Buffer Overflow

공격 실습을 통한 Buffer Overflow 이해

HW 1 (첫 번째 과제)

2022.09.28

TA: 박민수

Email : qkralstn157@naver.com

INDEX

01 소개

소개

❖Buffer Overflow 취약점 실습 및 분석

❖목적

- 지금까지 습득한 지식을 활용.
- 실습을 통해 **Buffer Overflow 취약점 공격 및 방어 기법 학습**
- 스스로 공부하는 능력 향상.
- 문제 해결 능력 향상.

소개

❖ Buffer Overflow 꼭 알아야 하나요?

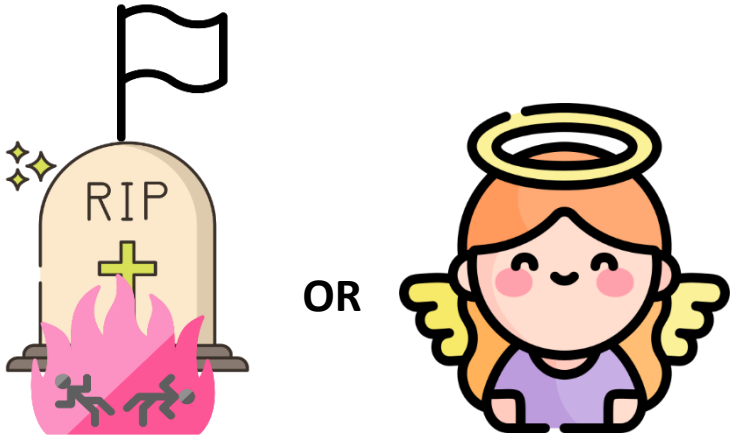
- 과거부터 현재까지 계속 언급되어지는 취약점.
- 다양한 지식 습득 가능.
- 남이 만든 Exploit 코드는 그만.
- 시스템 보안 학습에 기본.
- 그냥 재미있습니다...

소개

❖ 지난 과제(보안 개론)와 난이도 차이

- 문제 난이도는 굉장히 낮음.
- 스스로 공부 안하시면, 매우 높음.

지난 과제(보안 개론)
(Very Easy)



이번 과제
(Case by case)

소개

You can do it! Don’t give up.

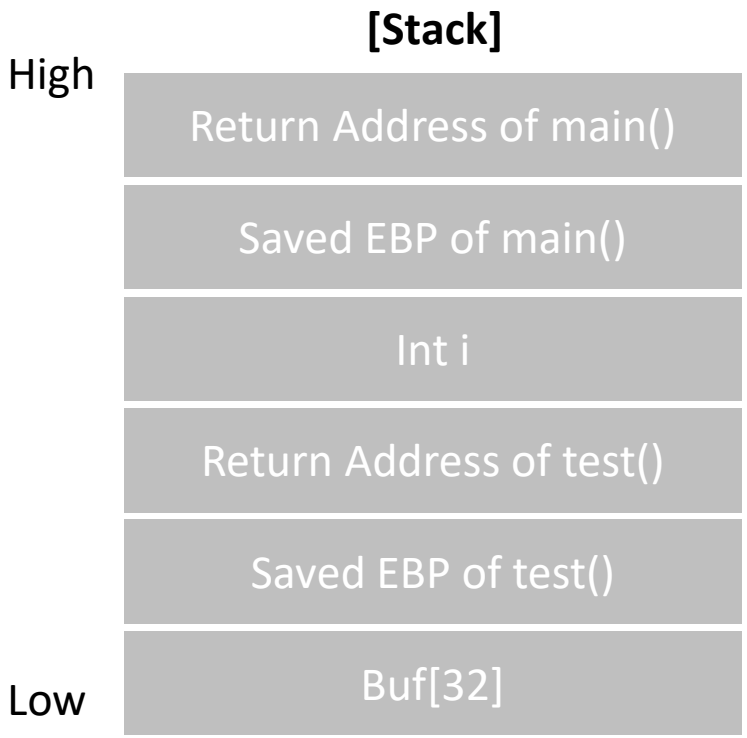
02 배경지식

배경지식

❖ Buffer Overflow란?

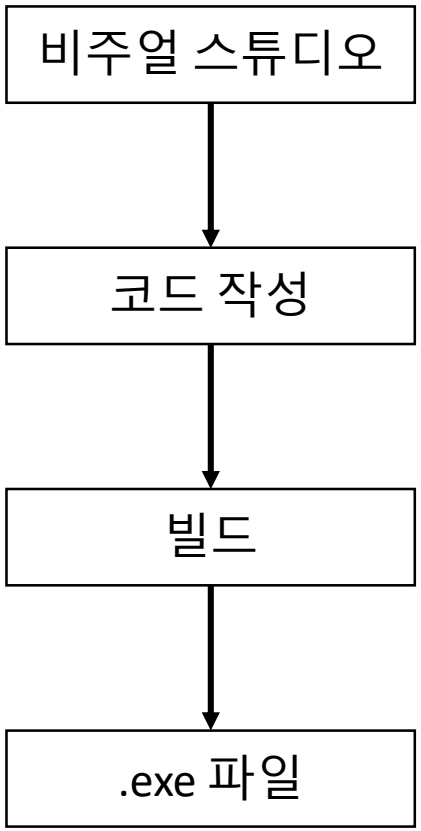
- 컴퓨터에서 메모리 저장 공간인 Buffer와 넘친다는 의미의 Overflow의 합성어이다.
- 즉, 지정된 메모리 공간 보다 초과한 입력을 허용하여 발생하는 취약점을 말한다.
- 참고 자료: CWE-119, CWE-121 등

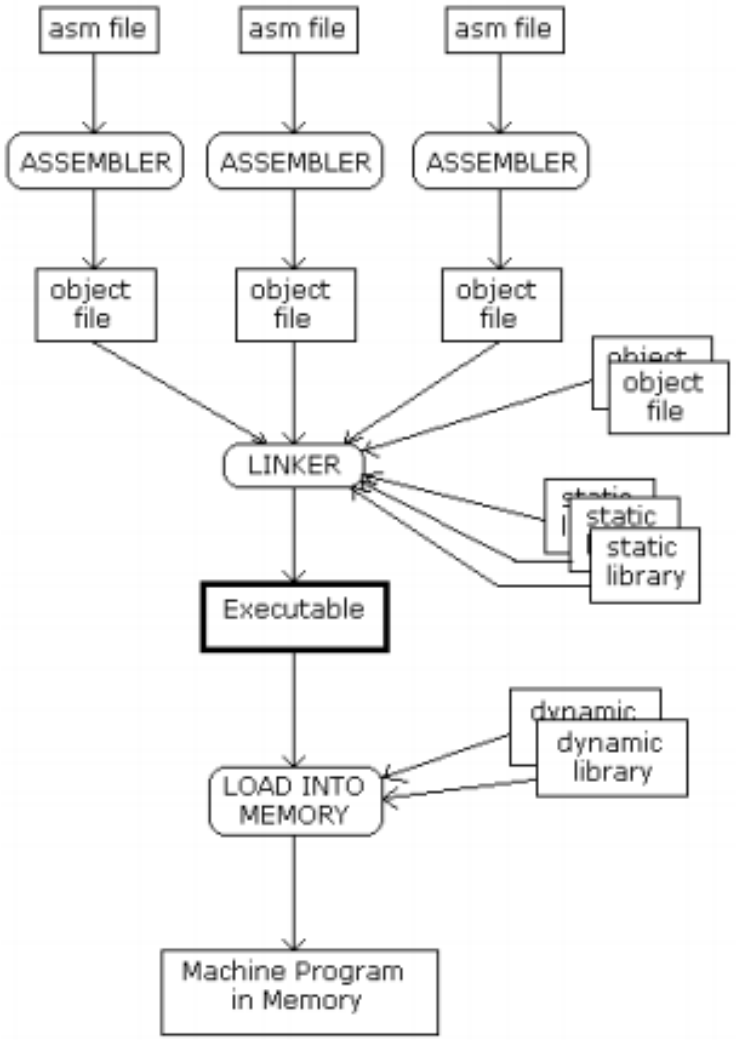
```
void test()
{
        char buf[32];
        scanf("%s", buf);
}

int main()
{
        int i=0x41414141;
        test();
        return 0;
}
```

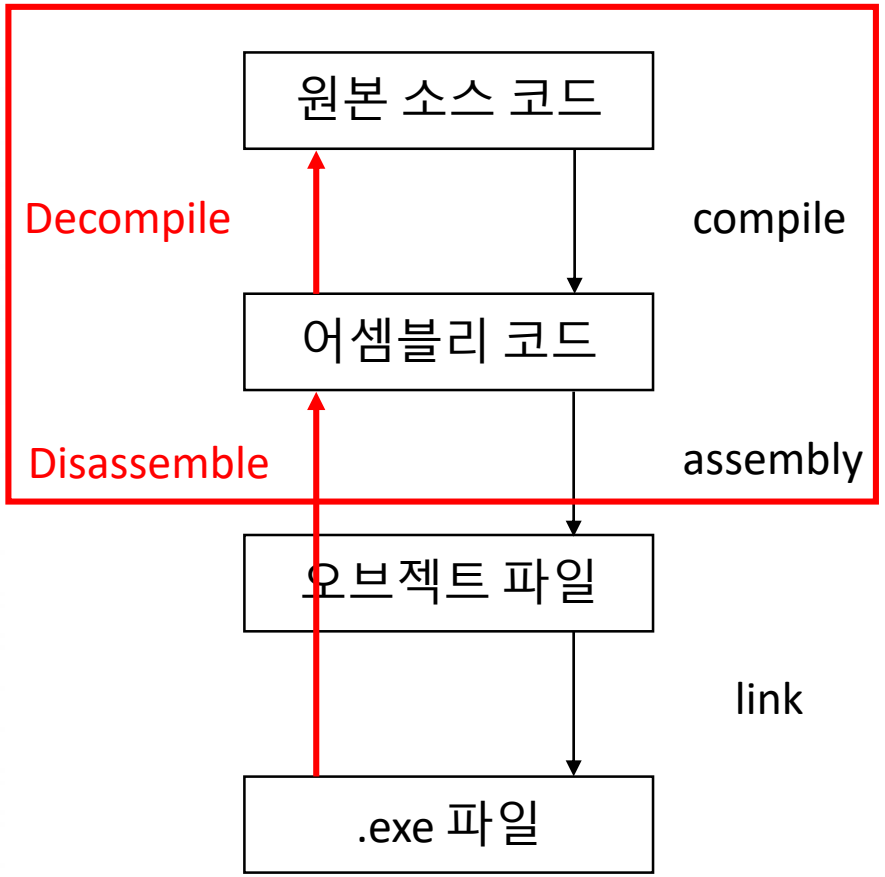


배경지식

❖ 프로그램 제작 과정





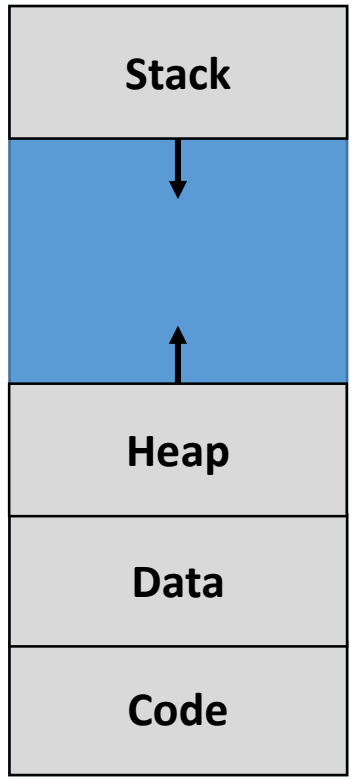


배경지식

❖ 프로세스 메모리 구조

- 운영체제는 사용자 영역과 커널 영역으로 메모리를 나누어 관리.
- 모든 프로그램은 사전에 약속된 형식을 따름.

[프로세스 메모리 구조]



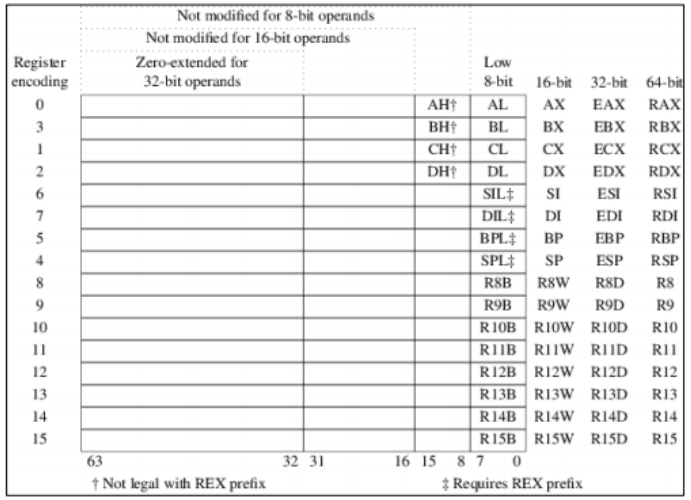
❖ 설명

- 코드영역(.text): 프로그램의 코드(명령어)가 저장되는 영역
- 데이터 영역(.bss/.data): 프로그램에서 사용하는 전역 변수가 저장되는 영역
- 스택 영역(Stack):
  - 함수 호출 시 생성되는 지역 변수와 매개 변수가 저장되는 영역
  - x86의 경우, 보통 높은 주소에서 낮은 주소로 성장
  - LIFO 구조: 나중에 쌓인(push) 프레임(정보)가 먼저 사용(pop)됨
- 힙 영역(Heap):
  - 동적 메모리 할당 시에 사용되는 영역
  - 낮은 주소에서 높은 주소로 성장

배경지식

❖ CPU 레지스터

- 빠른 연산을 위해 사용됨 -
  수행 문맥, 제어 정보, 데이터 등 저장
- 레지스터는 CPU에 내장되어 있어,
  CPU 아키텍처마다 다르다.

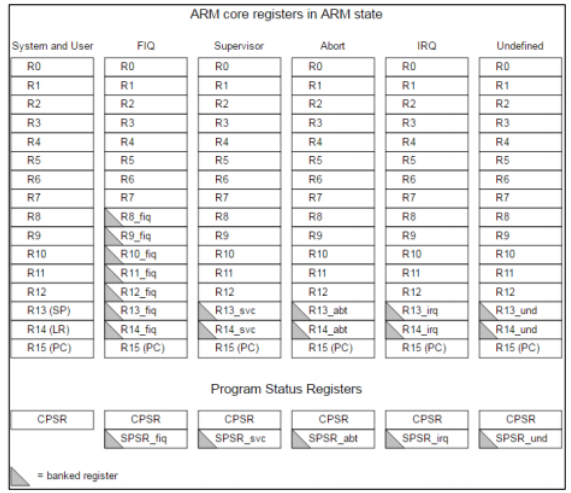


Register encoding	Zero-extended for 32-bit operands	Not modified for 16-bit operands	Not modified for 8-bit operands	Low 8-bit	16-bit	32-bit	64-bit
0			AH†	AL	AX	EAX	RAX
3			BH†	BL	BX	EBX	RBX
1			CH†	CL	CX	ECX	RCX
2			DH†	DL	DX	EDX	RDX
6				SIL‡	SI	ESI	RSI
7				DIL‡	DI	EDI	RDI
5				BPL‡	BP	EBP	RBP
4				SPL‡	SP	ESP	RSP
8				R8B	R8W	R8D	R8
9				R9B	R9W	R9D	R9
10				R10B	R10W	R10D	R10
11				R11B	R11W	R11D	R11
12				R12B	R12W	R12D	R12
13				R13B	R13W	R13D	R13
14				R14B	R14W	R14D	R14
15				R15B	R15W	R15D	R15

63 32 31 16 15 8 7 0

† Not legal with REX prefix ‡ Requires REX prefix

x86 아키텍처의 범용 목적 레지스터



arm 아키텍처의 코어 레지스터

❖명령어 세트(Instruction Sets)

- 아키텍처 별로 다른 명령어와 사용 방법이 존재.
- 데이터 이동: 메모리<->레지스터, 레지스터<->레지스터
- 산술/논리 연산: 더하고 빼기, 비트연산, 소수점 계산 등
- 제어 흐름: 분기, 호출, 반환 등

	x86 32-bit	x86 64-bit	ARM	PowerPC	MIPS
Classification	CISC (variable length)		RISC (4 byte instructions)		
Register Count	8	16	16	32	32
Return Value Register	eax	rax	r0	%r3	$2
First argument	[esp+4]	rdi	r0	%r3	$4
Second argument	[esp+8]	rsi	r1	%r4	$5
Function call	call	call	bl	bl	jalr
Function return	ret	ret	bx lr	blr	jr $31
Stack pointer	esp	rsp	sp	%r1	$29
Constants	5 or $5	5 or $5	#5	5	5
Assembly comment	;	;	@	#	#
Load constant	mov	mov	mov	li	li
Memory Load	mov	mov	ldr	lwz	lw
Memory Store	mov	mov	str	stw	sw
Created by:	Intel	AMD	Acorn	IBM	Stanford
creation year:	1970's	2003	1983	1990's	1980's
Used inside:	All mainstream PCs, original xBox		iPhone, iPad, ...	PS3, xBox 360, old Macs	NU64, old SGI

https://www.cs.uaf.edu/2011/spring/cs641/lecture/02_10_assembly.html

배경지식

❖ 어셈블리어(Assembly Language)

- 기계어와 일대일 대응하고 하드웨어에 종속적인 Low-Level 언어
- 컴퓨터 하드웨어가 이해할 수 있는 기계어를 연산 부호 형식으로 만든 언어
- 컴퓨터(CPU)와의 대화를 위한 대화 규칙

```
extern printf
global main

section .data
        message db "Hello, World!", 10, 0

section .text
main:
        push    dword message
        call printf

        mov eax, 1
        mov ebx, 0
        int 80h
```

배경지식

❖ 함수(Function)란?

- 소프트웨어에서 특정 동작을 수행하는 코드 덩어리 – ex) printf()
- 함수는 '시스템 정의 함수' 와 '사용자 정의 함수' 로 나눌 수 있음.
- 시스템 정의 함수는 운영체제에서 요구하는 형식에 맞추어 사용해야 함.
- 어셈블리 입장에서 볼 때, 함수 이름과 매개변수, 해당 함수를 제공하는 라이브러리 파일이 필요.

❖함수 호출 규약(Calling Convention)

- 호출자가 인수(argument)를 피호출자에게 넘기고, 피호출자의 작업 결과를 반환하는 방법을 정의한 규약
- Ex) stdcall, fastcall, cdecl …

배경지식

❖ GDB(GNU Debugger)

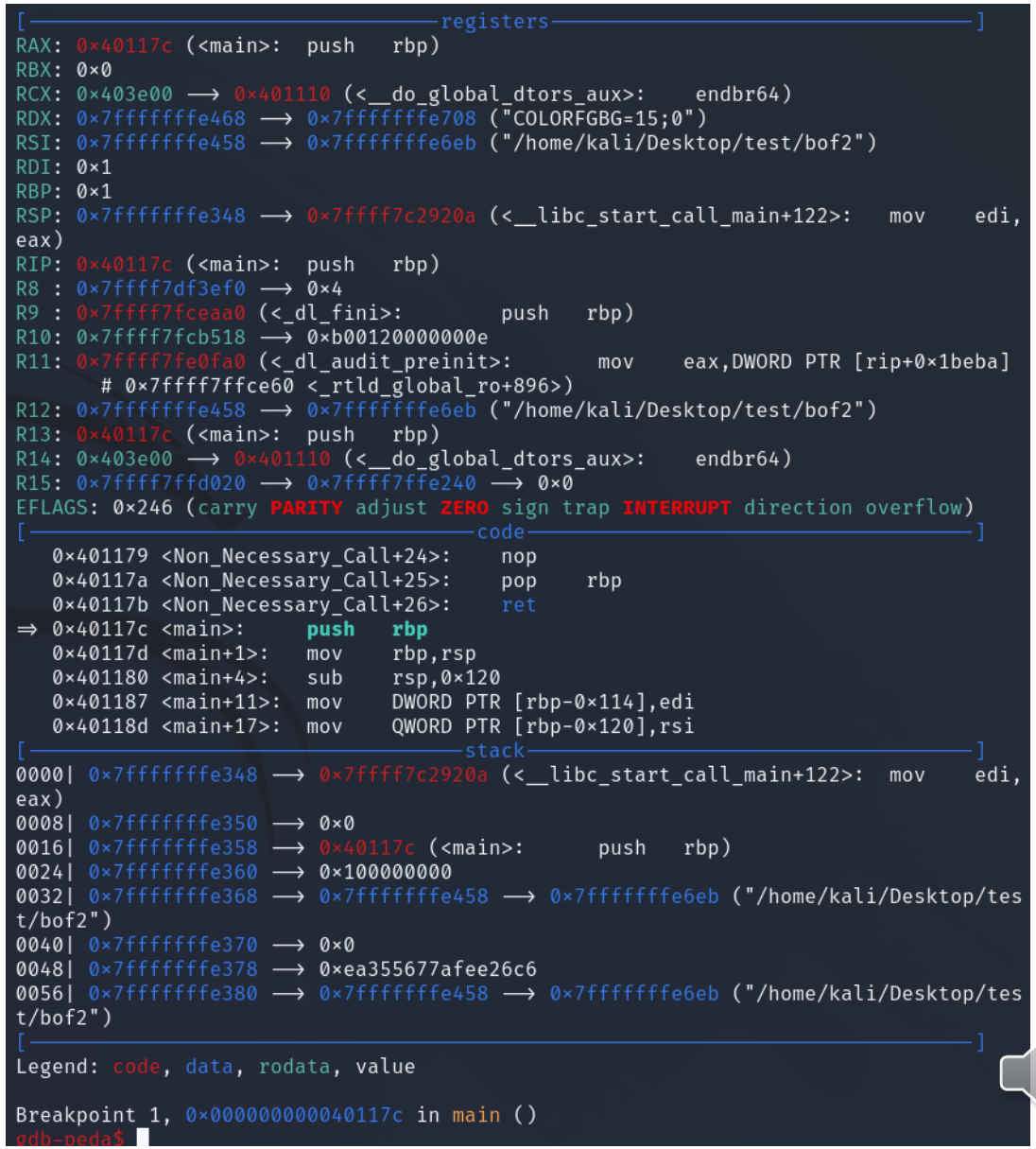
- Debugger: 프로그램을 동적으로 테스트하고 디버깅하는 프로그램.
- 프로그램의 실행 흐름을 추적할 수 있음.
- 메모리 및 변수의 값을 확인하고 이를 변경할 수 있음.

```
root@stud:/home/x86_exam/exam1# gdb -q ./a.out
Reading symbols from ./a.out...(no debugging symbols found)...done.
(gdb) disas main
Dump of assembler code for function main:
   0x08048060 <+0>:     push   $0x10
   0x08048062 <+2>:     push   $0x12345678
   0x08048067 <+7>:     mov    %ebx,%eax
   0x08048069 <+9>:     mov    $0xbffff098,%ebx
   0x0804806e <+14>:    movl   $0x10,(%ebx)
   0x08048074 <+20>:    mov    %eax,%ebx
End of assembler dump.
```

배경지식

❖ GDB-PEDA

- PEDA: Python Exploit Development Assistance for GDB.
- 리눅스 환경에서 binary 분석 및 exploit을 도와주는 도구.
- GDB만 활용하는 것 보다 다양한 기능을 제공.
- 프로그램을 분석하기에 더욱 용이함.
- PEDA를 설치하여 문제를 풀어보는 것을 권장!



배경지식

❖ pwntool in python

- Exploit의 편의성을 위해서 만들어진 툴.
- 주로 CTF 문제 풀이에 많이 사용 중.
- Python을 실행시키고 pwntool을 불러 사용.

❖Pwntool Github

- https://github.com/Gallopsled/pwntools

배경지식

❖ 메모리 보호 기법

- 취약점 자체를 보완하는 것이 아니라 취약점을 통한 공격(Exploit)을 막는 기법.
  - ASLR(Address Space Layout Randomization)
    - 바이너리가 실행될 때마다 데이터 영역(스택, 힙, 라이브러리 등)의 주소를 랜덤으로 변경.
      즉, 실행할 때마다 다른 주소 값을 갖는다.
  - NX-bit (No-eXcute)
    - 실행할 수 있는 메모리 영역에서만 코드를 실행할 수 있도록 만드는 메모리 보호 기법.
  - Stack Canary
    - 함수의 프롤로그에서 SFP(Stack Frame Pointer) 이전에 삽입되어 에필로그에서 해당 값이 변조되는지 검사한다.
      즉, Return Address가 변조되었는지 확인하는 기법.
  - PIE(Position Independent Executable)
    - 데이터 영역과 코드 영역에도 ASLR을 적용.
  - RELRO(RELocation Read-Only)
    - Partial RELRO
      - ELF 바이너리(.ctors, .dtors)들이 프로그램의 데이터 섹션들(.data. bss) 앞으로 오도록 재구성. GOT Writable.
    - Full RELRO
      - 데이터 섹션들(.data, .bss) 영역을 제외한 모든 영역이 Read-Only로 GOT Overwrite가 불가능.

배경지식

❖ **경고..!**

- 배경지식을 **모두 이해**하셔야...
  - **완벽하게 이해하고 풀어갈 수 있습니다**.
- 모르면 풀지 못합니다.

❖해당 지식은 **보안의 기초**입니다.

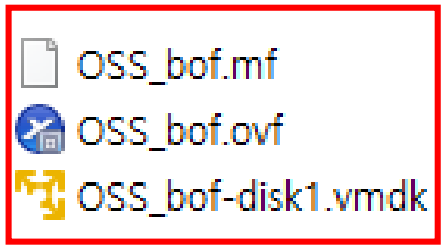
- **스스로 공부**하고 해결하기 위해 노력하시기 바랍니다.
- **해당 과제는 중간고사에서 큰 힘이 될 것입니다**.

03 과제 내용

과제 내용

❖실습 환경

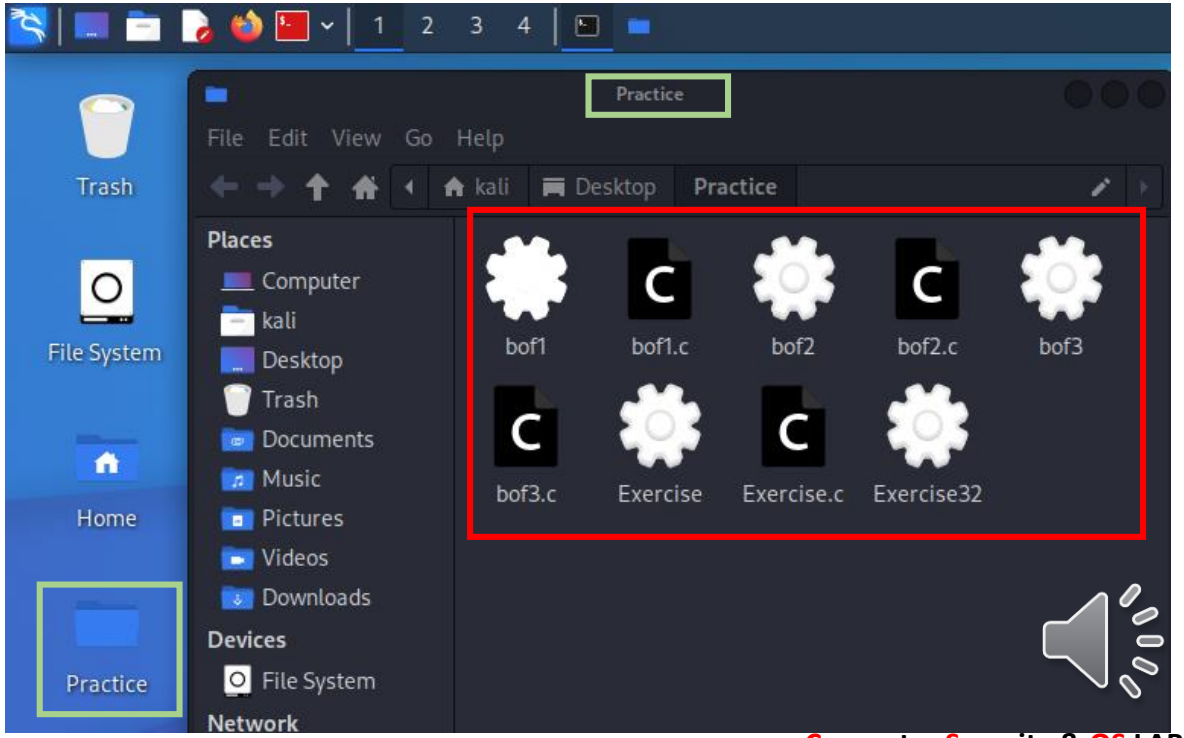
- 제공: 실습 가능한 kali-Linux 환경(필요한 패키지들이 설치되어 있음)
- GDB 설치가 가능하신 분: 따로 설치하여 문제 풀이를 진행해도 상관 없음.



- Practice.zip: https://drive.google.com/file/d/16q2q1lsNAzOE1433Ux-ApOJsU-5aKN35/view?usp=sharing
- OSS_bof.zip(Kali): https://drive.google.com/file/d/1AJtVo4rOLyu3v-iqwDfT4E12cUzE_3Id/view?usp=sharing

❖문제 구성

- 연습 문제 1개
- 과제 문제 3개.
- 문제 풀이는 실행 파일을 사용.
- **소스 코드**는 문제 풀이를 위한 **참고용**으로 따로 **수정할 필요 없음**.



과제 내용

❖실습 환경

- 설치 패키지 목록
  - apt-get update
  - apt-get upgrade
  - apt-get install python
  - apt-get install lib32z1
  - apt-get install python-pip
  - pip install pwntools
  - apt install git
  - apt-get install gdb –y
  - apt-get install gcc
  - apt-get install gcc-multilib

❖메모리 보호기법 해제

- 모든 메모리 보호 기법을 해제하였습니다.

❖Misc

- PEDA 설치
  - git clone https://github.com/longld/peda.git ~/peda
  - echo "source ~/peda/peda.py" >> ~/.gdbinit
- Python 3.10 버전 사용

과제 설명

❖ Exercise.c

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

Int main(int argc, char **argv)
{
        //Can you find these variables in GDB?
        volatile int modifed;
        char buffer[512];

        //You can find zero value in GDB.
        modified = 0;

        //This function is very dangerous function.
        gets(buffer);


        //You can find 'test' Opcode.
        if(modified != 0){
                printf("Good, You have changed correctly!\n");
        }else{
                printf("Try again… You can do this.\n");
        }
}
```

과제 설명

❖ Exercise.c 설명

- 32bit vs 64bit
  - 32bit 시스템과 64bit 시스템에서 컴파일 했을 때, 프로그램의 **차이**를 확인하세요.
  - 프로그램을 실행할 때 사용되는 **레지스터**를 확인하세요.
  - **함수 호출할** 때 **어떤 차이**가 있는지 확인하세요.

❖확인할 내용

- GDB를 사용하여 **변수들의 주소**를 찾으세요.
- ‘modified=0’;, **modified 변수에 0**을 넣는 명령 부분을 찾으세요.
- gets()는 **위험한 함수**로 분류됩니다. 왜 그럴까요?
- **'test’ Opcode**를 찾으세요. 해당 명령어의 역할은 무엇일까요?

과제 설명

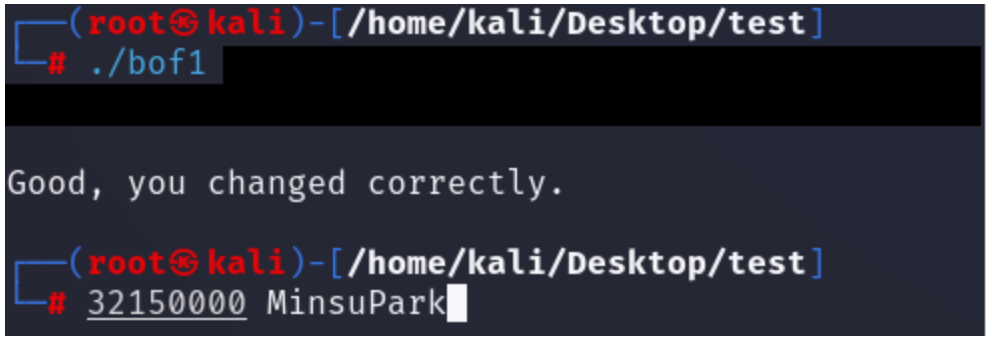
❖ bof1.c

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

Int main(int argc, char **argv)
{
        volatile int modifed;
        char buffer[128];
        if(argc ==1){
                printf("input more argument … \n");
                exit(1);
        }
        modified = 0;
        strcpy(buffer, argv[1]);

        if(modified == 0x66667562){
                printf("Good, You changed correctly!\n");
        }else{
                errx(1, "Please, Try Again. Don't give up.");
        }
}
```

과제 설명

❖ bof1.c 설명

- 다음과 같은 화면을 보이세요.



❖확인 내용

- 공격 수행에 작성한 payload를 보이세요.
- 공격 성공 화면을 보이고, 자신의 학번과 이름을 보이세요.

과제 설명

❖ bof2.c


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

void Necessary_Call()
{
	printf("You changed correctly the code flow");
}

void Non_Necessary_Call()
{
	printf("try again...");
}
```

```
int main(int argc, char **argv)
{
	volatile int (*fp)();
	volatile int canary;
	char buffer[256];

	fp = 0;
	canary = 0x61626364;

	gets(buffer);

	if(fp){
		printf("you changed fp pointer");
		if(canary==0x61626364){
			printf("calling function pointer, jumping to 0x%08x\n", fp);
			fp();
		}else{
			printf("This program was attacked. \n Cannary's value
0x%x", canary);
			exit(1);
		}
	}
}
```

과제 설명

❖ bof2.c 설명

- 다음과 같은 화면을 보이세요.

❖확인 내용

```
┌──(root㉿kali)-[/home/kali/Desktop/test]
└─# python bof2_python.py
[!] Could not find executable 'bof2' in $PATH, using './bof2' instead
[+] Starting local process './bof2': pid 28872
[*] Process './bof2' stopped with exit code 0 (pid 28872)
                                                          'You changed cor
rectly the code flow'

┌──(root㉿kali)-[/home/kali/Desktop/test]
└─# 32150000 MinsuPark
```

- Python Programming[pwntool 활용]을 활용하세요.
- 작성한 Python 프로그램 코드를 보여주세요.
- 공격 성공 시:“You changed correctly the code flow” 메시지 확인 가능.
- 공격 성공 화면을 보이고, 자신의 학번과 이름을 보이세요.

과제 설명

❖ bof3.c[보너스 문제]

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

int main(int argc, char **argv)
{
        /* Hint */
    // You have to run '/bin/sh'.
    // Find 'system' function address and '/bin/sh' address
    // If you found those addresses, Use pyhon programming.
    char buf[64];

    gets(buf);

    // If you have solved this problem, show your student ID and your name on the shell command
    // when you submit the assignment.
    // Ex) $32190000 Minsu Park
    // Additionally, show your python code.


}
```

과제 설명

❖ bof3.c 설명

- 다음과 같은 화면을 보이세요.

```
┌──(root㉿kali)-[/home/kali/Desktop/test]
└─# python bof3_python.py
[!] Could not find executable 'bof3' in $PATH, using './bof3' instead
[+] Starting local process './bof3': pid 610757
[*] Switching to interactive mode
$ pwd
/home/kali/Desktop/test
$ ls
bof1          bof3.c          peda-session-bof1.txt
bof1.c        bof3_python.py  peda-session-bof2.txt
bof2          core            peda-session-bof3.txt
bof2.c        Exercise        peda-session-bof4.txt
bof2_python.py  Exercise32    peda-session-Exercise32.txt
bof3          Exercise.c      peda-session-Exercise.txt
$ 32150000 MinsuPark
```

❖확인 내용

- Python Programming[pwntool 활용]을 활용하세요.
- 작성한 Python 프로그램 코드를 보여주세요.
- 공격 성공 시: 쉘 커맨드에서 리눅스 명령어 사용 가능.
- 공격 성공 화면을 보이고, 자신의 학번과 이름을 보이세요.
- 힌트: '/bin/sh', system 함수

04 과제 평가

과제 평가

❖ 보고서 작성(35점)

▪ 평가 항목

- 32bit vs 64bit 차이점 설명(5점)
  - 레지스터, 함수 호출 규약의 차이점을 설명하세요. (2.5점)
  - 내용 설명 시 예시 그림을 활용하세요. (2.5점)
- 문제 풀이[bof1.c, bof2.c](20점)
  - 각 문제 별 풀이 과정을 자세하게 기술하세요**[메모리 구조 그림 반드시 포함]**. (5점)
  - 공격에 사용된 Payload와 Python 코드를 보이세요. (5점)
  - 각 문제의 공격 성공 화면과 자신의 학번, 이름을 보이세요. (10점)
- 보너스 문제[bof3.c](10점)
  - bof3.c의 문제 풀이 과정을 자세하게 시술하세요**[메모리 구조 그림 반드시 포함]**.(2.5점)
  - 해당 프로그램에 적용된 메모리 보호 기법과 이를 우회할 수 있는 방안이 무엇이 있는지 서술하세요.(2.5점)
  - 공격에 사용된 Payload와 python 코드를 보이세요.(2.5점)
  - bof3.c의 공격 성공 화면과 자신의 학번, 이름을 보이세요.(2.5점)

과제 평가

❖보고서 제출 양식

- **보고서 표지 내용**
  - 과목명(운영 체제 보안), 분반 표시(1분반 또는 2분반)
  - 과제번호 및 제목(1번 과제: [Buffer Overflow 실습])
  - 성명, 학번
  - 제출일
- **과제 보고서 파일 이름**: " OS_sec(분반)_HW1_이름_학번_mmdd"
  - E.g) 1분반 홍길동(32150000), 제출 일이 10월 10일이면,
  - "OS_sec(1)_HW1_홍길동_1010"
  - 한글파일(.hwp) 또는 워드파일(.doc)로 제출 권장(PDF)파일도 가능
- **유의 사항[감점 요인 포함]**
  - **자신의 보고서에 대한 목차와 목차 별 페이지 표시할 것.**
  - **최소 10페이지 이상 작성**, 보고서 양이 부족한 경우 부족한 페이지 수 만큼 감점.
  - 개별적 실습 후 보고서 정리(**No Cheating**)
  - 신뢰할 만한 교재/자료/문서/논문/사이트를 활용

과제 평가

❖Deadline: 2022 10월 19일(수요일) 23:59분 59초까지 – 기간 엄수

❖제출 기간: 2022년 9월 28일 ~ 10월 19일 23시 59분 59초(3 주간)

- e-Campus의 “과제 및 평가” 메뉴 → “첫 번째 과제” 항목을 통해 제출
- 표절 비율(유사 비율)이 40% 이상이면, 감점될 수 있음. (표절 비율이 높을 수록 감점 비율이 더 높아짐)

❖질문 유의 사항

- 환경 구축에 대한 내용은 답변해드리지만, **문제 풀이 내용은 절대 받지 않습니다**.

Q&A