**BLG 460E SECURE PROGRAMMING**

HW1

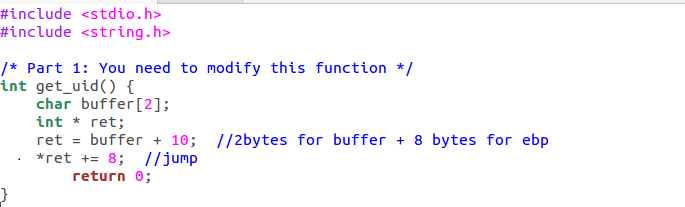
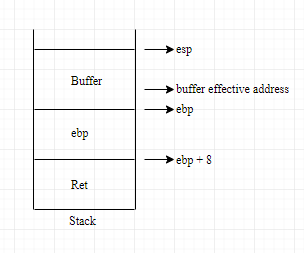
**Ahmet Göktuğ SEVİNÇ**

**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*.



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

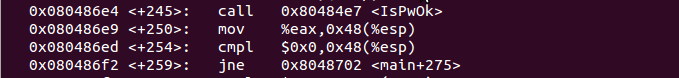
to the second instruction of the *uid = default\_uid*; operation.

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.

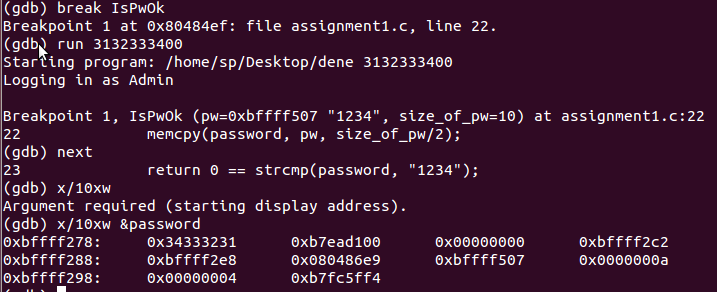
If I run my program without *–fno-stack-protector*, it does not jump to the calculated address and *uid* gets the value of *default\_uid*. That means, the return address in stack is somehow protected and even though we try to modify, it protects its contents.

**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.



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.



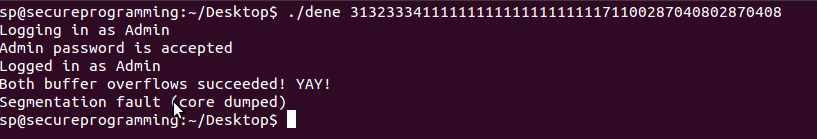
As seen in above image, to overwrite the return address, first of all, I put a break point to the *IsPwOK()* function and run the program with default argument of *3132333400*. After the break, I inspected memory parts starting from address of *password* by using *x/10wx* command. Then, I detected memory 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**11111111111111111111711002870408**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.



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.