

정보보안론 5주차 강의

과제

1. BOF(Buffer Overflow) 코드의 문제점을 파악하고 보안을 적용하시오.
2. Check_password() 함수의 스택구조를 확인하시오.
3. 메모리해 툴을 이용하여 메모리값을 변조하도록 하시오.
4. 프로그램 진행방향을 조작하시오.

! 시험 ! password123을 넣었는데 correct~! 라고 뜨는 이유

메모리 세그먼트

Code Segment	: 프로그램 실행 코드가 저장되는 공간
Stack Segment	: 함수 호출 시 생성되는 스택 프레임이 저장되는 공간
Data Segment	: 전역 변수, static 변수 저장
Extra Segment	: 추가적으로 사용하는 세그먼트

명령어는 모두 Code Segment 로 보낸다.

변수는 모두 Stack Segment 로 보낸다.

- ➔ 지역변수
- ➔ Stack 정보
- ➔ Return Address
- ➔ Error 처리

문자열 상수는 Data Segment 로 보낸다.

(영역이 3가지로 나뉘어짐)

- ➔ rdata ; read Only
- ➔ data : read Write
- ➔ .bss : 함수 주소

프로그래머를 위해 비워둔 Extra Segment

Check_password() 함수의 스택구조를 확인

스택 프레임 구조

몇개더 ...				<- esp
		0	0	<- pw (지역 변수 - char 배열, 10B + padding)
0	0	0	0	
0	0	0	0	
flag = 0				<- 지역변수 할당
8byte EBP				
mem_ebp				<- 이전 함수의 ebp
return address				<- ebp
				<- ESP
main()				<- EBP
OS stack				

Return Address (복귀 주소) → 함수가 끝나면 돌아갈 위치

이전 EBP (saved base pointer) → 호출한 함수의 스택 프레임 기준점

flags = 0 (레지스터 상태) → 상태 플래그 저장

EBP (base pointer) → 스택 프레임 기준점

ESP (stack pointer) → 현재 스택의 맨 위를 가리킴

Main() 함수가 호출되면 스택 프레임이 생기고, 모든 스택에 밑 바닥에는 자동으로 Return Address가 만들어진다.

함수가 끝날 때 Return Address로 복귀한다.

BOF(Buffer Overflow) 코드의 문제점을 파악하고 보안을 적용

보안 적용 방법 1 : 보안 옵션이 있는 컴파일로 가는 방법 (visual c++ 에서 gets_s()를 사용 한다.)

보안 적용 방법 2 : 변수를 명확하게 써야한다.

보안 적용 방법 3 : 돌아 오는 값을 확실하게 비교 해야한다.

```
[*] hecktest.c
1  #include <stdio.h>
2
3  int check_password()
4  {
5      int flag=0;          // 버퍼오버플로우 발생
6      char pw[10] = "";
7
8      printf("input password : ");
9      gets(pw); // 보안 적용 방법 1 (보안 옵션이 있는 컴파일로 가는 것) : visual c++ 에서 gets_s()를 사용 하는 것을 고려
10
11     if( strcmp (pw,"password") == 0)
12     {
13         flag = 1;
14     }
15     else
16     {
17         // 보안 적용 방법 2 (변수를 명확하게 써야한다.) : 이 코드가 있어야 한다.
18         flag = 0; // ""
19     }
20
21     return flag;
22 }
23
24 int main()
25 {
26     if( check_password() == 1) // 보안 적용 방법 3 : 돌아 오는 값을 확실하게 비교 해야한다.
27     {
28         printf("Correct~!\n");
29     }
30     else
31     {
32         printf("Wrong~!\n");
33     }
34     getch();
35     return 0;
36 }
```

! 시험 - 보안 적용 방법 (위의 사진 참고)!

메모리핵 툴을 이용하여 메모리값을 변조

프로그램 진행방향을 조작

코드에 hack_function() 추가

```
hecktest.c
1  #include <stdio.h>
2
3  int hack_function()
4  {
5      printf("Hack Function~!!\n");
6  }
7
8
9  int check_password()
10 {
11     int flag=0; // 비파괴비파괴로 발생
12     char pw[10] = "";
13
14     printf("input password : ");
15     gets(pw); // 보안 적용 방법 1 (보안 문헌이있는 컴파일로 가는 것) : visual c++ 에서 gets_s()를 사용 하는 것을 고려
16
17     if( strcmp (pw,"password") == 0)
18         flag = 1;
19     else
20         flag = 0; // 보안 적용 방법 2 (변수를 명확하게 써야한다.) : 이 코드가 있어야 한다.
21
22     return flag;
23 }
24
25 int main()
26 {
27     if( check_password() == 1) // 보안 적용 방법 3 : 돌아 오는 값을 확실하게 비교 해야한다.
28     {
29         printf("Correct~!\n");
30     }
31     else
32     {
33         printf("Wrong~!\n");
34     }
35     getch();
36     return 0;
37 }
38
```

스냅샷 이용

push ebp 부터 ret 까지가 한 모듈 , 리턴 주소를 변경하면 hack_function()으로 가도록 할 수 있다.

hecktest.exe - PID: 15632 - 모듈: hecktest.exe - 스레드: 주 스레드 20064 - x32dbg

파일(F) 보기(V) 디버그(D) 추적(N) 플러그인(P) 즐겨찾기(I) 설정(O) 도움말(H) Aug 19 2025 (TitanEngine)

CPU 로그 메모 중단점 메모리 맵 호출 스택 SEH 스크립트 기록

EIP ECX EDX 004014E0 83EC 0C sub esp,c
004014E3 C705 34504000 00000000 mov dword ptr ds:[405034],0
004014E4 E8 3E0A0000 call hacktest.401F30
004014E5 83C4 0C add esp,c
004014E6 E9 86FCFFFF jmp hacktest.401180
004014E7 90 nop
004014E8 90 nop
004014E9 90 nop
004014EA 90 nop
004014EB 90 nop
004014EC 90 nop
004014ED 90 nop
004014EE 90 nop
004014EF 90 nop
004014F0 55 push ebp
004014F1 89E5 mov ebp,esp
004014F2 83EC 18 sub esp,18
004014F3 C70424 00404000 mov dword ptr ss:[esp],hecktes
004014F4 E8 76110000 call <JMP.&puts>
004014F5 C9 leave
004014F6 C3 ret
004014F7 55 push ebp
004014F8 89E5 mov ebp,esp

FPU 숨기기

EAX 0062FFCC
EBX 002D4000
ECX 004014E0 <hecktest.Opt
EDX 004014E0 <hecktest.Opt
EBP 0062FF84 "t15u"
ESP 0062FF78 <hecktest.Opt
ESI 004014E0 <hecktest.Opt
EDI 004014E0 <hecktest.Opt
EIP 004014E0 <hecktest.Opt

EFLAGS 00000244
ZF 1 PF 1 AF 0

기본값 (stdcall) 5 : ☐ 잠금 해제

1: [esp+4] 002D4000 002D4000
2: [esp+8] 75355D30 <kernel32.BaseThread
3: [esp+C] 0062FFDC 0062FFDC
4: [esp+10] 7769D6D8 ntdll.RtlInitializeExce
5: [esp+14] 002D4000 002D4000

주소 Hex 77631000 14 00 16 00 18 DE 63 77 08 00 0A 00 F8 DD 63 77
77631001 00 00 02 00 B4 3E 63 77 18 00 1A 00 20 DE 63 77
77631002 1C 00 1E 00 12 DE 63 77 02 00 04 00 E0 DD 63 77
77631003 60 03 69 77 10 76 68 77 00 00 00 00 E0 EF 73 77
77631004 80 D0 68 77 10 76 68 77 60 EF 73 77 E0 EF 73 77
77631005 00 00 00 00 57 14 01 E2 46 15 C5 43 A5 FE 00 8D
77631006 EE E3 D3 F0 06 00 00 00 A4 DC 63 77 01 00 00 00
77631007 9A 88 13 35 96 50 8D 4E 8E 2D 62 44 02 25 F9 3A
77631008 06 00 01 00 88 DC 63 77 02 00 00 00 E3 28 2F 4A
77631009 89 53 41 44 BA 9C 06 9D 1A 4A 6E 38 06 00 02 00
7763100A 8C DC 63 77 03 00 00 00 76 6C 6F 1F E1 80 39 42
7763100B 95 88 83 D0 F6 D0 DA 78 06 00 03 00 50 DC 63 77
7763100C 04 00 00 12 0A 0F 8E B3 BF E8 4F B9 A5 48 F0
7763100D 50 A1 5A 9A 0A 00 00 00 C0 DC 63 77 00 39 6A 77
7763100E 00 F0 73 77 60 D8 63 77 80 EE 73 77 00 53 67 77
7763100F 00 F0 73 77 10 76 68 77 60 EF 73 77 03 00 00 00
77631010 60 EF 73 77 A0 78 68 77 60 F0 73 77 E0 19 6A 77
77631011 60 EF 73 77 A0 78 68 77 60 F0 73 77 E0 19 6A 77

주석 1 [x] 로컬 [x] 스택

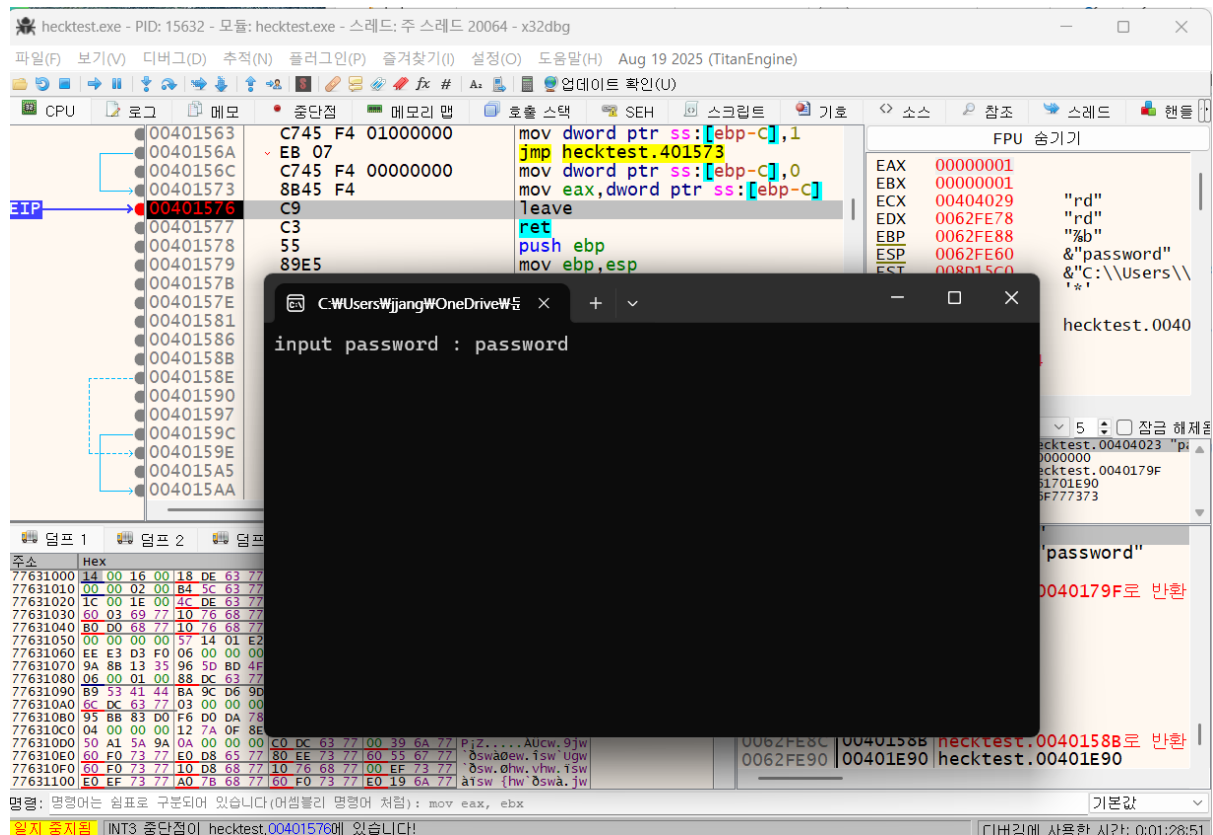
0062FF78 75355D49 kernel32.BaseThreadInit
0062FF7C 002D4000
0062FF80 75355D30 kernel32.BaseThreadInit
0062FF84 0062FFDC
0062FF88 7769D6D8
0062FF8C ntdll.RtlInitializeExce
0062FF90 8EC7EABD
0062FF94 00000000
0062FF98 00000000
0062FF9C 00000000
0062FFA0 00000000
0062FFA4 00000000
0062FFA8 00000000

명령: 명령어는 실행이 완료되어 있습니다(어셈블리 명령어 처리): mov eax, ebx

현재 중지됨 [00401578]에 중단점을 설정하였습니다!

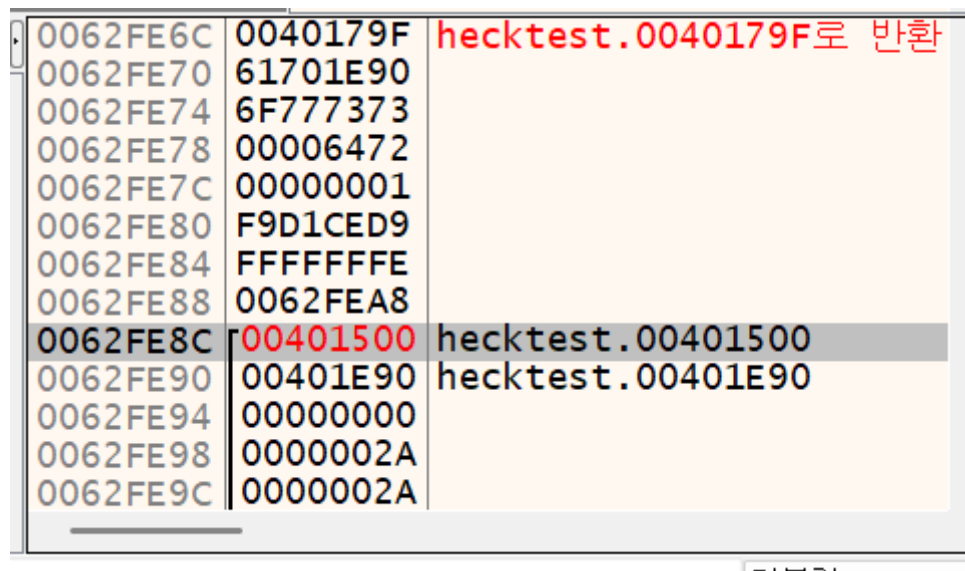
디버깅에 사용한 시간: 0:01:27:44

hack_function() 끝나는 곳에 break 걸기



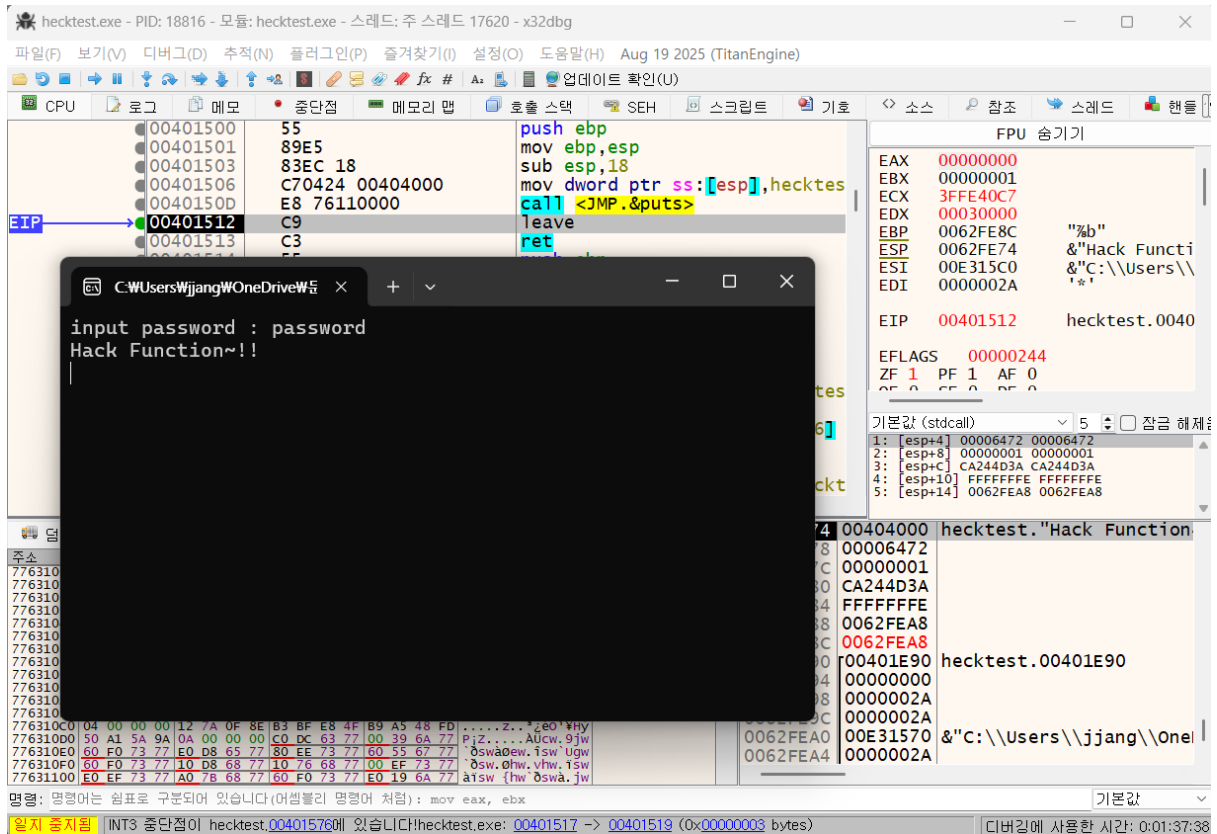
EBP(마지막 스택) 주소 확인 -> 00401500

메모리값 변조 -> 0040158B를 00401500으로 뒤에 두개를 변경



프로그램 진행방향 조작

leave : 파괴하는 명령 -> F8 입력



! 시험 - 스택 영역과 힙 영역 !

스택 영역 -> 지역변수, Stack 정보, Return Address, Error 처리

프로그램 로직이 동작하기 위한 인자와 프로세스 상태를 저장하는 데 사용.

레지스터의 임시 저장, 서브루틴 사용 시 복귀 주소저장, 서브루틴에 인자 전달 등에 사용.

힙 영역

프로그램이 동작할 때 필요한 데이터 정보를 임시로 저장하는 데 사용

프로그램이 실행될 때까지 알 수 없는 가변적인 양의 데이터를 저장하기 위해 프로그램 프로세스가 사용할 수 있도록 미리 예약된 메인 메모리 영역.

힙 영역은 프로그램에 의해 할당되었다가 회수되는 작용을 되풀이함

힙의 기억 장소는 포인터 변수를 통해 동적으로 할당받고 돌려주며, 연결 목록이나 나무, 그래프 등의 동적인 데이터 구조를 만드는 데 반드시 필요함

프로그램 실행 중에 해당 힙 영역이 없어지면 메모리 부족으로 이상 종료

! 시험 - 각각의 레지스터의 용도 !

레지스터 : CPU의 임시 메모리로 CPU 연산과 어셈블리어의 동작에 필요

표 5-1 80x86 CPU의 레지스터

범주	80386 레지스터	이름	비트	용도
범용 레지스터	EAX	누산기(accumulator)	32	산술 연산에 사용(함수의 결과 값 저장)
	EBX	베이스 레지스터(base register)	32	특정 주소 저장(주소 지정을 확대하기 위한 인덱스로 사용)
	ECX	카운트 레지스터(count register)	32	반복적으로 실행되는 특정 명령에 사용(루프의 반복 횟수나 좌우 방향 시프트 비트 수 기억)
	EDX	데이터 레지스터(data register)	32	일반 데이터 저장(입출력 동작에 사용)
세그먼트 레지스터	CS	코드 세그먼트 레지스터 (code segment register)	16	실행 기계 명령어가 저장된 메모리 주소 지정
	DS	데이터 세그먼트 레지스터 (data segment register)	16	프로그램에서 정의된 데이터, 상수, 작업 영역의 메모리 주소 지정
	SS	스택 세그먼트 레지스터 (stack segment register)	16	프로그램이 임시로 저장할 필요가 있거나 사용자의 피호출 서브루틴이 사용할 데이터와 주소 포함
	ES, FS, GS	엑스트라 세그먼트 레지스터 (extra segment register)	16	문자 연산과 추가 메모리 지정에 사용되는 여분의 레지스터
포인터 레지스터	EBP	베이스 포인터(base pointer)	32	SS 레지스터와 함께 스택 내의 변수값을 읽는데 사용
	ESP	스택 포인터(stack pointer)	32	SS 레지스터와 함께 스택의 가장 끝 주소를 가리킴
	EIP	명령 포인터(instruction pointer)	32	다음 명령어의 오프셋(상대 위치 주소)을 저장하며, CS 레지스터와 합쳐져 다음에 수행될 명령의 주소 형성
범주	80386 레지스터	이름	비트	용도
인덱스 레지스터	EDI	목적지 인덱스(destination index)	32	목적지 주소의 값 저장
	ESI	출발지 인덱스(source index)	32	출발지 주소의 값 저장
플래그 레지스터	EFLAGS	플래그 레지스터(flag register)	32	연산 결과 및 시스템 상태와 관련된 여러 가지 플래그 값 저장

프로그램 실행 구조

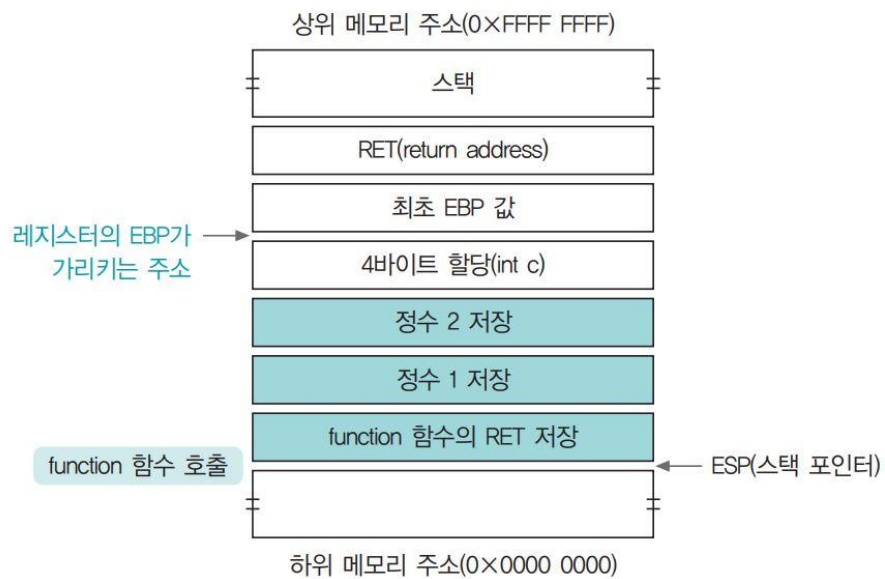


그림 5-6 pushl \$2, pushl \$1, call function 실행 시 스택의 구조

! 시험 - 버퍼 오버플로 공격의 대응책 !

버퍼 오버플로에 취약한 함수 불사용

strcpy(char *dest, const char *src);

strcat(char *dest, const char *src);

getwd(char *buf);

gets(char *s);

fscanf(FILE *stream, const char *format, ...);

scanf(const char *format, ...);

realpath(char *path, char resolved_path[]);

sprintf(char *str, const char *format);