할 수 없었다. 따라서 Set-UID root 권한으로 프로그램을 실행되더라도, 일반 사용자 계정으로 실행하면 /etc/zzz 파일을 수정할 수 없다는 것을 확인할 수 있다.

3.2 느낀 점

4주차 실습에서는 메모리 세그먼트를 직접 확인하여 운영체제 시스템에서 파일 저장 형태를 확인할 수 있었다. Hex Editor를 통해서 오브젝트 파일을 수정할 때 섹션별로 데이터가 저장되는 점을 확인하는 실습에서 문자열을 직접 변경하는 것은 금방 찾았어도, 조건문의 논리 연산자를 찾기 위해 시간이 오래 걸려 아예 조건문을 수정한 파일을 컴파일해 다른 부분을 찾아서 수정을 했다. SetUID Program 실습을 통해서 리눅스 환경에서의 명령어를 다양하게 찾아보고 적용해보면서 배울 수 있었다. SetUID가의 보안 문제를 알 수 있었다. 시간이 된다면 SetUID에 대해서 이론적으로도 배워보고 싶었다.