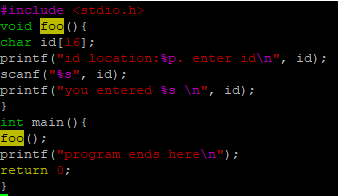
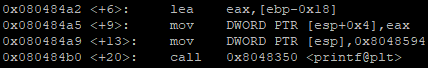
1. Change f2.c as follows. What is the location of id[] inside gdb? What is the real location of id[] during the execution? How far is id[] from the stack location where the return address for "foo" is stored?

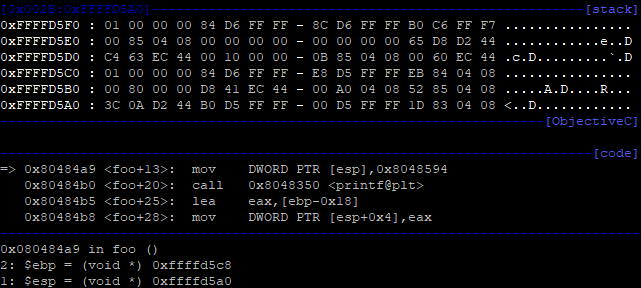
f2.c





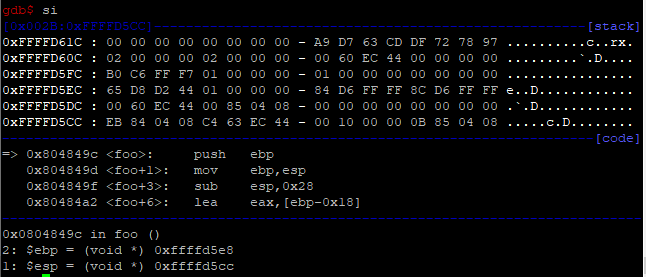
id의 실제 위치는 변경한 f2를 실행시켰을 때 알 수 있다. 실행 결과, id의 실제 위치는 0xffffd5e0에 있다고 나온다.





lea eax,[ebp-0x18]을 한 후 esp+0x4에는 id의 gdb 주소가 들어있다. 이때 esp+0x4는 ffffd5a4인데, 그 위치에서 스택을 확인해보면 ffffd5b0이 있고 이것이 id의 gdb 주소란걸 알 수 있다.

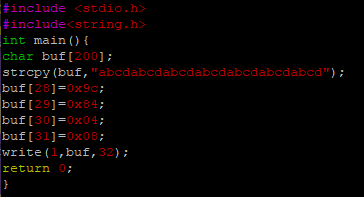
.



위 그림을 보면 ffffd5cc의 위치에 foo의 리턴주소인 80484eb가 들어있기 때문에 foo의 리턴주소가 저장되는 stack location은 ffffd5cc라는걸 알 수 있다. id의 gdb 주소인 ffffd5b0과 ffffd5cc의 차이는 10진수로 28만큼 난다.

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

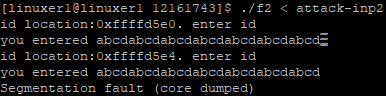
inp-write.c





enter id를 두번 입력하는 attack 파일을 만들려면 기존 id크기인 24바이트와 old ebp의 크기인 4바이트를 합쳐서 28바이트는 랜덤으로 입력해주고, 그 뒤에 마지막 4바이트는 foo의 시작주소를 넣어서 foo가 다시 실행되도록 해야한다. 위 캡처본에서 foo의 시작주소는 804849c라는 것을 알 수 있다.

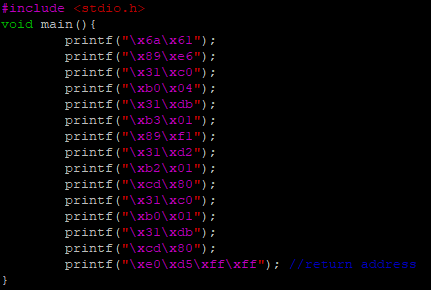




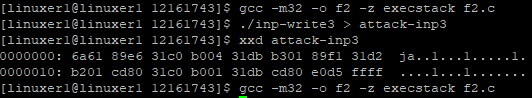
컴파일 후 실행해본 결과, enter id를 두번 출력하는데 성공했다.

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 0x20 (blank) or 0x00 (end of string), so make sure your attack-inp does not contain either of these. If it contains them, you should change f2.c to avoid them

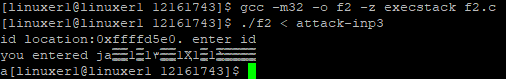
inp-write3.c



id를 입력하는 부분에 어느 한 쓰레기값 대신 machine instruction으로 채우고 리턴 주소에는 id의 실제 시작주소를 넣으면, id에 저장돼있는 프로그램을 실행할 수 있게 된다. 이걸 사용해서 id에는 a를 출력하는 machine instruction을 넣고, 리턴 주소에는 id의 실제 시작주소인 ffffd5e0을 넣는다.



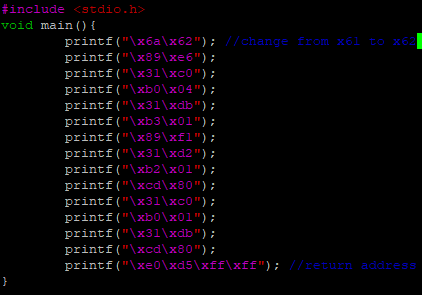
xxd를 사용해서 0x00과 0x20이 없는 것을 확인했다. (만약 있다면 id사이즈를 16에서 17이나 다른걸로 바꿔줌.)

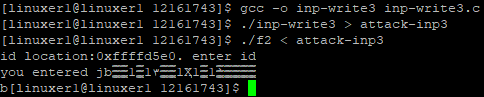


putty 시스템은 buffer overflow를 기본적으로 막아주고있기 때문에 stack 영역에서의 실행이 불가능하게 한다. 이 때 stack에서의 실행을 가능하게 해주는 것이 -z execstack이고, 이걸 사용해서 f2.c를 컴파일 한 후 attack-inp3을 사용해서 f2를 attack하면, attack이 성공하여 a가 정상적으로 출력되는걸 확인할 수 있다.

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

inp-write3.c

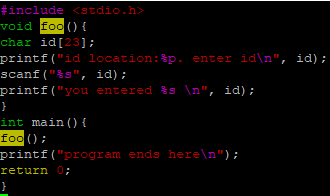


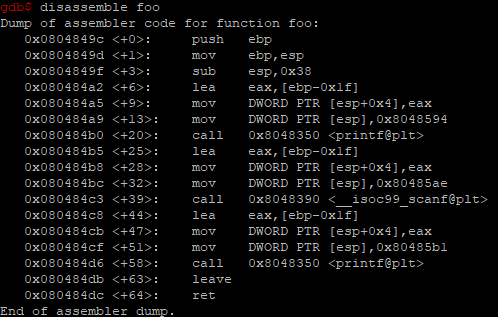


inp-write3.c를 고치기 전에 \x6a\x61이 원래 push 0x61라는 instruction을 나타내고, 여기에서 0x61이 바로 write(1,”a”,1); 에서의 a를 가리키는데, 이 때 x61을 x62로 바꿔주면 a 대신 b를 출력할 수 있게 된다.

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

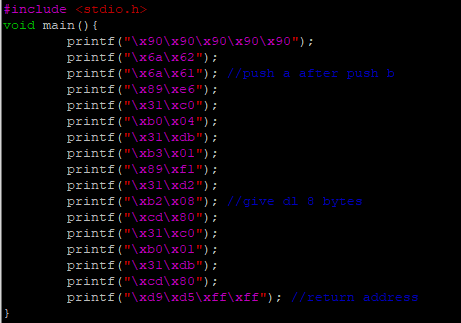
f2.c



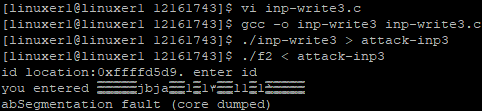


id 공간이 모자랄 것을 생각하여 f2.c에서 id를 id[16]에서 id[23]으로 바꿔주었다. 그 결과 id는 0x1f, 즉 31만큼의 크기를 갖게 돼었고, old ebp인 4바이트까지 포함하면 id부터 리턴 주소 전까지 총 35바이트의 공간을 갖게된다.

inp-write3.c



a를 push 한 다음에 b를 push 해줘야하기 때문에 printf(“\x6a\x62”)를 두번째 줄에 추가해줬다. inp-write3.c에서 10번째 줄이 바로 mov dl, 1 명령어고, 이것은 하나의 character 만큼 dl에 공간을 제공해준다는 뜻이다. 앞에서처럼 캐릭터를 a 하나만 push 하면 사실상 스택에는 61만 들어가는게 아니라 00 00 00 61로 총 네개의 캐릭터가 들어가게된다. 이 때 esp는 원래 리턴주소가 끝나는 곳에 있었고, push를 하면서 esp는 4바이트 앞당겨져 오기 때문에 a가 들어가게 되는 곳은 리턴주소가 된다. 이렇게 리턴주소에 영향을 미치는데 까지는 문제가 없지만, 문제는 여기에서 b를 더 push하게 되면 스택 안에는 00 00 00 62 00 00 00 61로 8개의 캐릭터가 들어가게 되는데, 이렇게 되면 리턴주소를 넘어선 공간에까지 영향을 미치게 된다. 그렇기 때문에 ab를 정상적으로 출력하기 위해선 dl에 캐릭터 하나가 아니라 캐릭터 8개만큼 공간을 제공해줘야 한다. 그리고 앞에서 id 배열을 16에서 23으로바꿔줘서 id부터 리턴주소까지의 거리가 28에서 35로 바뀌었지만, inp-write3.c에서 machine instruction으로 채워진건 30바이트밖에 안되기 때문에 맨 앞줄에 아무것도 수행하지 않는 nop instruction인 0x90을 5개 추가해줘서 35바이트를 채웠다.



ab가 정상적으로 출력됐다.